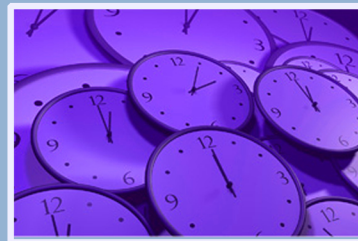


# PATAT 2012

9th International Conference on the Practice and Theory of Automated Timetabling  
Son, Norway, Tues. 28<sup>th</sup> - Fri. 31<sup>st</sup> August 2012



**CONFERENCE PROCEEDINGS**

# PATAT 2012

Proceedings of the 9<sup>th</sup> International Conference on the  
Practice and Theory of Automated Timetabling

29 - 31 August 2012, Son, Norway

Edited by:

Dag Kjenstad, SINTEF ICT, Norway

Atle Riise, SINTEF ICT, Norway

Tomas Eric Nordlander, SINTEF ICT, Norway

Barry McCollum, Queen's University Belfast, UK

Edmund Burke, University of Nottingham, UK

ISBN 978-82-14-05298-5

Published by SINTEF

## Preface

The International Series of Conferences on the Practice and Theory of Automated Timetabling (PATAT) is held bi-annually as a forum for both researchers and practitioners to exchange experience and ideas. I am delighted to welcome you to the 9<sup>th</sup> conference here at Quality Spa & Resort in Son, Norway.

The conference is organized in sessions covering the topics of Admission and Surgery Scheduling, Nurse Rostering, High School Timetabling, University Course Timetabling, Examination Timetabling, Generalized Timetabling, Personnel Rostering, Task Scheduling and Sports Scheduling. The fusion of research and practice is a central theme of PATAT. Five separate sessions are therefore devoted to the practice of timetabling, and researchers, software vendors and practitioners alike will be able to give presentations and participate in discussions on relevant timetabling issues. A separate session will be devoted to the 3<sup>rd</sup> International Timetabling Competition, ICT 2011. This year the competition focuses on the field of High School Timetabling, and final results will be presented at the conference. Altogether the program features 63 presentations which represent the state-of-the-art in automated timetabling: 4 plenary presentations, 20 full papers, 26 extended abstracts, 5 system demos, 5 extended abstracts from ITC 2011, and 8 keynote practitioner talks.

As was the case in the two preceding conferences, a post-conference volume of selected and revised papers is to be published in a Special Issue of the Annals of Operations Research. Authors of full papers and extended abstracts are encouraged to submit full papers to this special issue after the conference.

I would like to express my gratitude to the many individuals who have helped organize this conference; the members of the Steering Committee who continue to ensure the ongoing success of the series; the members of the Program Committee who have worked hard to referee the conference submissions; our sponsors who have helped fund student participation, local transportation and invited presentations from renowned plenary speakers; and as always we are most grateful to all authors and delegates.

Finally I would like to thank SINTEF for hosting this year's conference. SINTEF is among Europe's largest contract research organizations and a non-profit organization with 2100 employees from 69 countries. Special thanks go to Janet Skallerud, Atle Riise and Tomas Nordlander in the local organizing committee for help and support in preparing this conference to the highest possible standard. I wish you an enjoyable time in Son and a successful conference for all.

Welcome to the conference!



Dag Kjenstad

## PATAT 2012 Conference Program Committee

Salwani Abdullah	Universiti Kebangsaan Malaysia, Malaysia
Panayiotis Alefragis	TEI of Mesolonghi, Greece
Hesham Alfares	King Fahd University, Saudi Arabia
Viktor Bardadym	Trasys, Belgium
Peter Brucker	University of Osnabrück, Germany
Peter Cowling	University of York, UK
Patrick De Causmaecker	Katholieke Universiteit Leuven, Belgium
Kathryn Dowsland	Gower Optimal Algorithms Ltd. UK
Wilhelm Erben	University of Applied Sciences, Germany
Luca Di Gaspero	Università di Udine, Italy
Michel Gendreau	Université de Montréal, Canada
Alain Hertz	Ecole Polytechnique de Montréal, Canada
Graham Kendall	University of Nottingham, UK
Jeffrey Kingston	University of Sydney, Australia
Raymond Kwan	University of Leeds, UK
Gilbert Laporte	HEC Montréal, Canada
Rhyd Lewis	Cardiff University, UK
Arne Løkketangen	Molde University College, Norway
Amnon Meisels	Ben-Gurion University, Beer-Sheva, Israel
Paul McMullan	Queen's University of Belfast
Keith Murray	Purdue University, USA
Tomáš Müller	Charles University Prague, Czech Republic
Ender Ozcan	University of Nottingham, UK
Ben Paechter	Napier University, UK
Gilles Pesant	Ecole Polytechnique de Montréal, Canada
Sanja Petrovic	University of Nottingham, UK
Nelishia Pillay	University of KwaZulu-Natal, South Africa
Gerhard Post	University of Twente, The Netherlands
Jean-Yves Potvin	Université de Montréal, Canada
Rong Qu	University of Nottingham, UK
Louis-Martin Rousseau	Ecole Polytechnique de Montréal, Canada
Celso C. Ribeiro	Universidade Federal Fluminense, Brazil
Hana Rudova	Masaryk University, Czech Republic
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
Pascal Van Hentenryck	Brown University, USA
Greet Vanden Berghe	KaHo St.-Lieven, Belgium
Stefan Voss	University of Hamburg, Germany
Dominique de Werra	EPF-Lausanne, Switzerland
George White	University of Ottawa, Canada
Michael Wright	Lancaster University, UK
Jay Yellen	Rollins College, USA



## Table of Contents

### Plenary Presentations

Personnel scheduling - Challenging combinatorial optimisation problems with a personnel scheduling component.....	10
<i>Greet Vanden Berghe</i>	

### Full Papers

Repairing High School Timetables with Polymorphic Ejection Chains.....	16
<i>Dr. Jeffrey H. Kingston</i>	
Application of a parallel computational approach in the design methodology for the Course timetabling problem .....	31
<i>Jorge Alberto Soria-Alcaraz, Martin Carpio, Héctor J. Puga and Marco Sotelo-Figueroa</i>	
A Combined Local Search and Integer Programming Approach to the Traveling Tournament Problem.....	45
<i>Marc Goerigk and Stephan Westphal</i>	
Real-life Curriculum-based Timetabling .....	57
<i>Tomáš Müller and Hana Rudová</i>	
Schedule Pattern: an Innovative Approach to Structuring Time in Secondary Schools Scheduling .....	73
<i>Baiyun Tao and Rick Dwyer</i>	
Using the PEAST Algorithm to Roster Nurses in an Intensive-Care Unit in a Finnish Hospital.....	83
<i>Nico Kyngäs, Kimmo Nurmi, Eyjólfur Ingi Ásgeirsson and Jari Kyngäs</i>	
A Tour Scheduling Problem with Fixed Jobs: use of Constraint Programming .....	94
<i>Tanguy Lapègue, Damien Prot and Odile Bellenguez-Morineau</i>	

Fairness in Academic Course Timetabling .....	114
<i>Moritz Mühlenthaler and Rolf Wanka</i>	
The effect of neighborhood structures on examination timetabling with artificial bee colony .....	131
<i>Bolaji Asaju Laaro, Ahamad Tajudin Khader, Mohammed Azmi Al-Betar, Mohammed Awadallah and J.Joshua Thomas</i>	
A matheuristic approach to the shift minimisation personnel task scheduling problem.....	145
<i>Pieter Smet and Greet Vanden Berghe</i>	
A stepping horizon view on nurse rostering .....	161
<i>Fabio Salassa and Greet Vanden Berghe</i>	
The Patrol Scheduling Problem.....	175
<i>Hoong Chuin Lau and Aldy Gunawan</i>	
Patient-to-room assignment planning in a dynamic context .....	193
<i>Wim Vancroonenburg, Patrick De Causmaecker and Greet Vanden Berghe</i>	
Decomposing the High School Timetable Problem .....	209
<i>Christos Valouxis, Christos Gogos, Panayiotis Alefragis and Efthymios Housos</i>	
Near-Optimal MIP Solutions for Preference Based Self-Scheduling .....	222
<i>Eyjólfur Ingi Ásgeirsson and Guðríður Lilla Sigurðardóttir</i>	
Application of Particle Swarm Optimization to the British Telecom Workforce Scheduling Problem.....	242
<i>Maik Günther and Volker Nissen</i>	
Integer Programming Techniques for the Nurse Rostering Problem .....	257
<i>Haroldo Gambini Santos, Túlio Toffolo, Sabir Ribas and Rafael Gomes</i>	
A Survey on Workforce Scheduling and Routing Problems .....	283
<i>J. Arturo Castillo-Salazar, Dario Landa-Silva and Rong Qu</i>	
A Constraint Programming Approach to the Traveling Tournament Problem with Predefined Venues .....	303
<i>Gilles Pesant</i>	
Hyper-Heuristics for Educational Timetabling.....	316
<i>Nelishia Pillay</i>	

## Extended Abstracts

Scheduling the Brazilian Football Tournament in Practice.....	342
<i>Celso Ribeiro and Sebastián Urrutia</i>	
Scheduling Cricket Umpires Using Neighbourhood Search – The Dramatic Impact of a Simple Change in Neighbourhood Definition .....	346
<i>Mike Wright</i>	
A Column Generation Approach for Solving the Patient Admission Scheduling Problem .	349
<i>Troels Martin Range, Richard Lusby and Jesper Larsen</i>	
Timetabling and field assignment for training youth football teams in amateur leagues....	352
<i>Celso Ribeiro, Renatha Capua and Simone Martins</i>	
Rostering RAF Air Traffic Control Personnel .....	355
<i>Richard Conniss, Tim Curtois, Sanja Petrovic and Edmund Burke</i>	
High School Timetabling: Modeling and solving a large number of cases in Denmark .....	359
<i>Matias Sørensen and Thomas Stidsen</i>	
Adaptive large neighborhood search for student sectioning at Danish high schools .....	365
<i>Simon Kristiansen and Thomas Stidsen</i>	
Investigation of fairness measures for nurse rostering .....	369
<i>Pieter Smet, Simon Martin, Djamila Ouelhadj, Ender Özcan and Greet Vanden Berghe</i>	
Patient Admission Scheduling with Operating Room Constraints .....	373
<i>Sara Ceschia and Andrea Schaerf</i>	
Timetabling of sorting slots in a logistic warehouse.....	377
<i>Antoine Jouget, Dritan Nace and Christophe Outteryck</i>	
Days off scheduling - A 2-phase approach to personnel rostering.....	381
<i>Sophie van Veldhoven, Gerhard Post and Egbert van der Veen</i>	
Self-Rostering applied to case studies .....	384
<i>Suzanne Uijland, Egbert van der Veen, Johann Hurink and Marco Schutten</i>	
Towards Fair and Efficient Assignments of Students to Projects .....	388
<i>Marco Chiarandini, Rolf Fagerberg and Stefano Gualandi</i>	
Regulation-based University Course Timetabling .....	391
<i>Michael Zeising and Stefan Jablonski</i>	

Co-evolving add and delete heuristics.....	395
<i>Jerry Swan, Ender Ozcan and Graham Kendall</i>	
Semidefinite Programming Relaxations in Timetabling II: Algorithms (Abstract).....	400
<i>Jakub Marecek and Andrew J. Parkes</i>	
A GRASP Algorithm for the University Timetabling Problem.....	404
<i>Walace Rocha, Maria Boeres and Maria Rangel</i>	
Using integer linear programming methods for optimizing the real-time pump scheduling	407
<i>Louise Brac De La Perriere, Antoine Jouglet, Alexandre Nace and Dritan Nace</i>	
A study of hyper-heuristics for examination timetabling .....	410
<i>Ender Özcan, Anas Elhag and Viral Shah</i>	
Nurse Timetabling: Linking Research and Practice .....	415
<i>Barry McCollum</i>	
Next Steps for the Examination Timetabling Format and Competition .....	418
<i>Barry McCollum, Paul McMullan, Tomas Muller and Andrew J. Parkes</i>	
Directing selection within an extended great deluge optimisation algorithm.....	421
<i>Ryan Hamilton-Bryce, Paul McMullan and Barry McCollum</i>	
Academic Timetabling: Space Sharing Strategies .....	423
<i>Dr Barry McCollum</i>	
A Hybrid Evolutionary Algorithm for the Generalized Surgery Scheduling Problem.....	426
<i>Atle Riise, Carlo Mannino and Edmund K. Burke</i>	
An exact decomposition approach for the optimal real-time train rescheduling problem ...	428
<i>Carlo Mannino and Leonardo Lamorgese</i>	
A real world personnel rostering problem with complex objectives.....	433
<i>Oddvar Kloster</i>	

## System Demonstrations

An Intelligent, Interactive & Efficient Exam Scheduling System (IIEESS v1.0) .....	437
<i>Chun Bao Zhu and Tha Nu</i>	

Aspen Scheduler: a Web-based Automated Master Schedule Builder for Secondary Schools .....	451
<i>Baiyun Tao and Rick Dwyer</i>	
A Open source timetable production system for courses and exams .....	460
<i>Ruben Gonzalez-Rubio and Balkrishna Sharma Gukhool</i>	
School Time Tabling ITT software.....	466
<i>Ilana Cohen-Zamir and Doron Bar</i>	

## The Third International Timetabling Competition

The Third International Timetabling Competition .....	479
<i>Gerhard Post, Luca Di Gaspero, Dr. Jeffrey H. Kingston, Barry McCollum and Andrea Schaerf</i>	
An Evolutionary Algorithm for High School Timetabling.....	485
<i>Jonathan Domrös and Jörg Homberger</i>	
An Adaptive Large Neighborhood Search algorithm.....	489
<i>Matias Sørensen, Simon Kristiansen and Thomas K. Stidsen</i>	
A SA-ILS approach for the High School Timetabling Problem.....	493
<i>George Henrique Godim Da Fonseca, Haroldo Gambini Santos, Túlio Ângelo Machado Toffolo, Samuel Souza Brito and Marccone Jamilson Freitas Souza</i>	
HySST: Hyper-heuristic Search Strategies and Timetabling .....	497
<i>Ahmed Kheiri, Ender Ozcan and Andrew Parkes</i>	

# Plenary Presentations

## Personnel scheduling

### Challenging combinatorial optimisation problems with a personnel scheduling component

Greet Vanden Berghe

**Abstract** Personnel scheduling can become a particularly difficult optimisation problem due to human factors. And yet: people working in healthcare, transportation and other round the clock service regimes perform their duties based on a schedule that was often manually constructed. The unrewarding manual scheduling task deserves more attention from the timetabling community so as to support computation of fair and good quality results. The present abstract touches upon a set of particular characteristics of personnel rostering problems for which, for the time being, only very scattered models and algorithms exist.

Besides being hard to solve, personnel scheduling never occurs as an isolated problem in real life. The interconnectedness of personnel scheduling and other vertical decision levels of the organisation constitutes the second focus of this extended abstract. Not only is it difficult to produce an acceptable solution to a personnel rostering problem, it is also cumbersome to detect possible infeasibilities or conflicting constraints, caused by decisions at a higher level than the scheduling level. Part of the contribution is dedicated to mutual parameters at the manpower, staffing and rostering level.

Next to the vertical influences, personnel scheduling cannot be ignored as an optimisation problem that is influenced by other optimisation problems in an organisation, e.g. patient admission scheduling, operating theatre scheduling and personnel scheduling cannot really be solved independently. Another interesting set of problems consists of strongly intertwined personnel rostering and other problems, such as vehicle routing and rostering combined in the home care scheduling problem.

---

G. Vanden Berghe  
KAHO Sint-Lieven, Computer Science, CODES, Gebr. De Smetstraat 1, 9000 Gent, Belgium  
KU Leuven, Department of Computer Science, E. Sabbelaan 53, 8500 Kortrijk, Belgium  
Tel.: +32-9-2658610  
Fax: +32-9-2256269  
E-mail: greet.vandenberghel@kahosl.be

Each separate decision level includes challenging research questions and opportunities. Large margins for improvement exist when crossing the borders of decision levels or optimisation problems interfering with personnel scheduling.

**Keywords** Personnel scheduling · automated rostering · decision levels

## 1 Introduction

Personnel scheduling is a relevant logistic problem in healthcare, transportation and the service industry [10]. It covers a wide range of optimisation problems [4], most of them dealing with tasks or shifts that need to be covered by a team of people over a given planning horizon. Manual planners as well as scientists optimising schedules generally agree that personnel scheduling is a difficult problem to solve. Some characteristics, including personnel's skills, round the clock work, large sets of contractual and individual constraints contribute to the hardness of the problem.

The academic side of personnel scheduling reveals a rather well-established combinatorial optimisation problem, while only little supporting theory is available. The actual problem definitions and constraints differ considerably among academic papers [7], which has led to a very large variety of models and algorithms, e.g. [2, 3, 8, 12, 15, 16, 19, 20].

Personnel scheduling offers plenty of challenges to the timetabling community, both in terms of theory and application development.

## 2 Manual versus automated scheduling

Despite many years of excellent advancements in personnel scheduling research, only little results have made it to practical decision support systems [13]. This is rather unfortunate for organisations spending a large budget on human resources and, in particular, for the planners who are responsible for generating weekly or monthly rosters without advanced software systems.

Many explanations can be given for the lack of effective decision support. Addressing the shortcomings probably requires multi-disciplinary approaches, rather than faster or better optimisation algorithms. The main research challenges include:

- precisely capturing the actual coverage needs, the actual meaning of required skills and experience, time definitions, etc. Manual planners are inclined to controlling out every possible detail of the scheduling problem. How much detail should be contained in the model for automated decision support?
- correctly interpreting the complex hard and soft constraints. Working time regulations that have been interpreted in contradictory ways within one single organisation are very common indeed. How should they be modelled in an automated system?



- capturing information on the perceived importance of constraints and objectives. The implicit ranking of constraints and objectives differs surprisingly often from the official ranking. This may lead to unwanted situations in which the personnel prefers a low quality schedule over a high quality one because of a different quality perception. Should the implicit ranking be ignored? If not, how can it be determined?
- accurately modelling the human factors [17]. The planner's expertise concerning fairness among members of staff, the actual skill level of people, the private life of individuals, the personal preferences and concerns, etc. cannot be ignored. This implicit information is again very hard to capture.
- developing mechanisms for dealing with unpredictable issues. The need for quick rescheduling in case of unexpected staff shortage has been studied, e.g. [18]. Which other urgent decision or optimisation problems cannot be ignored in the context of automated decision support?
- implement state of the art personnel scheduling approaches within central software systems of the organisation, if such systems exist at all. How should the new approach be implemented, integrated and maintained?

### 3 Personnel scheduling and other decision levels

#### 3.1 Vertical decision levels

Personnel scheduling is subject to constraints set at other decision levels. One of the most restrictive constraints is determined by the estimated workload per hour, shift, day and department. It is quite common that the corresponding coverage constraints are unsatisfiable, given the available people, their contracts and the set of regulations. Eventual schedules executed in practice are indeed not always feasible.

Without going into methodologies for accurately determining the workload, the estimations cannot be made independently from assumptions on the available members of staff per department. The number of personnel is one issue but their contracts and skills are at least as important.

The individual members of personnel have been assigned to each department at the staffing level. These decisions should ideally be influenced by the potential subsequent schedule quality. Unfortunately, there exists no analytical relationship between the personnel composition and the schedule quality, not even when the workload is precisely known. One possible decision aid is to compute personnel schedules for varying compositions of personnel. The outcome may help to select particular contract and skill mixes for staffing each department.

The management should be concerned about manpower planning, which is a long term organisation wide decision to be made. Besides determining how many people the organisation should hire at a certain point in time, it should also consider which skill and age mix is required and which mechanisms, e.g. recruitment, training, dismissal, are necessary to attain this staff composition

in a certain period in time. Including future scheduling concerns into manpower planning is an interesting research direction, which should lead to improved long-term schedule qualities.

### 3.2 Horizontal decision levels

Some optimisation problems are solved by people responsible for completely different structural parts of an organisation, without any hierarchical dependency. Nurse rostering, operating theatre scheduling [5] and patient admission scheduling [6,9], for example, are three demanding optimisation problems in hospitals. A good solution for one of the three problems strongly constrains the other two problems. Some interaction among the three planners would definitely help a hospital to increase its service level and personnel satisfaction at the same time.

Besides the previous challenge at the operational level, departmental personnel scheduling cannot always be dealt with in an isolated manner. Departmental planners try hard to improve the quality of their own schedule. They sometimes need to do that by temporarily transferring a member of staff from one department to another, while having only subjective information about workload differences. This interesting negotiation process can be automatised in an objective manner [14], which opens up many perspectives for better decision support.

A different issue arises when personnel scheduling is strongly interwoven with other combinatorial optimisation problems, in such a way that the problems cannot be solved independently at all. Problems like this are called ‘structured problems’. Home care scheduling [1,11] is one example of a problem that requires nurse rostering and vehicle routing to be solved at the same time. A good quality home care schedule should optimise service to the patients while also optimising individual nurses schedules and minimising driving time or distance. One possible way to address the problem could be to develop one single model and a general purpose heuristic to generate a solution. It may be better, however, to take advantage of the existing knowledge on nurse rostering and vehicle routing and include some aspects in a dedicated approach.

Task scheduling is another example of a structured personnel scheduling problem, common in the retail, production and health care sector. Shift scheduling is too coarse grained to produce satisfactory solutions to the task scheduling problem, but the tasks need to be composed such that good quality shift schedules are obtained for the personnel.

The connection with other problems makes it obviously harder to arrive at acceptable solutions. Nevertheless, it is interesting to have sufficient supply of new personnel scheduling problems for the timetabling community to keep improving existing optimisation approaches and developing completely new ones.

## References

1. S. Bertels and T. Fahle. A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. *Computers & Operations Research*, 33(10):2866–2890, 2006.
2. B. Bilgin, P. De Causmaecker, B. Rossie, and G. Vanden Berghe. Local search neighbourhoods to deal with a novel nurse rostering model. *Annals of Operations Research*, 194(1):33–57, 2012.
3. E.K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188:330–341, 2008.
4. E.K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
5. Brecht Cardoen, Erik Demeulemeester, and Jeroen Beliën. Operating room planning and scheduling: A literature review. *European Journal Of Operational Research*, 201(3):921–932, 2010.
6. Sara Ceschia and Andrea Schaerf. Local search and lower bounds for the patient admission scheduling problem. *Computers and Operations Research*, 38(10):1452–1463, 2011.
7. P. De Causmaecker and G. Vanden Berghe. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14:3–16, February 2011.
8. F. Della Croce and F. Salassa. A variable neighborhood search based matheuristic for nurse rostering problems. In B. McCollum, E. K. Burke, and G. White, editors, *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, pages 167–175. Queen’s University Belfast, 2010.
9. Peter Demeester, Wouter Souffriau, Patrick De Causmaecker, and Greet Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine*, 48(1):61–70, 2010.
10. 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.
11. P. Eveborn, P. Flisberg, and M. Ronnqvist. Laps Care—an operational system for staff planning of home care. *European Journal of Operational Research*, 171(3):962–976, 2006.
12. M. Isken. An implicit tour scheduling model with applications in healthcare. *Annals of Operations Research*, 128:91–109, 2004.
13. D. L. Kellogg and S. Walczak. Nurse scheduling: From academia to implementation or not? *Interfaces*, 37(4):355–369, 2007.
14. Ruben Lagatie, Stefaan Haspeslagh, and Patrick De Causmaecker. Negotiation Protocols for Distributed Nurse Rostering. In Toon Calders, Karl Tuyls, and Mykola Pechenizkiy, editors, *Proceedings of the 21st Benelux Conference on Artificial Intelligence*, pages 145–152, 2009.
15. Z. Lü and J.-K. Hao. Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, 218(3):865 – 876, 2012.
16. B. Maenhout and M. Vanhoucke. An electromagnetic meta-heuristic for the nurse scheduling problem. *Journal of Heuristics*, 13:359–385, 2007.
17. E.J. Lodree Jr., C.D. Geiger, and X. Jiang. Taxonomy for integrating scheduling theory and human factors: Review and research opportunities. *International Journal of Industrial Ergonomics*, 39(1):39 – 51, 2009.
18. M. Moz and M.V. Pato. A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research*, 34(3):667–691, 2007.
19. K. Nonobe. An approach using a general constraint optimization solve. Proceedings of PATAT2010, 2010.
20. C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, and E. Housos. A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, 219(2):425–433, 2012.

# Papers

# Repairing High School Timetables with Polymorphic Ejection Chains

Jeffrey H. Kingston

Received: date / Accepted: date

**Abstract** This paper introduces polymorphic ejection chains, and applies them to the problem of repairing time assignments in high school timetables while preserving regularity. An ejection chain is a sequence of repairs, each of which removes a defect introduced by the previous repair. Just as the elements of a polymorphic list may have different types, so in a polymorphic ejection chain the individual repairs may have different types. Methods for the efficient realization of these ideas, implemented in the author's KHE framework, are given, and some initial experiments are presented.

**Keywords** High school timetabling · Ejection chains

## 1 Introduction

Most work in timetabling utilizes two phases: a *construction* phase, in which an initial solution is built, for example by a construction heuristic, and a *repair* phase, in which the solution is improved, for example by local search.

Local search works well initially, but its effectiveness declines, for two reasons. The solution's *defects* (specific points where it is deficient) become few and isolated, but local search continues to change parts of the solution where there are no defects to remove. And a point is reached where the small changes it typically makes (moves and swaps) have all been tried and have little chance of improving the solution.

Some repair methods attempt to avoid these problems. One well-known example is very large-scale neighbourhood search (Ahuja et al. 2002; Meyers and Orlin 2007). It repeatedly deassigns and reassigns many related variables. It can be targeted at specific defects by building neighbourhoods around them (Ryan and Rezanova 2010), and it may make many more changes than one move or swap.

---

Jeffrey H. Kingston  
School of Information Technologies  
The University of Sydney  
E-mail: jeff@it.usyd.edu.au

This paper repairs time assignments in high school timetables using *ejection chains*. An ejection chain is a sequence of *repair operations* (also called *repairs*), which are usually but not necessarily moves and swaps. The first repair removes one defect but introduces another; the next removes that defect but introduces another; and so on. Starting at defects and coordinating repairs like this avoids the problems with local search identified above. A key point is that defects that appear as the chain grows are not known to have resisted attack before. It might be possible to repair one of them without introducing another, bringing the chain to a successful end.

Ejection chains are not new. Augmenting paths, found in matching algorithms, are examples of them, and they occur naturally to anyone who tries to repair a timetable by hand. They were brought into focus and named by Glover (1996), who applied them to the travelling salesman problem. In timetabling, they have been applied to nurse rostering (Dowland 1998), teacher assignment (Kingston 2008), and time repair (Kim and Chung 1997). Kim and Chung (1997) is very cryptic, unfortunately.

This paper tells three interwoven stories. The first story is concerned with repair operations for high school timetables which improve them without disrupting their *regularity*. Informally, a regular timetable is one whose events occur at similar times. For example, the well-known arrangement followed by many North American universities, in which each course occupies one of the sets of times  $\{Mon1, Wed1, Fri1\}$ ,  $\{Mon2, Wed2, Fri2\}$ , and so on, is perfectly regular. Although this paper uses these repairs as steps in ejection chains, they could equally well be used in the conventional way, to define neighbourhoods for local search algorithms.

The second story concerns the design of ejection chain software. This paper seems to be the first to explicitly recognize the polymorphism inherent in ejection chains: each repair in the chain may have a different type. This insight leads to a framework in which any number of different types of defects can be repaired together, bringing the method to a level of generality (in the sense of applicability to many combinatorial optimization problems) approaching that of metaheuristics.

The third story concerns the implementation of these ideas within the author's KHE high school timetabling framework (Kingston 2010a), including data structures for expressing regularity and marshalling defects for repair. Ejection chain algorithms are *white-box algorithms* (Parkes 2010): they need access to more information about the current solution than just its cost, so the implementation effort is quite high.

This paper uses the recently developed XML format for high school timetabling as its specification of the high school timetabling problem (Kingston 2010b; Post et al. 2012, 2011). This specification will not be repeated here, since it is enough to understand that the problem is to assign times and resources (teachers, rooms, and so on) to a collection of events so as to avoid clashes and satisfy a number of other constraints, as far as possible. More detail is given as needed throughout the paper.

## 2 Defects

Local search algorithms need access to the cost of the solution and to the repair operations that may be used to change it (moves, swaps, and so on). Ejection chain algorithms also need access to its *defects*: the specific points where problems lie.

In high school timetabling, as in most real-world combinatorial optimization problems, there are several types of defects (clashes, events scheduled at undesirable times, and so on), and several types of repairs. This inherent polymorphism is partly obscured in most timetabling work: repair types may be recognized, but the defects are lumped together into a single number, the overall cost. Fully polymorphic repair, in which defects are repaired in ways specific to their types, is rare in the timetabling literature. One recent paper used it to improve staff rosters (Ásgeirsson 2010).

Surprisingly, for the time repair problem which is the focus of this paper, there are only three defect types. The XML format has 15 constraint types, and there is one defect type for each constraint type, but only two of them matter here: the *prefer times defect*, which occurs when an event is assigned an undesirable time (for example, an afternoon time when the event is supposed to occur during the morning), and the *spread events defect*, which occurs when the events for one subject are supposed to be spread evenly through the *cycle* (the chronologically ordered sequence of all times when events may occur), but instead some of them occur close together in time.

The third defect type is not derived from any constraint in the XML format, but it is nevertheless the most important type of all. Suppose a time assignment assigns six Science events, each requiring one Science laboratory, to the third time on Tuesdays, and suppose the school has only five Science laboratories. Then when rooms are assigned later on, one of these six events must miss out.

As is well known, such problems can be detected by building a bipartite matching graph for each time of the cycle. Each graph contains one *supply node* for each resource in the instance, and one *demand node* for each demand for a resource made by the events running at the graph's time. Each demand node is connected to the supply nodes that represent resources capable of satisfying the demand. To decide whether the demands of the events running at one time can all be satisfied, find a maximum matching in this graph. If it fails to touch every demand node, there is a problem.

For example, the six Science events produce six demand nodes connected to the five supply nodes representing the five Science laboratories (plus other demands for student groups and teachers). One of the six demand nodes will fail to match.

KHE has matching graphs, and considers each unmatched demand node to be a defect called a *demand defect*. Demand defects include clashes and use of resources at times when they are unavailable as special cases. For example, a clash turns up as two demand nodes, both linked only to a single supply node representing the resource in contention. So two of the constraints of the XML format that apply to resources, the *avoid clashes* and *avoid unavailable times* constraints, are taken account of in this way. Some other constraints on the timetables of resources are relevant to time assignment when the resources involved are preassigned. They are not treated here, but the polymorphic approach makes it straightforward to do so.

### 3 Time assignment in KHE

This section explains how time assignment is modelled by the author's KHE platform. There is a lot of detail, and it will be necessary to introduce some of KHE's jargon.

In KHE, an *instance* of the high school timetabling problem (a case of the problem for a given school in a given year or semester) is a structured object which remains immutable after creation. A *solution* of an instance is a mutable structured object containing the time and resource assignments which define the solution. Keeping solutions separate from instances has several advantages. For example, it helps when constructing multiple solutions in parallel.

The lessons for one subject usually need to be spread evenly through the cycle. For example, a class might attend Mathematics for a total of six times, which are to be spread through the five days of the week. On the day when the class meets for two times, it is usually best for those times to be adjacent, forming one lesson of twice the usual duration. It is common for the total number of times devoted to one subject to be rigidly prescribed, but for the way in which that total is split into individual lessons of varying duration to be more flexible. This is handled as follows.

An instance contains *events* of fixed duration, each representing a single subject, plus constraints saying how they may be split into lessons. For the Mathematics example just given, there might be one Mathematics event of duration 6, and constraints saying that 5 or 6 lessons are required, whose durations may be 1 or 2.

In KHE, the solution contains the individual lessons, which are called *meets*. Each meet has a fixed integer *duration*, meaning that it runs for that many consecutive times; a *starting time*, which is a variable requiring assignment; and a set of *tasks*, each of which is a variable requiring the assignment of one resource, such as a student group, a teacher, or a room. Tasks contain demand nodes and are the source of demand defects. Meets and tasks may be preassigned.

Although the aim is to assign a starting time to each meet, KHE does this in an indirect way, by assigning one meet to another. The meaning is that the two meets must have the same starting time, but that time is yet to be determined. Assignment is directed (it assigns one meet to another, not two meets to each other) and includes an offset. For example, suppose meet  $m_1$  has duration 1 and meet  $m_2$  has duration 2. Then  $m_1$  may be assigned to  $m_2$  with offset 0, meaning that  $m_1$  starts at the same time as  $m_2$ , or with offset 1, meaning that it starts at the second time of  $m_2$ . The assigned meet may not run outside the interval of time that the meet it is assigned to is running. For example,  $m_2$  may not be assigned to  $m_1$  at any offset.

Assigning one meet to another supports *hierarchical timetabling*, in which a few meets are timetabled together, then the whole assembly is timetabled into a larger context, and so on until the complete timetable is built. When this is done, a sequence of assignments builds up, from one meet to another, from that meet to a third, and so on. Special meets called *cycle meets* are available such that assignment to a cycle meet effectively assigns a time. When every assignment sequence ends at a cycle meet, every meet has a starting time. There is usually one cycle meet for each sequence of adjacent times in the cycle not spanning a meal break or the end of a day.

One use for hierarchical timetabling is to link meets whose events are required to run simultaneously. For example, suppose there are five Mathematics events of equal duration, one for each of five Year 8 student groups. These events are required to run simultaneously so that the Year 8 students can be regrouped by ability at Mathematics. To handle this, break the events into meets of the same durations, choose one student group to be the leader, and assign the meets of the non-leader student



groups to corresponding meets of the leader student group, with offset 0. This forces the events to be simultaneous. The common starting times of the meets will be determined later, when assignments are made to the leader student group's meets.

There are good reasons to remember that certain meets are derived from one event. For example, they usually need to be spread evenly through the cycle, which might involve finding assignments for them all at the same point during solving. In KHE, meets may be grouped together into sets called *nodes*. Usually, one node holds the meets derived from one event, but there are exceptions, such as *runaround nodes*, which hold small timetables in which several student groups attend several subjects. There is also a *cycle node* holding the cycle meets.

By convention, a non-cycle meet lies in a node if and only if its assignment may be changed. The meets of the leader student group in the example above would lie in a node, but the other meets, which already have final assignments, would not.

In addition to holding meets, each node except the cycle node usually has a parent node, forming the nodes into a tree rooted at the cycle node that the author has called the *layer tree* (Kingston 2006). When a meet lies in a node, it may only be assigned to meets in the parent of that node, forcing all sequences of assignments to eventually end in cycle meets as desired. Although it is not forbidden, there is an assumption that meets which share a node will not be assigned so as to overlap in time. For example, it is not desirable for two meets derived from the same event to overlap in time.

#### 4 Regularity

Regularity has two forms. *Meet regularity* occurs when the sets of times at which two meets are running are either disjoint, or one is a subset of the other. For example, two meets of duration 2, one starting at the first time on Wednesdays, the other at the second time, are not regular. When all meets have duration 1, meet regularity is automatic. In practice, most meets have duration 1 or 2, and it only takes a little care to achieve very good meet regularity.

*Node regularity* occurs when the sets of times at which the meets of two nodes are running are either disjoint, or one is a subset of the other. Since most nodes contain several meets, node regularity is harder to achieve than meet regularity.

Node regularity is important. In the author's experience, based on Australian high schools, the main type of defect left over at the end of solving is the *split assignment*, in which one teacher attends some of the meets of an event, and another teacher attends the others. Split assignments are permitted, but they are undesirable. Some of their causes are inherent in instances: part-time teachers who are difficult to utilize effectively, and tight teacher workload limits that force every teacher to be used to capacity. But node irregularity also causes split assignments, and it is the one cause that solvers can do something about.

Because node regularity depends on coordinating the assignments of many meets (perhaps five derived from one event and five from another), it is not likely to arise by chance in the course of repairing the assignments made to individual meets, not even if the solver explicitly measures irregularity and favours assignments that reduce it. (KHE does not do this at present, and the argument just given explains why it is not

a priority.) On the other hand, when constructing a time assignment node by node to begin with, it is easy to find previously assigned nodes with compatible sets of meets whose assignments can be re-used.

The author's current strategy for achieving node regularity, then, is as follows. When constructing the initial time assignment, give priority to node regularity, even ahead of minimizing demand defects. KHE offers a function that does this. Its algorithm is a descendant of the tiling algorithm published some years ago by this author (Kingston 2005), so it will not be described in detail here (the KHE User's Guide has a full description). Then repair the initial assignment, but restrict the repairs to operations which do not disrupt such node regularity as is already present.

The algorithm which constructs the initial time assignment begins by assigning the nodes of whatever student form seems likely to serve best as a template for assigning the others (usually the most senior form). It tries hard to find a very good assignment for this form; since no other forms have been timetabled yet, it should be possible to assign it with no defects at all. For each node of this first form, the set of times that its meets are running is called a *zone*. The algorithm stores these zones permanently in the cycle node (the common parent of the forms' nodes). When assigning subsequent forms, it tries to ensure that each node's meets are assigned entirely within one zone, as far as possible.

This approach is only effective if the chosen form attends classes at every time of the cycle, since if not, some times will receive no zone, or at any rate no guidance on which zone they should lie in. A more general approach, not implemented yet, would be to look through the instance and decide on a set of zones in advance. This is effectively what North American universities do when they define zones  $\{Mon1, Wed1, Fri1\}$ ,  $\{Mon2, Wed2, Fri2\}$ , and so on.

## 5 Repair operations that preserve regularity

The picture of the data that a time repair algorithm has to work with is now complete: meets grouped into nodes and assigned to meets in parent nodes with zones. The assignment of a node is considered regular if it places all its meets into one zone. Deeper in the tree there will be nodes with no zones; they may be considered to have a single zone holding all their meets.

Several repair operations preserve existing node regularity within this structure.

Given two nodes which are children of the same parent node and which have meets with the same durations, the assignments of corresponding meets may be swapped. Each node will be as regular after the swap as the other was before it. This *node swap* repair operation will usually be neutral with respect to prefer times and spread defects, and it could well reduce demand defects. For example, if the Year 12 students attend both Mathematics and English for 6 times per week, in lessons of equal durations, then the times they attend Mathematics can be swapped with the times they attend English without reducing regularity.

The assignments of two meets of equal duration may be swapped when they are assigned to the same zone, whether or not they lie in the same node. For example, suppose some zone includes a double time containing the first two times on Wednesday,

and that the Year 10 students attend History at the first of these times and Science at the second. Then these two events may be swapped without reducing regularity. This *meet swap* repair operation is also available, in a slightly different form, when the two meets' durations differ, provided their assignments are adjacent in time. When the meets are derived from the same event, a meet swap accomplishes nothing and would not be tried. Two meets may also be swapped if they are the only meets in their nodes, but in that case the operation is better classified as a node swap.

It is possible to go further when irregularity is already present. For example, if a meet lies in a different zone from all the others in its node, it can be moved to any zone without increasing irregularity. The algorithm presented in this paper does not yet check for such cases.

Node and meet swaps can be applied at any level of the layer tree. Here is one surprisingly high-level application. Make a new node with the same number of meets as the cycle node, and the same durations. Make the new node a child of the cycle node and assign its meets to the corresponding meets of the cycle node. Let the new node have the same zones as the cycle node, and delete the zones of the cycle node. Then move all the other child nodes of the cycle node so that they become children of the new node, and assign all their meets to the meets of the new node corresponding to their previous assignments. This reorganization does not change the timetable, nor its node regularity as measured by zones; it merely interposes a redundant node between the cycle node and its child nodes.

Apply meet swaps to the meets of the new node. The cycle node now has no zones, so these meets are free to swap with each other whenever they have the same durations or are adjacent in time. Since the cycle node has one meet for each set of consecutive times not spanning a break, one swap might swap the entire Wednesday morning timetable with the entire Thursday morning timetable, for example. Depending on how meet-regular the timetable is, it may also be possible to break these meets into smaller ones, and swap half-mornings and so on.

In the Australian instances which are this author's main focus, virtually all student group resources are preassigned to events whose total duration equals the number of times in the cycle. Under those circumstances, the only practical repair is the meet swap (or several meet swaps, as in the node swap) between meets containing the same preassigned student group resources. However, there are European instances in which students attend for fewer times, and even in Australian instances there are staff meetings which just need to occur whenever the teachers involved are most available. Thus, there is a need for a repair operation which moves one meet to a new time. This time should be one when the meet's preassigned resources are not busy—just the opposite of what is required when swapping.

Meet moves and swaps can be unified into a single well-known repair operation, here called the *Kempe meet move*. It starts by moving one meet, say from time  $t_1$  to time  $t_2$ . If that causes clashes with other meets, those other meets are moved from  $t_2$  to  $t_1$ . If that in turn causes clashes with other meets, the other meets are moved from  $t_1$  to  $t_2$ , and so on for as long as new clashes appear. A Kempe meet move could fail for several reasons, but when it succeeds, it has moved the meet without increasing the overall number of clashes or the number of cases where a preassigned resource attends a meet at a time when it is unavailable.

All the meets moved are required to have the same duration, except in the special case where the second meet moved is adjacent to the first in time. In that case, all the meets moved on odd-numbered steps are required to have the same duration, all the meets moved on even-numbered steps are required to have the same duration, and each meet moves to the other end of the block of adjacent times during which the first two meets to be moved were originally running.

The ejection chain algorithm of this paper uses two repair operations: node swaps and Kempe meet moves. Kempe meet moves conveniently avoid the clumsiness of having one repair operation which swaps a meet to some times and another which moves it to the rest. Node regularity is preserved by allowing only repairs that do not increase the number of zones to which the meets of any affected node are assigned.

## 6 From defects to repairs

The repairs just defined may be applied in the traditional way to build neighbourhoods for local search which preserve node regularity. However, if they are to be used to repair defects, a path must be defined from each defect to a set of alternative repairs, each of which removes that defect.

Given a defect, the first step is to find the *contributing meets*: those meets whose assignments contribute to the defect and may be changed. For a prefer times defect, look through the meets derived from its event to find those assigned undesirable times. For a spread events defect, look through the monitored meets to find those for which a move to some other day would reduce the spread cost. For a demand defect, query the matching graph to find the demand nodes that are competing for the insufficient supply. (KHE offers an operation for this. For the Science laboratories example given some time ago, it would return all six Science laboratory demand nodes.) From each demand node, proceed to its task and from there to the task's meet.

For each meet identified by these steps, ascend its sequence of assignments to other meets. Any non-cycle meet on this sequence that lies in a node is a contributing meet: changing its assignment is permitted and might fix the defect.

For each contributing meet  $m$ , a Kempe meet move is possible to each legal offset in each meet of the parent node of  $m$ 's node (except  $m$ 's current meet and offset), provided the move does not change  $m$ 's zone. Ejection chains work best when repairs do actually remove the defects that provoked them, so each of these moves is only tried if it will do this: when repairing prefer times defects, only moves that will give the meet a preferred time are tried; when repairing spread events defects, only moves which will reduce the spread cost are tried; and when repairing demand defects, all moves are tried, since they all move  $m$  away from the insufficient supply.

KHE offers a *layer* data structure which groups together nodes with the same parent node and the same preassigned resources. After all Kempe meet moves of  $m$  have been tried without success, node swaps are tried between  $m$ 's node and the other nodes of its layer which have meets of the same durations as  $m$ 's node. Node swaps are not likely to be very effective at removing prefer times defects and spread events defects, since they merely shift these defects from one event to another, but they are potentially very valuable for removing demand defects.

## 7 Polymorphic ejection chains

The previous section showed how to identify the meets whose assignments contribute to a defect, and a set of alternative repairs that remove that defect. But it is very likely that removing one defect will create another. This is where ejection chains enter the picture: they chain repairs together, with each repair removing a defect created by the previous repair, until, with luck, a repair occurs which creates no new defects. This section introduces ejection chains and shows how to implement them polymorphically, that is, so that any number of types of defects, and any number of types of repairs, can be incorporated in a uniform way.

The heart of the ejection chain algorithm is a function that will be called `Augment`, since it is based on the well-known function for finding an augmenting path in bipartite matching. `Augment` targets one defect and tries a set of alternative repairs on it. Each repair removes the defect, but may create new defects. If no new defects of significant cost appear, `Augment` terminates successfully. If one significant new defect appears, `Augment` calls itself recursively in an attempt to remove that defect; in this way a chain of coordinated repairs is built up. If two or more significant new defects appear, `Augment` undoes the repair and continues with alternative repairs. It could try to remove all the new defects, but that would rarely succeed in practice.

The author's formulation of `Augment` has the following interface:

```
bool Augment(Defect d, Solution s, Cost c);
```

It has precondition

```
cost(s) >= c && cost(s) - cost(d) < c
```

where `cost(s)` is the current overall cost of solution `s`, and `cost(d)` is the contribution to this cost made by `d`, which is one of `s`'s defects.

If `Augment` can change `s` so as to reduce `cost(s)` to less than `c`, it does so and returns `true`; otherwise it leaves `s` unchanged and returns `false`. The second part of the precondition implies that removing `d` without adding any new defects, if that can be done, would achieve success. Here is an abstract implementation:

```
bool Augment(Defect d, Solution s, Cost c)
{
    repair_set = RepairsOf(d);
    for( each repair r in repair_set )
    {
        new_defect_set = Apply(s, r);
        if( cost(s) < c )
            return true;
        for( each e in new_defect_set )
            if( cost(s) - cost(e) < c && Augment(e, s, c) )
                return true;
        UnApply(s, r);
    }
    return false;
}
```

It begins by finding a set of repairs for  $d$ . For each of those, it applies the repair and receives the set of new defects introduced by that repair, checks for success, then if success has not been achieved it unapplies the repair and continues with the next repair, returning `false` when all repairs have been tried without success.

Success could come in two ways. Either one of the repairs reduces  $\text{cost}(s)$  to below  $c$ , or some new defect  $e$  has cost large enough to ensure that removing it alone would constitute success, and a recursive call targeted at  $e$  succeeds. Notice that  $\text{cost}(s)$  may grow without limit as the chain deepens, provided that there is a single defect  $e$  whose removal would reduce the cost of the solution to less than  $c$ .

When there are several defect types, several `Augment` algorithms are needed, one for each defect type, dynamically dispatched on the type. Ejection chains are naturally polymorphic: repairing a demand defect could create a spread defect, repairing that defect could create a prefer times defect, and so on. Repairs can usually be generated and applied directly, rather than being represented as a set of objects as above.

The tree searched by `Augment` as presented may easily grow to exponential size, which is not the intention. The author has tried two methods of limiting its size, both of which seem to be useful. They may be used separately or together.

The first method is to limit the depth of recursion to a fixed constant, perhaps 3 or 4. The maximum depth is passed as an extra parameter to `Augment`, and reduced by one on each recursive call, with value 0 preventing further recursion. Not only is this method attractive in itself, it also supports *iterative deepening*, in which `Augment` is called several times on the same defect, with the depth parameter increased each time. Another idea is to use a small depth limit on the first iteration of the main loop (see below), and increase it on later iterations.

The second method is the one used by the augmenting path method from bipartite matching. Just before each call on `Augment` from the main loop, the entire solution is marked unvisited (by incrementing a single global visit number, not by traversing the entire solution). When a repair changes some part of the solution, that part is marked visited. Repairs that change parts of the solution that are already marked visited are tabu. In this way, the size of the tree is limited to at most the size of the solution.

Given a solution and the set of all its defects, or a subset of its defects that it is expedient to target, the main loop cycles through the set repeatedly, calling `Augment` on each defect in turn, with  $c$  set to  $\text{cost}(s)$ . The set of defects and  $\text{cost}(s)$  change with each successful `Augment`. The main loop exits when `Augment` has been tried on every defect since the last successful call to `Augment`. At that point, no further successful augments are possible, assuming that `Augment` contains no randomness. This is a very clear-cut stopping criterion compared with, say, the stopping criteria used by metaheuristics. Under reasonable assumptions, it ensures that the whole algorithm runs in polynomial time, for the same reason that hill-climbing does.

## 8 Realizing ejection chains in KHE

This section sketches how the author's KHE platform supports ejection chains. Full details are available in KHE's documentation.

KHE has *monitor* objects, each of which monitors one point of application of one constraint, or one demand node of one matching graph. Each monitor contains a cost, which KHE keeps up to date as the solution changes. For example, for each set of meets which are required to spread evenly through the cycle there is a monitor. This monitor is notified whenever the starting time of any of its meets changes, triggering it to update its cost and notify any change.

*Group monitors* may be created to monitor other monitors. These other monitors notify their group monitor of any change in cost, rather than notifying the solution directly. The cost of a group monitor is the total cost of the monitors it monitors, so when any of its monitors' costs change, the group monitor's cost does too, and it must notify its own group monitor, and so on.

Group monitors are useful when several nominally distinct monitors monitor the same thing in reality. Take the example of several events required by a link events constraint to run simultaneously. These are usually handled by a preprocessing step which links their meets together so that only simultaneous assignment is possible thereafter. Each event may have its own spread events monitor, but these all monitor the same thing in reality and are best treated as a single monitor, which is done by grouping them. It is the group monitor that appears on lists of monitors, with the monitors it groups hidden from view below it.

The object representing the solution as a whole is (among other things) a group monitor. Provided each monitor reports its cost to this special group monitor, either directly or via intermediate group monitors, this special group monitor will hold the current total cost of all non-group monitors, which is the cost of the solution.

A defect (of type `Defect` in the algorithm above) is realized in KHE as a monitor, often a group monitor, of non-zero cost. For each type of defect (or group of related defects), the user has to write an implementation of `Augment` specialized to that type of defect, and register it with an *ejector* object defined by KHE, which also holds other useful information, such as the desired method of limiting depth. Each of these functions iterates over a set of repairs of the user's choice, applying and unapplying each repair in turn, and calling a function supplied by KHE which handles the testing for success and the recursive call, dynamically dispatching it to one of the functions registered with the ejector. KHE also supplies an implementation of the main loop. So the user only has to implement the code that converts a defect into a set of repairs and applies and unapplies each repair in turn; the rest comes for free.

Each group monitor makes available the set of its child monitors whose cost is non-zero. Keeping this set up to date requires a small constant amount of work each time a monitor reports a change in its cost from zero to non-zero or vice versa. The set of defects iterated over by the main loop of the ejection chain algorithm is just this set of monitors, for some particular group monitor given to the ejector object by the user. This group monitor could be the special solution group monitor, in which case every defect in the solution is open to repair, or it could be a group monitor whose children are just some of the monitors, in which case only defects among those children are open to repair. For example, the time repair ejector object is given a group monitor whose children are those monitors concerned with time assignment.

While a repair is being applied, KHE's *tracing* feature records which children of the ejector's group monitor changed in cost during the repair, and by how much. The

subset of these monitors whose cost increased is the new defect set used by Augment. KHE also offers a *transactions* feature which allows the operations carried out since some starting point to be recorded, and subsequently undone and redone. This makes it easy to undo a complex repair, such as a Kempe meet move, as required by the UnApply step of Augment. Transactions are also used by a variant of the algorithm which tries all ejection chains, remembers the best, and redoes it at the end.

## 9 Experiments

This paper has been concerned with explaining how ejection chains can be applied to real-world timetabling problems, in particular to repairing time assignments while preserving regularity. It is not empirical in orientation. Accordingly, the experiments of this section have very modest aims: to show that the algorithm for repairing time assignments produces a reasonable result in a reasonable time, and to shed light on some design choices, without claiming to be definitive.

The algorithm tested here is KHE's standard ejection chain solver, configured to use a simple form of iterative deepening (Sect. 7), and loaded with augment functions that make Kempe meet moves and node swaps after carrying out the mapping from defects to repairs described in Sect. 6.

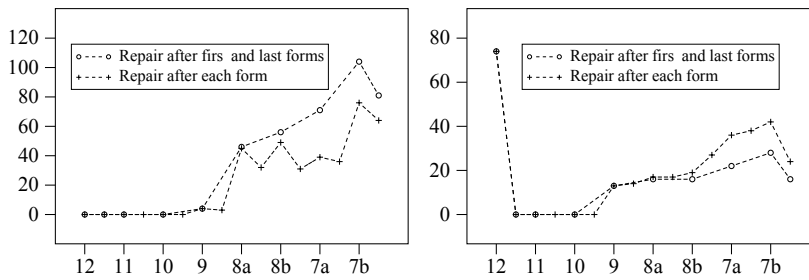
In the XML format, each constraint has an integer weight which is multiplied by the number of defects to produce a cost. Each constraint also has a Boolean *required* attribute. If this attribute is true, the constraint is 'hard' and its cost is added to a total called the *infeasibility value*  $i(s)$  of the solution  $s$ . If the attribute is false, the constraint is 'soft' and its cost is added to a different total called the *objective value*  $o(s)$  of the solution. A solver aims to minimize the ordered pair  $(i(s), o(s))$ . The experiments reported here are confined to one difficult real-world instance (BGHS98 from the XHST2011 archive (Post 2011)). It has no prefer times constraints, and its spread constraints are soft with weight 1. Demand defects are treated as hard with weight 1. So, in these experiments, the hard cost is the number of demand defects, and the soft cost is the number of spread defects.

A *student form*, or just *form*, is a set of resources, each representing one group of students from the same age cohort. The construction algorithm assigns times to the meets of one form at a time. The first question, then, is whether repair is needed at all, and if so, whether it is needed after assigning each form. Repair is certainly needed after assigning the first form: it is important to assign that form as well as possible, because its assignments guide the assignments of the other forms. Accordingly, two runs were performed, the first repairing after the first and last forms only, taking 10.0 seconds (all times include both construction and repair), and the second after each form, taking 16.1 seconds.

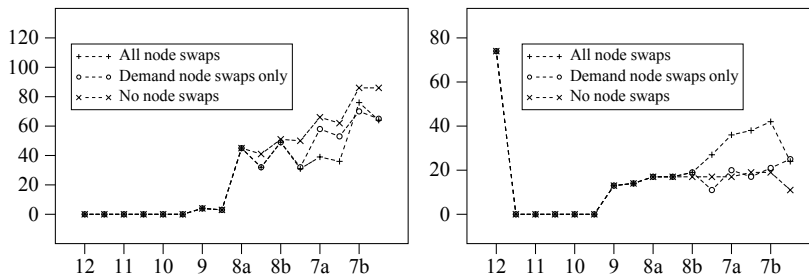
The results appear in Fig. 1. Repair after each form is superior, producing 17 fewer demand defects in the end, and more at intermediate points. This is not surprising: more repairs succeed when the timetable is only partly assigned and resource demands are not pressing.

The difference between the final number of demand defects when only the first form is repaired (104) and when every form is repaired (64) is important, because

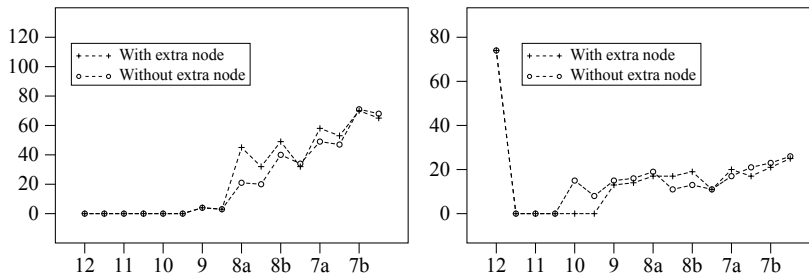




**Fig. 1** Effectiveness of repair. The vertical axis represents hard cost in the first graph, soft cost in the second. The horizontal axes have one label for each student form, in order of assignment. For each form there is one data point representing the cost after constructing the time assignment of that form, possibly followed by a second point representing the cost after repair. Some repairs increase soft cost, because decreasing hard cost takes priority. The first four forms (12, 11, 10, and 9) are complete forms, the next four (8a, 8b, 7a, and 7b) are half-forms. Other ‘forms’ containing staff meetings follow these forms, but they add virtually no cost so have been omitted.



**Fig. 2** Effectiveness of node swaps. Details as in Fig. 1.



**Fig. 3** Effectiveness of an extra node under the cycle node. Details as in Fig. 1.

repairs that sacrifice regularity will need to be applied to these remaining defects. After the runs reported here, the author’s solver removes all zones and interior nodes, thereby removing all requirements for regularity, and runs the repair algorithm again. Demand defects then drop dramatically, typically to just 5 or 6, at the cost of some loss of regularity that shows up in split resource assignments later.

The author’s long-term goal is to solve high school instances reliably in about 10 seconds, including resource assignment, which follows after time assignment and takes several seconds. This raises the question (also interesting for its own sake) of

whether there are unproductive parts of the algorithm that could be removed, saving time without degrading performance.

Node swaps are one possibility. As explained previously, they seem likely to be effective at removing demand defects, but unlikely to be effective at removing spread defects. This is investigated by continuing with repair after assigning each form, but now trying it with all node swaps (taking 16.1 seconds as before), node swaps when repairing demand defects only (12.9 seconds), and no node swaps (23.3 seconds). The apparent contradiction in the last run time, of doing less work but taking longer, is quite common with ejection chains. It is a sign that node swaps are removing defects effectively, so that the algorithm struggles without them.

The results appear in Fig. 2. Omitting all node swaps is inferior, producing significantly more demand defects from form 8a onwards. There is some support for using node swaps on demand defects only (the end results, at least, are indistinguishable), and it does save time (3.2 seconds).

Another possibility for reducing run time is to omit the extra node directly under the cycle node. Its presence more than doubles the number of Kempe meet moves available to most repairs. This is investigated by repairing after each form, with node swaps for demand defects only, but now trying with this extra node (taking 12.9 seconds as before) and without it (8.0 seconds).

The results appear in Fig. 3. The extra node leads to 3 fewer demand defects in the end, but may not be worth its cost in run time (4.9 seconds). More data are needed.

## 10 Conclusion

This paper introduces polymorphic ejection chains and shows that they are effective for repairing time assignments in high school timetables while preserving regularity. Unlike local search, polymorphic ejection chains target specific defects and build sets of coordinated repairs. Based on the results in this paper and other papers cited earlier, it seems likely that they would be effective in improving solutions to many real-world combinatorial optimization problems.

There is little more to do in the area of time repairs that preserve regularity, but other applications beckon. Within high school timetabling, removing resource defects such as split assignments and unwanted gaps in teachers' timetables requires complex repairs that reassign times and resources together. That promises to be an even richer area of application for polymorphic ejection chains than the one explored here.

## References

- R. Ahuja, Ö. Ergun, J. Orlin, and A. Punnen, A survey of very large-scale neighbourhood search techniques, *Discrete Applied Mathematics*, 123, 75–102 (2002)
- Eyjólfur Ingi Ásgeirsson, Bridging the gap between self schedules and feasible schedules in staff scheduling, PATAT10 (Eighth international conference on the Practice and Theory of Automated Timetabling, Belfast, August 2010)
- Kathryn A. Dowsland, Nurse scheduling with tabu search and strategic oscillation, *European Journal of Operational Research*, 106, 393–407 (1998)

- Fred Glover, Ejection chains, reference structures and alternating path methods for traveling salesman problems, *Discrete Applied Mathematics*, 65, 223–253 (1996)
- Myoung-Jae Kim and Tae-Choong Chung, Development of automatic course timetabler for university, *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling*, 182–186 (1997)
- Jeffrey H. Kingston, A tiling algorithm for high school timetabling, *Practice and Theory of Automated Timetabling V*, Springer Lecture Notes in Computer Science 3616, 208–225 (2005)
- Jeffrey H. Kingston, Hierarchical timetable construction, *Practice and Theory of Automated Timetabling VI*, Springer Lecture Notes in Computer Science 3867, 294–307 (2007)
- Jeffrey H. Kingston Resource assignment in high school timetabling, PATAT08 (Seventh international conference on the Practice and Theory of Automated Timetabling, Montreal, August 2008)
- Jeffrey H. Kingston, The KHE High School Timetabling Engine, <http://www.it.usyd.edu.au/~jeff/khe> (2010)
- Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, <http://www.it.usyd.edu.au/~jeff/hseval.cgi> (2010)
- Carol Meyers and James B. Orlin, Very large-scale neighbourhood search techniques in timetabling problems, *Practice and Theory of Automated Timetabling VI*, Springer Lecture Notes in Computer Science 3867, 24–39 (2007)
- Andrew J. Parkes, Combined Blackbox and AlgebRaic Architecture (CBRA), PATAT2010 (Eighth international conference on the Practice and Theory of Automated Timetabling, Belfast, August 2010)
- Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, and David Ranson, An XML format for benchmarks in high school timetabling, *Annals of Operations Research* 194, 385–397, 2012
- Gerhard Post, Jeffrey H. Kingston, Samad Ahmadi, Sophia Daskalaki, Christos Gogos, Jari Kyngäs, Cimmo Nurmi, Haroldo Santos, Ben Rorije and Andrea Schaerf, An XML format for benchmarks in high school timetabling II, *Annals of Operations Research* (online), <http://10.1007/s10479-011-1012-2>, 2011
- Gerhard Post, High school timetabling web site, <http://wwwhome.math.utwente.nl/~postgf> (2011)
- David M. Ryan and Natalia J. Rezanova, The train driver recovery problem – solution method and decision support system framework, PATAT2010 (Eighth international conference on the Practice and Theory of Automated Timetabling, Belfast, August 2010)

---

## Application of a parallel computational approach in the design methodology for the Course timetabling problem

Soria-Alcaraz Jorge A. · Carpio Martín ·  
Puga Héctor · Sotelo-Figueroa Marco A.

the date of receipt and acceptance should be inserted later

**Abstract** The process of gathering enough experimental statistical data over a set of instances of the Course timetabling problem (CTTP) could take a lot of time to an interested researcher. There exist several parallel computing models capable to accelerating the execution process of metaheuristic algorithms. This paper explores the idea to use a parallel model in a metaheuristic algorithm over the Course Timetabling Problem in order to reduce the time that a investigator needs to collect enough data to make a proper conclusion. our parallel approach uses the Methodology of design model for CTTP. The methodology of design is a strategy applied before the execution of an algorithm for timetabling problem. This strategy has recently emerged and aims to generalize and provide a context-independent layer to different versions of the Course timetabling problem. Finally a well-know set of instances was tested with a parallel GA and a sequential GA in order to determine the advantages of the proposed approach for CTTP.

**Keywords** Methodology of design · Parallel computing · Genetic Algorithm · Cellular Genetic Algorithm

### 1 Introduction

The timetabling problem is one of the most difficult, common and diverse problems inside an university. This problem tries to assign several activities into *timeslots* to make a *timetabling*. The main objective of this problem is to obtain a timetabling with the minimum conflicts between assigned activities. [1]

There exist several university timetabling problems as described by Adriaen et. al [2] for example the *Faculty timetabling* tries to assign teachers to

subjects *Class-teacher timetabling* assigns subjects to a fixed group of students. *Classroom assignment* ensures that every pair teacher-subject have a classroom. *Examination timetabling* assigns events like final exams to a set of individual student and *Course timetabling* assigns subjects to individual students minimizing the conflicts (usually time-conflicts) between the assigned events. This paper is focused into the last timetabling problem type.

Like most timetabling problems, the *Course timetabling* is NP-Complete [3] [4]. The reason for this is attributed by the literature to a combinatorial explosion of possible events assigned into time slots, as well as the constraints that each university uses in its own course timetabling creation.

The course timetabling problem (CTTP) is also considered by Rodriguez y Quezada[5], like a "offline" problem meaning that the resolution of the course timetabling is not needed in real time i.e the final user of a solver for the CTTP can wait a reasonable time (several days or even a week) in order to get a "good" solution. However in the academic or experimental field the researcher needs to perform a high number of test with his/her proposed algorithm in order to accumulate enough statistical data to prove the quality of the proposed approach.

That is why the researcher of CTTP have an added challenge: in addition to create an algorithm that is at least comparable with the current state of the art, such algorithm needs to be reasonably fast in order to perform enough experiments and gather statistical evidence to finally make a conclusion. This paper explores the possibility by means of parallel metaheuristics and parallel computing to create fast algorithms for the CTTP with good performance over a well-know instances of the CTTP.

It is an undisputed fact that the CTTP problem usually differs greatly from one university to another. So the researcher has the risk that once he tuned his algorithm to a specific college and moving to testing in another university his approach could do not replicate desired characteristics or ,in the worst of cases, cannot be possible to obtain a solution applicable to reality. In this sense a new approach has emerged ,The Methodology of Design, this approach give us a generic methodology to resolve a widely set of CTTP problems by means of the application of a context-independent layer. [1] In this paper we use this Methodology of Design in order to build a parallel algorithm capable to: A) being applied to the ITC2002, ITC2007 instances and B) have a higher speed than its sequential counterpart making faster the process of experimentation and gathering data.

The paper is organized as follows. Section 2 presents a brief explication of the Methodology of Design, the parallel build-up for the course timetabling problem, the solution approach and its justification. Section 3 contains the experimental set-up, results, their analysis and discussion. Finally Section 4 include some conclusions and future work.

	ACM0403	SCM0414	SCB0421	SCE0418	ACH0408	ACM0401	ACH0404
ACM0403	4	1	1				
SCM0414	1	10	3	3			6
SCB0421	1	3	3	2			2
SCE0418		3	2	3			3
ACH0408							
ACM0401						4	1

Fig. 1 MMA Matrix

## 2 Solution Approach

### 2.1 Problem Definition

A clear and concise definition of the CTTTP is given by Conant-Pablos [6]: A set of events (courses or subjects)  $E = e_1, e_2, \dots, e_n$  is the basic element of a CTTTP. Also there are a set of periods of time  $T = t_1, t_2, \dots, t_s$ , a set of places (classrooms)  $P = p_1, p_2, \dots, p_m$ , and a set of agents (students registered in the courses)  $A = a_1, a_2, \dots, a_o$ . Each member  $e \in E$  is a unique event that requires the assignment of a period of time  $t \in T$ , a place  $p \in P$  and a set of students  $S \subseteq A$ , so that an assignment is a quadruple  $(e, t, p, S)$ . A timetabling solution is a complete set of  $n$  assignment, one for each event, which satisfies the set of hard constraints defined usually by each university of college. This problem is documented to be at least as a NP-complete problem [3] [4].

### 2.2 Methodology of Design for the Course Timetabling Problem

In the literature it can be seen that there is a problem with the diversity of course timetabling instances due different university policies. This situation directly impacts in the reproducibility and comparison of course timetabling algorithms[7]. The state of art indicates some strategies to avoid this problem. For example, a more formal problem formulation [7] as well as the construction of benchmark instances [8]. These schemes are useful for a deeper understanding of the university timetabling complexity, but the portability and the reproducibility of a timetabling solver in another educational institution is still in discussion[1]. In this sense, we use a context-independent layer for the course timetabling resolution process. This new layer integrates timetabling constraints into three basic structures *MMA matrix*, *LPH list* and *LPA list*.

**MMA matrix:** This matrix contains the number of students in conflict between subjects i.e. the number of conflicts if two subjects are assigned in the same timeslots. An example of this matrix can be seen in the Figure 1.

**LPH list :** This structure have in its rows the subjects offered. In its columns have the offered timeslots, So this list give us information about the allowed timeslots per subject. one example of this list can be seen on 1.

**Table 1** LPH list

	Day 1	Day 2
$e_1$	$\langle t_3 \rangle$	$\langle t_2 \rangle$
$e_2$	$\langle t_2 \rangle$	$\langle t_2 \text{ or } t_1 \rangle$

**LPA list** :This list shows in its rows each event and the classrooms available to be assigned to each event without conflict.

**Table 2** LPA list

<i>event</i>	<i>Classrooms</i>
$e_1$	$\langle p_4, p_{l1}, p_{c2} \rangle$
$e_2$	$\langle p_{lab}, p_{c2} \rangle$
$e_3$	$\langle p_6, p_{b2}, p_{b3}, p_{b4} \rangle$
$e_4$	$\langle p_{lab}, p_{l2} \rangle$
$\vdots$	$\vdots$
$e_{530}$	$\langle p_{d7} \rangle$

Once we obtain these structures by means of the natural/original inputs of our CTTTP problem, we ensures *by design* the non-existence of violations by the selection of any values shown in LPH and LPA. Our problem now is to deal with students conflicts only. We work with these conflicts by means of the next minimization function:

$$\min(FA) = \sum_{i=1}^k FA_{V_i} \quad (1)$$

$$FA_{V_j} = \sum_{s=1}^{(M_{V_j})-1} \sum_{l=1}^{M_{V_j}-s} (A_{j,s} \wedge A_{j,s+l}) \quad (2)$$

Where:  $FA$ = Student conflicts of current timetabling.  $V_i$ = Student conflicts from "Vector"  $i$  of the current Timetabling.  $A_{j,s} \wedge A_{j,s+l}$ = students that simultaneously demand subjects  $s$  and  $s + 1$  inside the "Vector"  $j$ .  $A$  means a student that demands subject  $s$  in a timetabling  $j$ .

Now we can talk about the most important element in the design methodology: the concept of vector. This vector is a binary representation of an event.[9][1] We can construct them as seen on table 3 where each  $v_i$  is a vector who represents event  $e_i$ .

The vectors can be easily added and subtracted allowing them to form *sets*. the symbols used for these sets of vectors are  $V_A, V_B, \dots$  and so on. One characteristic is that the number of vectors sets is related with the number of timeslots offered by the current timetabling. The main idea about vectors is to have a space where we can work with events without assigned them yet to a fixed timeslot. This independent layer of context generalizes in some way the solution process of the CTTTP problem.

Our problem now is to construct a fixed number of vectors sets (usually the cardinality of timeslots set) in order to obtain zero conflict on MMA, LPH and LPA. It is precisely for the vector sets construction that we build a parallel metaheuristic algorithm, but once we have it, if other CTTTP problem can be expressed by means of the Methodology of design then we expect work with it without any modification in the algorithm.

**Table 3** Vector Construction

Events	$e_1$	$e_2$	...	$e_{a-1}$	$e_a$
$v_1$	1	0	...	0	0
$v_2$	0	1	...	0	0
$\vdots$	$\vdots$	$\vdots$	...	$\vdots$	$\vdots$
$v_{a-1}$	0	0	...	1	0
$v_a$	0	0	...	0	1

### 2.3 Parallel Computing and Cellular Genetic Algorithms

The main objective of parallel computing is to execute code concurrently on different processors i.e in the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem for example: To be run using multiple CPUs, To solve a problem broken into discrete parts that can be executed concurrently and instructions from an algorithm executed simultaneously on different CPUs [10].

Also we use a genetic algorithm. A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution.[11] This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover [11].

With these two concepts we built a Cellular Genetic Algorithm (cGA). the cellular genetic algorithm was initially designed for working in massive parallel machines composed of many processors executing simultaneously the same instructions on different data.[12]. The first cGA model know was proposed by Robertson in 1987 [13] implemented on a CM1 computer. It was a model were all steps of GA algorithm(selection, replacement, recombination and mutation) were executed in parallel.[12]. This approach has shown great execution speed as well as better fitness performance against sequential or canonical GA.

Many researchers still think about the equivalence between cGA and massive parallel machines. Today with technologies like Java Threads or CUDA cores, we do not need a massive parallel computer in order to build and execute a cGA.



The cGA model simulates the natural evolution from the point of view of the individual. The essential idea of this model is to provide the population of a special structure defined as a connected graph, where each vertex is a common GA individual or *Cell* that is only allowed to communicate with its nearest neighbours. Particularly, individuals are conceptually in a toroidal mesh and are only allowed to recombine with close individuals.[12] An example of this type of interaction can be seen on figure 2.

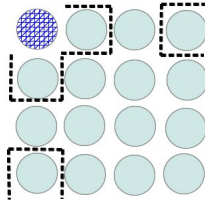


Fig. 2 Simple toroidal population and interaction

The neighborhood of a specific individual on the cellular grid is overlapped by its neighbors. This ensures that good traits and characteristics can travel throughout the grid. In cGAs the reproductive cycle is executed inside the neighborhood of each individual and, generally, consists in selecting among its neighbors with a certain criterion (Tournament selection or Roulette wheel) a parent with which the cell can apply any recombination operators and finally update its own genetic material.

The mutation is simply performed by selecting randomly one cell and minimum change its genetic material. This reproduction cycle can be executed in parallel, executing each cell in a different java thread or CUDA core. A pseudo-code of canonical cGA proposed by Alba Et.al [12] can be seen on algorithm 1.

---

#### Algorithm 1 Pseudo-code of a canonical cGA

---

```

1: procEvolve(cga)
2: GenerateInitialPopulation(cga.pop)
3: Evaluation(cga.pop)
4: while !StopCondition() do
5:   for individual ← 1 to cga.popsi do
6:     neighbors ← CalculateNeighborhood(cga, position(individual));
7:     parents ← Selection(neighbors);
8:     offspring ← Recombination(cga.Pc, parents);
9:     offspring ← Mutation(PM);
10:    Replacement(position(individual), auxiliaryPop, offspring);
11:   end for
12:   cga.pop ← auxiliaryPop;
13: end while
14: return Best(cga.pop)

```

---

**Table 4** Cell Codification

	$e_1$	$e_2$	$\dots$	$e_l$
$cell_1$	$V_B$	$V_D$	$\dots$	$V_B$
$cell_2$	$V_A$	$V_B$	$\dots$	$V_C$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$cell_n$	$V_D$	$V_A$	$\dots$	$V_C$

From algorithm 1 we seen that the cGA algorithm not differ greatly from a sequential GA. The differences of cGA approach are simply the Grid/Toroidal interaction and its parallelism.

#### 2.4 Combining methodology of Design with cGA for the CTTP problem

As seen in previously sections the definition of the cGA have similar operators and parameters as a Sequential GA(sGA). In this section we define the codification, operators and parameters used as well as several details of our grid configuration.

We use the Methodology of Design approach shown in section 2.2 in order to have a CTTP solver to test over different CTTP type instances. We have for each instance 3 list MMA,LPH and LPA. The construction of each of these lists is beyond the scope and purpose of this article. These list ensures by design that every 3-tuple  $(e, t, p)$  will be an allowed selection so it can be applied to reality. The main optimization exercise is to minimize the conflict of students  $S \subseteq A$  by means of the permutation of the events/vectors into timeslots/Vector-sets and the MMA matrix.

The Codification of each Cell or individual is an array of Integer values each integer represent an ID of a Vector set and its size is equal that the cardinality of the event set. The number of Vector sets is defined by the cardinality of the desire timeslots set. So basically from table 4 we can read for the Cell 1: *event* 1 is assigned into *Vector – set* B, *event* 2 is assigned into *Vector – set* D... and so on.

The operators used in each cell are simple. For Selection we use Roulette wheel so every neighbour may have the chance to reproduce with the selected cell but better solutions have more probability to do it. Given the integer representation the recombination operator is single point crossover, in each generation a random crossover point is selected so the genetic material of parents is interchanged from it. The Mutation operator is done at the final of each generation, where we randomly selects a cell and a single integer to change. Also we implement a form of Elitism: in each execution the best cell from the grid will not be modify it all i.e the best cell can interact and update genetic information to its neighbors but no one neighbor can change its own genetic material.

The Parameters used are: for stop criteria we use a fixed number of functions points (a function point means a single decoding of the information of the cell and its execution in the fitness function), for recombination and mutation we use a percentage fixed by the user.

For the grid configuration we divide our population (cells) into several islands. At the beginning of each iteration, all the islands send the cells of their first/last column/row to their neighbor islands. After receiving the individuals from the neighbors, a sequential cGA is executed in each island/subpopulation (figure). This approach has been documented by Alba et al [12] with good results. Finally the neighborhood model used by each cell and island is the NEWS model (North, West, East, South) similar to figure 3. Each island is executed in a JAVA Thread and synchronously waits for all the other islands ends a generation to interchange cells to continue.

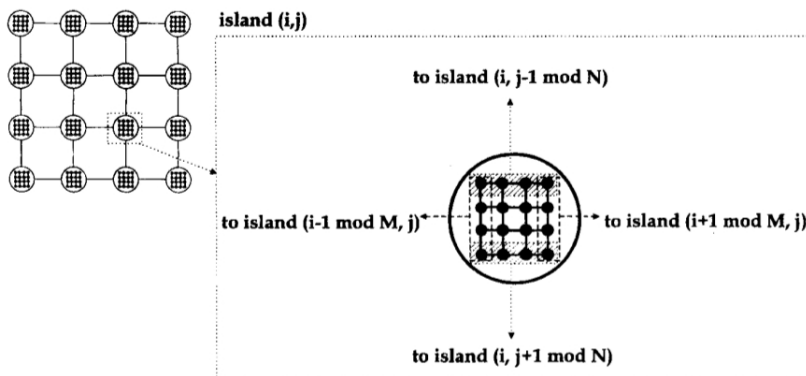


Fig. 3 Grid configuration model used. From Alba et. al

### 3 Experiments and Results

Once we have cGA proposed algorithm we can do several experiments in order to find the speed-up or performance difference between our cGA and sGA. In this section we will explain each experiment as well as the characteristics of the used benchmark.

#### 3.1 Instances Used

We chose a set of well-known instances for the TTP such as ITC2002, ITC2007 from PATAT. The timetabling problem instances ITC 2002 has been designed by Ben Paechter for the Metaheuristics Network.[15] It is a reduction of a typical university course timetabling problem. It consists of a set of events to be scheduled in 45 timeslots (5 days of 9 hours each), a set of rooms in which

events can take place, a set of students who attend the events, and a set of features satisfied by rooms and required by events. Each student attends a number of events and each room has a size.

The ITC2007 instances has been uses as benchmark for the the second International Timetabling Competition contest sponsored by PATAT and WATT. These instances are similar to the ITC2002 ones but, these instances have 3 more constraints: a subset of valid timeslots for subject, an ordering between subjects and a preferred set of timeslots per subject.

### 3.2 Experimental design

According to Alba et.al [12] the recommendations used to compare parallel-sequential algorithm we will use two algorithms: Our parallel proposed approach (cGA) defined in section 2.4 and its sequential counterpart(sGA). For the cGA we implement 16 isles divided into a 4x4 grid, each island have a inner grid of 16 Cells in a 2D array of 4x4 cells similar to figure 3 making 256 individuals/cells in total with an elitism of 16 cells per generation(one cell per island). For the sGA we can say that it is a normal GA without any special modifications. Its recombination, selection and mutation operators are the same that the proposed cGA. Its Elitism is implemented by selecting the best 16 individuals from a population of 256 elements the same of our cGA.

We use a the *weak speedup* metric proposed by Alba et.al [12] because it compares the parallel algorithm developed by a researcher against his own sequential version. For the experiment we execute 100 independent test in each instance from ITC2002 and ITC2007 with cGA and sGA. Each algorithm executes exactly 1000 functions points before stop, the results are shown on table 5.

Our tables 5 and 6 shown the results for our test over ITC2002 and ITC2007 instances. Our fitness function evaluates the number of conflicts (student conflicts) on the timetabling built by the algorithms, so a less value of fitness means a better solution for the CTTTP. We need to say that these evaluations was made just only considering hard constraints for ITC2002 and ITC2007. The column *Best\_fit* shows the best fitness reached by the Algorithm (sGA or cGA) in our experiments. *Avg\_fit* shows the average fitness from our algorithms over 100 independent test. *std\_devf* shows the standard deviation value for the fitness obtained from our series of test. *Avg.time* shows the average time in seconds needed for the algorithm to finish one single test. *std\_devt* shows the standard deviation value of the time used. Finally the *speedup* column shows the weak speed-up metric proposed by Alba et.al [12].

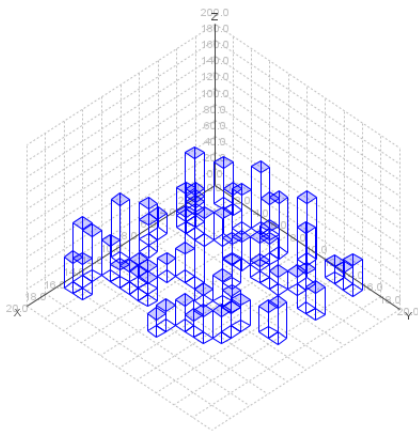
### 3.3 Analysis of results

As it can be seen on table 5 we achieve a better speed in the cGA against the sGA, this is not a surprise because we utilize JAVA Threads so each island

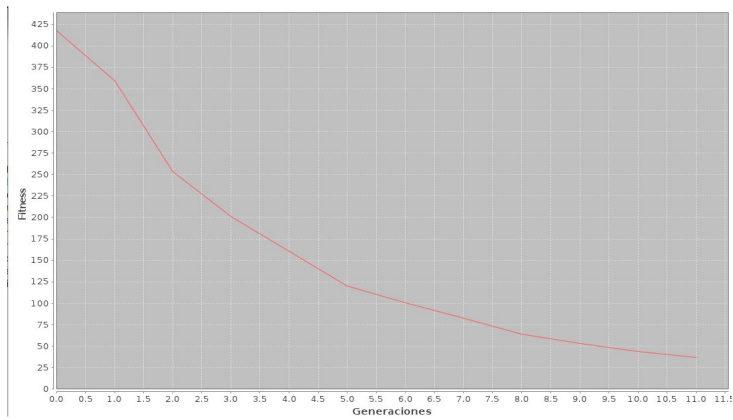
**Table 5** Results experiment ITC2002

Instance		Best_fit	Avg_fit	Std_devf	Avg_time	std_devt	Speedup
ITC2002-1	sGA	281	324.3	18.8	12.59	2.96	
	cGA	149	188	14.9	2.35	0.09	5.35
ITC2002-2	sGA	262	305.5	18.2	7.97	2.50	
	cGA	146	179	14.0	2.97	0.15	2.68
ITC2002-3	sGA	296	330.4	15.3	10.96	2.58	
	cGA	165	202.7	13.0	2.95	0.17	3.71
ITC2002-4	sGA	429	485.5	23.9	8.30	2.25	
	cGA	255	304	22.9	3.01	0.10	2.75
ITC2002-5	sGA	424	480.18	26.57	6.82	2.09	
	cGA	246	293.3	21.4	2.78	0.13	2.45
ITC2002-6	sGA	427	481.8	26.56	7.09	2.24	
	cGA	247	294.8	21.06	2.65	0.15	2.67
ITC2002-7	sGA	419	503.4	30.55	7.21	2.23	
	cGA	238	287.9	24.9	2.63	0.16	2.74
ITC2002-8	sGA	311	371.94	25.14	8.25	2.51	
	cGA	168	210.2	18.65	3.01	0.19	2.74
ITC2002-9	sGA	299	346.8	20.54	12.99	2.82	
	cGA	186	207.9	16.56	3.2	0.20	4.05
ITC2002-10	sGA	285	335.15	21.24	9.67	2.74	
	cGA	176	201.9	14.2	3.0	0.19	3.22
ITC2002-11	sGA	304	350.4	19.4	12.12	2.43	
	cGA	174	208.2	14.56	3.12	0.32	3.88
ITC2002-12	sGA	268	312.1	19.15	10.03	3.12	
	cGA	148	188.5	14.25	3.40	0.52	2.95
ITC2002-13	sGA	343	396.44	24.26	9.51	1.42	
	cGA	186	231.6	19.54	2.30	0.32	4.13
ITC2002-14	sGA	455	520.15	28.17	7.79	0.877	
	cGA	251	313.3	23.20	3.21	0.19	2.42
ITC2002-15	sGA	362	449.5	28.65	8.06	0.8	
	cGA	209	261.4	22.44	3.10	0.22	2.6
ITC2002-16	sGA	304	369.67	20.44	10.14	1.69	
	cGA	191	223.6	18.22	3.70	0.36	2.74
ITC2002-17	sGA	396	468.9	29.22	7.33	2.46	
	cGA	231	289.4	22.19	3.18	0.05	2.30
ITC2002-18	sGA	250	307.14	20.3	12.19	2.55	
	cGA	148	181.5	14.98	3.59	0.06	3.39
ITC2002-19	sGA	408	497.1	28.95	9.45	2.97	
	cGA	224	297.9	23.0	3.60	0.13	2.28
ITC2002-20	sGA	380	449.14	26.35	8.35	2.47	
	cGA	224	266.0	21.54	3.18	0.16	2.62

or sub population is working in a parallel way. However we obtained not only a better speed but a good performance as well, as seen on the results table, our cGA presents a lower fitness value and a lower standard deviation value for both the time and the objective function. This result can be explained because the elitism used in the cellular model. This approach conserves several cell/individual with different genetic value, then in the phase of interchange this genetic material travels over the grid. In the sGA cells preserved by the elitism have similar genetic material. An snapshot of the Celular Grid as well as its performance graph can be seen on fig 4 and 5.



**Fig. 4** Snapshot of cGA grid after a test



**Fig. 5** Performance of cGA over a test

**Table 6** Results experiment ITC2007

Instance		Best_fit	Avg_fit	Std_devf	Avg.time	std_devt	Speedup
ITC2007-1	sGA	1253	1362.4	55.0	11.1	3.0	
	cGA	881	975.3	45.96	3.92	1.05	2.83
ITC2007-2	sGA	1251	1388.5	56.87	11.21	3.06	
	cGA	892	999.0	42.78	4.57	0.37	2.45
ITC2007-3	sGA	434	556.8	57.80	4.79	1.11	
	cGA	215	286.75	27.0	2.66	0.28	1.80
ITC2007-4	sGA	527	619.2	44.0	4.71	1.08	
	cGA	275	342.8	30.55	2.56	0.29	1.83
ITC2007-5	sGA	698	805.3	35.70	9.73	3.04	
	cGA	479	564.4	32.28	4.68	0.42	2.07
ITC2007-6	sGA	723	794.81	36.72	10.73	3.41	
	cGA	480	552.18	33.15	3.8	0.39	2.82
ITC2007-7	sGA	265	334.9	28.96	4.5	1.27	
	cGA	154	187.8	15.5	2.46	0.39	1.82
ITC2007-8	sGA	287	375	36.14	5.75	0.93	
	cGA	147	196.3	18.72	2.5	0.42	2.3
ITC2007-9	sGA	1190	1403.1	69.77	10.52	3.26	
	cGA	860	979.23	52.67	4.29	0.70	2.45
ITC2007-10	sGA	1256	1400.3	53.58	11.05	3.41	
	cGA	899	1011.6	47.38	3.83	0.45	2.88
ITC2007-11	sGA	488	612.84	58.64	4.55	0.97	
	cGA	235	319.21	32.20	2.68	0.14	1.69
ITC2007-12	sGA	433	588.11	65.15	4.83	1.24	
	cGA	249	317.74	28.55	1.56	0.16	3.09
ITC2007-13	sGA	751	843.75	34.24	8.26	2.66	
	cGA	521	590.17	31.19	2.75	0.20	3.00
ITC2007-14	sGA	724	813.11	36.41	8.20	2.62	
	cGA	512	572.86	26.77	2.28	0.08	3.59
ITC2007-15	sGA	231	300.96	30.23	4.67	1.04	
	cGA	131	154.0	12.20	1.43	0.06	3.26
ITC2007-16	sGA	237	326.85	34.47	4.59	0.96	
	cGA	113	147.21	16.82	1.41	0.06	3.255
ITC2007-17	sGA	286	432.15	69.81	3.23	0.21	
	cGA	46	160.15	35.05	1.10	0.05	2.93
ITC2007-18	sGA	865	1000.59	56.14	4.47	0.95	
	cGA	512	635.17	38.87	1.47	0.04	3.04
ITC2007-19	sGA	686	814.96	58.10	7.14	2.24	
	cGA	426	482.5	32.52	1.88	0.05	3.79
ITC2007-20	sGA	738	879.07	59.34	10.53	2.86	
	cGA	430	497.52	38.96	2.38	0.05	4.42
ITC2007-21	sGA	761	847.17	28.99	12.97	4.34	
	cGA	516	614.27	26.82	3.03	0.07	4.28
ITC2007-22	sGA	1400	1556.15	59.93	16.87	4.98	
	cGA	1084	1185.74	75.48	3.84	0.08	4.39
ITC2007-23	sGA	2595	2882.6	114.88	7.93	2.42	
	cGA	1902	2137.85	88.12	2.36	0.07	3.36
ITC2007-24	sGA	757	886.48	52.31	8.78	2.61	
	cGA	453	527.9	35.12	2.41	0.06	3.64

## 4 Conclusions and future work

This paper has presented a comparison between a parallel computing model for a genetic algorithm and a sequential genetic algorithm for the context of CTTTP problem. The cGA proposed aims to help the researcher of CTTTP to obtain good solutions in a relative short lapse of time against sequential Genetic Algorithm. The cGA proposed utilizes a toroidal grid configuration that ensures the interchange of good genetic material over the whole population. The proposed approach utilizes the Methodology of Design model to generalize the process of CTTTP solution by means of generic structures, this model ensures that if any other type of CTTTP can be produce the same structures the proposed algorithm will be capable to work over it with similar performance. This paper has explored the use of parallel computing for a well known set of instances for the CTTTP. The cGA has show a better performance against a sequential GA in a better time allowing the researcher to accelerate the process of gathering statistical data.

For future work we propose to apply this approach to Unitime.org instances. Also we Propose to apply this model over not only Java Threads but CUDA Nvidia cores in order to perform a faster experimentation.

## Acknowledgement

Authors thanks the support received from the Consejo Nacional de Ciencia y Tecnologia (CONACYT) México.

## References

1. J.A. Soria-Alcaraz, Diseño de horarios con respecto al alumno mediante tcnicas de cmputo evolutivo. Master's thesis, Instituto Tecnologico de Len (2010)
2. M. Adriaen, P. Causmaecker, Proceedings PATAT **1**, 330 (2006)
3. T.B. Cooper, J.H. Kingston, The compeity of timetable construction problems. Ph.D. thesis, The University of Sydney (1995)
4. R.J. Willemen, School timetable construcion: Algorithms and complexity. Ph.D. thesis, Institutefor Programming research and Algorithms (2002)
5. R. Martinez, Q. Aguilera, COMCEV **1**, 169 (2005)
6. S.E. Conant-Pablos, D.J. Magaa-Lozano, H. Terashima-Marin, MICAI Mexican international conference on artificial intelligence **1**, 408 (2009)
7. A. Schaerf, L.D. Gaspero, Practice and Theory of Automated Timetabling, PATAT, Springer-Verlag Berlin Heidelberg, LNCS 3867 **1**, 40 (2007)
8. R. Lewis, Metaheuristics for university course timetabling. Ph.D. thesis, University of Notthingham. (August 2006)
9. J.A. Soria-Alcaraz, M. Carpio, H. Puga, Dcima Primera Reunin de Otoo de Potencia, Electrnicna y Computacin del IEEE, XI ROPEC, Morelia **1**, 24 (2009)
10. B. Barney. Introduction to parallel computing. URL [https://computing.llnl.gov/tutorials/parallel\\_comp/](https://computing.llnl.gov/tutorials/parallel_comp/)
11. J. Holland, The University of Michigan Press, Ann Harbor (1975)
12. E. Alba, B. Dorronsoro, Cellular Genetic Algorithms **1**, Springer Science+Business Media, LLC (2008)
13. G. Robertson, in *In Proc. of the Second International Conference on Genetic Algorithms(ICGA)* (1987), pp. 140–147



14. E. Alba, B. Dorronsoro, Cellular Genetic Algorithms Springer Science+Business Media, LLC (2008)
15. B. Paechter. The course timetable problem (2002). URL <http://www.metaheuristics.net/index.php?main=4&sub=44>

---

# A Combined Local Search and Integer Programming Approach to the Traveling Tournament Problem

Marc Goerigk · Stephan Westphal

Received: date / Accepted: date

**Abstract** The *traveling tournament problem* is a well-known combinatorial optimization problem with application to sport leagues scheduling, that sparked intensive algorithmic research over the last decade. With the *Challenge Traveling Tournament Instances* as an established benchmark, the most successful approaches to the problem use meta-heuristics like tabu search or simulated annealing, partially heavily parallelized. Integer programming based methods on the other hand are hardly able to tackle larger benchmark instances.

In this work we present a hybrid approach that draws on the power of commercial integer programming solvers as well as the speed of local search heuristics. Our proposed method feeds the solution of one algorithm phase to the other one, until no further improvements can be made. The applicability of this method is demonstrated experimentally on the *galaxy* instance set, resulting in currently best known solutions for most of the considered instances.

**Keywords** traveling tournament problem · tabu search · integer programming · sports scheduling

## 1 Introduction

Before the start of each season, every sports league is faced with the problem of scheduling the games among their teams such that a variety of require-

---

Partially supported by grant SCHO 1140/3-2 within the DFG programme *Algorithm Engineering*.

---

M. Goerigk and S. Westphal  
Institut für Numerische und Angewandte Mathematik  
Universität Göttingen Lotzestr. 16-18  
D-37083 Göttingen  
Germany  
Tel.: +49-551-3920035  
Fax: ++49-551-393944  
E-mail: s.westphal@math.uni-goettingen.de

ments is taken into account. The planners have to synchronize every feasible schedule to the availability restrictions of the sports sites, the most interesting games have to be matched to available TV slots and specific league-requested matchups have to be taken care of. Additionally, there is a wish to minimize the total distances driven by the teams over the season, which becomes even more important for bigger countries.

In this paper we will focus on the problem of minimizing the total distances driven by the teams in the way addressed by the well-known Traveling Tournament Problem (TTP). This sports scheduling problem which has been introduced by Easton et al. [ENT01] is inspired by Major League Baseball and is considered to be practically hard to solve.

### 1.1 Sports Scheduling and the Traveling Tournament Problem

*Sports Scheduling* in general deals with the design of tournaments. A *single round robin tournament* on  $n$  teams, where  $n$  is an even number, consists of  $(n - 1)$  rounds (also called *slots*). In each round  $n/2$  games, which are themselves ordered pairs of teams, take place. Every team has to participate in one game per round and must meet every other team exactly once. It is standard to assume  $n$  to be even since in sports leagues with  $n$  being odd, usually a dummy team is introduced, and whoever plays it has a day off, which is called a *bye*. For scheduling single round robin tournaments a rather general and useful scheme called the *canonical schedule* has been known in sports scheduling literature for at least 30 years [dW81]. It is based on the polygon/circle method, which was first suggested by Kirkman in 1847 [Kir47]. One can think of Kirkman's method as a long table at which  $n$  players sit such that  $n/2$  players on one side face the other players seated on the other side of the table. Every player plays a match against the person seated directly across the table. The next round of the schedule is obtained when everyone moves one chair to the right with the crucial exception that there exists one person at the end of the table who never moves and always maintains the seat from his or her first round. Note that this method only specifies who plays whom when and not where. The canonical schedule introduced by de Werra defines for each of the encounters specified by the method described above, at whose site they take place such that the number of successive home or away games is minimized [dW81].

A *double round robin tournament* on  $n$  teams consists of  $2(n-1)$  rounds and every team must meet every other team twice: once at its own home venue (*home game*) and once at the other team's venue (*away game*). A popular policy in practice is to obtain a double round robin tournament from a single round robin tournament by *mirroring*, that is repeating the matches of round  $k$  for  $k = 1, \dots, n - 1$  in round  $k + n - 1$  with changed home field advantage. Consecutive home games are called a *home stand* and consecutive away games form a *road trip*. The *length* of a home stand or road trip is the number of opponents played (and not the distance traveled).

In this work we consider the *traveling tournament problem* (TTP) as described in [ENT01]:

**Definition 1 (The Traveling Tournament Problem TTP( $k$ ) [ENT01])**

Let a set of  $n$  teams and a distance matrix  $(d_{ij})$  be given. Find a feasible double round robin tournament of the teams satisfying the following condition:

1. The length of any home stand is at most  $k$ .
2. The length of any road trip is at most  $k$ .
3. Game  $j$  at  $i$  is not followed immediately by game  $i$  at  $j$ .
4. The sum of the distances traveled by the teams is minimized.

As it is the case in most real-world applications, we henceforth assume  $k = 3$  throughout the paper. The third requirement is known as *no-repeater constraint*.

An example instance from [Tri11] is given in Table 1: For every team, the distance to every other team is known. Table 2 shows the corresponding optimal solution with objective 416: On day 1, team 1 plays away against team 4, then away against team 2, at home against team 3 on day 3 and so on. Note that, as demanded, there is no home stand or road trip with length larger than 3.

	Team 1	Team 2	Team 3	Team 4
Team 1	0	10	15	34
Team 2	10	0	22	32
Team 3	15	22	0	47
Team 4	34	32	47	0

**Table 1** Instance *galaxy4*.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6
Team 1	-4	-2	3	4	2	-3
Team 2	3	1	4	-3	-1	-4
Team 3	-2	-4	-1	2	4	1
Team 4	1	3	-2	-1	-3	2

**Table 2** Optimal solution to *galaxy4*.

## 1.2 Previous Work

So far, most efforts concerning the TTP have led to a variety of algorithms aiming to minimize the total distance driven by the teams. Kendall et al.

[KKRU10] provide a good overview of the work done on the TTP and sports scheduling in general. Just to mention a very few examples, hybrid algorithms with constraint programming (CP) exist by Benoist et al. [BLR01] who additionally use Lagrange relaxation. Easton et al. [ENT01] merge CP with integer programming while Henz [Hen04] combines CP with large neighborhood search. Anagnostopoulos et al. [AMHV06], Hentenryck and Vergados [vHV06], Gaspero and Schaerf [GS07] and Lim et al. [LRZ06] propose neighborhood search-based algorithms, whereas Ribeiro and Urrutia [Rib11] focus on the special class of constant distance TTP where break maximization is equivalent to travel distance minimization.

On the theoretical side, Thielen and Westphal settled the complexity by showing that the TTP is strongly *NP*-hard [TW11]. Miyashiro et al. [MMI08] provide a  $2 + (9/4)/(n - 1)$  approximation for the intensively studied special case  $k = 3$  by means of the *Modified Circle Method*, a variation of the canonical schedule. In [YIMM09] Yamaguchi et al. obtain an algorithm with approximation ratio  $(2k - 1)/k + O(k/n)$  for  $k \leq 5$  and  $(5k - 7)/(2k) + O(k/n)$  for  $k > 5$ . Again they make use of the canonical schedule, now refined such that the teams are ordered around the 'table' such that most of the distances driven are part of a near optimal traveling salesman tour which clearly has positive effects on the length of many distances traveled. As  $k \leq n - 1$ , they showed this way that a constant factor approximation for any choice of  $k$  and  $n$  exists. However, they did not show how this factor looks exactly. This was done later by Westphal and Noparlik [WN12], whose algorithm was also able to compute new bests for all galaxy instances with at least 22 teams.

### 1.3 Contribution

Due to its computational difficulty, exact solution approaches already fail from a size of 12 teams on [Tri11]. In this paper we propose a combination of local search heuristics with integer programming methods to overcome local optima. In an experimental evaluation, we were able to calculate new best known solutions on most instances of the considered benchmark set.

### 1.4 Overview

In Section 2, we describe the algorithmic details of our heuristic approach to the traveling tournament problem, which we evaluate extensively in Section 3. Section 4 concludes the paper and gives an outlook on further research.

## 2 A Combined Local Search and Integer Programming Heuristic

The heuristic we consider consists of two separate phases: The local search phase, which is described in Section 2.1, and the integer programming phase

as described in Section 2.2. After explaining these phases, we give an overview of the whole algorithm in Section 2.3

The motivation for combining these methods is the following: Local search heuristics are an effective means to improve solutions even for large instances, but local minima pose complications. When the search is not able to leave such a minimum anymore, it helps to use a view that does not consider the same neighborhood as before. Therefore, breaking up the current solution structure by applying integer programming methods is able to provide the local search with a fresh start from an even improved solution. Within the VLNS classification framework [AEOP02], we are using restrictions on the original problem, but they are not likely solvable in polynomial time.

## 2.1 Phase 1: Tabu Search

The first part of the proposed algorithm consists of a local search phase. This might be steepest descent, simulated annealing, or any other suitable (meta-)heuristic. In this work we specifically considered a tabu search heuristic that uses the standard neighborhood as described in [AMHV06] and [DGS05]. The five neighborhood search moves are presented in Table 3.

Neighborhood	Input	Effect
Swap Homes	$t_1, t_2 \in T$	Swap home/away pattern for matches between $t_1$ and $t_2$ . No further adjustments necessary.
Swap Teams	$t_1, t_2 \in T$	Swap all matches of teams $t_1$ and $t_2$ . Adjust opponents accordingly.
Swap Days	$d_1, d_2 \in D$	Swap two days. No further adjustments necessary.
Swap Teams Partial	$t_1, t_2 \in T, d \in D$	Swap opponents of teams $t_1$ and $t_2$ on day $d$ . This will cause more swaps to resolve resulting conflicts.
Swap Days Partial	$t \in T, d_1, d_2 \in D$	Swap the opponents of team $t$ on days $d_1$ and $d_2$ . This will cause more swaps between these days to reestablish feasibility.

**Table 3** Local neighborhood for tabu search algorithm.

Furthermore, we the following algorithm specifications are used:

1. **Neighborhood:** In every iteration, the whole neighborhood is considered. That is, all moves of the types given in Table 3 are evaluated, and the best non-tabu move is chosen.

2. **Tabu List:** The list contains whole solutions, not just partial properties. The list length is dynamic in the sense that every considered solution becomes tabu, until a new global optimum is found, which triggers the list to be cleaned.
3. **Stopping criterion:** If a prescribed number *max idle iterations* of iterations have passed without finding a new global optimum, the search is reset: The current global optimum is restored and the tabu list cleaned. After a given number *max restarts* of resets, the search is aborted.
4. **Objective:** In order to expand the search space to infeasible solutions, we add a penalty to the original problem objective for every violated home stand, road trip and no-repeater constraint (Constraints 1.-3. in the problem definition), but do not forbid schedules that violate these constraints.
5. **Dynamic Penalty:** The infeasibility penalty is dynamically adapted throughout the search process, such that sequences of feasible solutions decrease the penalty, and sequences of infeasible solutions increase it.
6. **Dominance:** Feasible solutions in the neighborhood that are better with respect to the original objective than the current best solution are always preferred to infeasible solutions, even if their modified objective might be better.

Though there are of course many more possibilities concerning the fine-tuning of the tabu search, this approach turned out to be most promising in preliminary experiments.

## 2.2 Phase 2: Integer Programming

In order to leave a local optimum of the local search procedure, we apply integer programming methods that do not depend on the neighborhood as presented in Table 3. For our experiments, we use a variation of the simple formulation presented in [Rib11] with  $\mathcal{O}(n^3)$  variables. The variables  $x_{ijk}(i, j = 1, \dots, n, k = 1, \dots, 2n - 2)$  represent the decision if team  $i$  plays away against team  $j$  on day  $k$ , while  $y_{tij}(i, j, t = 1, \dots, n)$  denotes that team  $t$  travels from team  $i$  to team  $j$  anywhere in the schedule.

$$\min \sum_{i,j=1}^n d_{ij}x_{ij1} + \sum_{t,i,j=1}^n d_{ij}y_{tij} + \sum_{i,j=1}^n d_{ji}x_{ij,2n-2} \quad (1)$$

$$x_{iik} = 0 \quad (i = 1, \dots, n, k = 1, \dots, 2n - 2) \quad (2)$$

$$\sum_{j=1}^n (x_{ijk} + x_{jik}) = 1 \quad (i = 1, \dots, n, k = 1, \dots, 2n - 2) \quad (3)$$

$$\sum_{k=1}^{2n-2} x_{ijk} = 1 \quad (i, j = 1, \dots, n, i \neq j) \quad (4)$$

$$\sum_{l=0}^3 \sum_{j=1}^n x_{ij,k+l} \leq 3 \quad (i = 1, \dots, n, k = 1, \dots, 2n - 2 - 3) \quad (5)$$

$$\sum_{l=0}^3 \sum_{i=1}^n x_{ij,k+l} \leq 3 \quad (j = 1, \dots, n, k = 1, \dots, 2n - 2 - 3) \quad (6)$$

$$x_{ijk} + x_{jik} + x_{ij,k+1} + x_{ji,k+1} \leq 1 \quad (i, j = 1, \dots, n, k = 1, \dots, 2n - 3) \quad (7)$$

$$z_{iik} = \sum_{j=1}^n x_{ijk}, \quad (i = 1, \dots, n, k = 1, \dots, 2n - 2) \quad (8)$$

$$z_{ijk} = x_{ijk} \quad (i, j = 1, \dots, n, i \neq j, k = 1, \dots, 2n - 2) \quad (9)$$

$$y_{tij} \geq z_{tik} + z_{tj,k+1} - 1 \quad (t, i, j = 1, \dots, n, k = 1, \dots, 2n - 3) \quad (10)$$

$$x_{ijk}, z_{ijk}, y_{tij} \in \{0, 1\} \quad (t, i, j = 1, \dots, n, k = 1, \dots, 2n - 2) \quad (11)$$

As there is little hope in solving this program directly, we divide it into two smaller problems based on the provided input solution: Optimizing the *home-away-pattern* (HA-opt) and optimizing the rest of the schedule with fixed home-away-pattern (non-HA-opt). Both subproblems are described in the following.

*Optimize Home-Away-Pattern.* In order to optimize the home-away-pattern for a given solution, we have to fix the decisions when two teams face another. Let  $\tilde{x}$  be the given solution to the  $x$  variables. We now add the Constraint (12) to the original problem:

$$x_{ijk} + x_{jik} = \tilde{x}_{ijk} + \tilde{x}_{jik} \quad (i, j = 1, \dots, n, k = 1, \dots, 2n - 2) \quad (12)$$

By doing so, we fix if team  $i$  plays against team  $j$  on day  $k$ , but leave the venue open. In terms of the solution format as described in Table 2, we restrict the optimization process to the signs  $+$ ,  $-$  of the schedule.

The resulting problem is also known as the *Timetable Constrained Distance Minimization Problem*, and has been introduced in [RT06].

*Fix Home-Away-Pattern.* The resulting second partial problems consists of finding optimal team matchups, when travel and home days are fixed for every team. As before, let  $\tilde{x}$  be the given solution. We add Constraints (13) and (14) to the problem formulation:

$$\sum_{i=1}^n x_{ijk} = \sum_{i=1}^n \tilde{x}_{ijk}, \quad (j = 1, \dots, n, k = 1, \dots, 2n - 2) \quad (13)$$

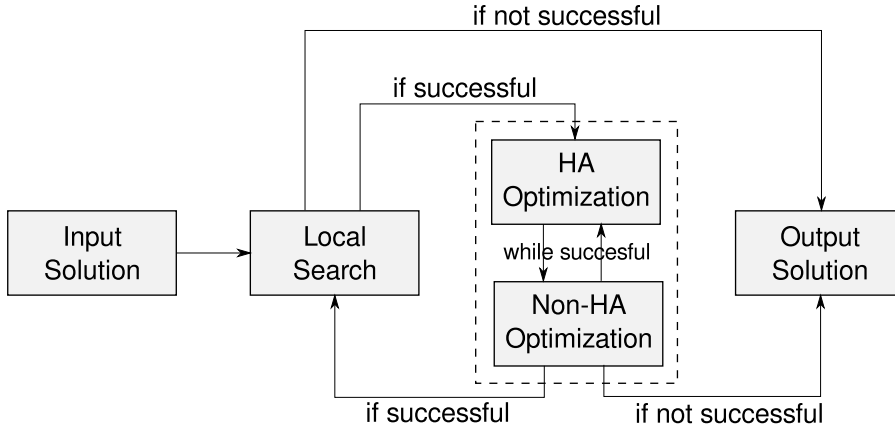
$$\sum_{j=1}^n x_{ijk} = \sum_{j=1}^n \tilde{x}_{ijk}, \quad (i = 1, \dots, n, k = 1, \dots, 2n - 2) \quad (14)$$

These constraints force a team to play away if this is the case for the input solution, and to play at home otherwise. For the solution format of Table 2, this means that we fix the signs, but can change the actual opponents.



### 2.3 Phase Combination

Having described the two phases separately, we now focus on how to combine them. In Figure 1, a diagram of the proposed algorithm structure is given.



**Fig. 1** Algorithm overview.

The solution process needs to be provided with a starting solution, whose creation is described in the experimental setup of Section 3. First, a local search is performed until its stopping criterion is satisfied. If it was possible to improve the solution, we pass it to the second phase, and end the solution process otherwise. In the second phase, we repeatedly optimize the home-away-pattern and its counterpart problem, until both partial problems are not able to improve the current solution anymore. If the phase was able to improve the given solution at all, we repeat phase 1, and end the algorithm otherwise.

Additionally, we propose another feature that turned out to be valuable in our experiments. The local search phase considers many solutions that may be inferior to the current best solution, but are structurally so different that a home-away optimization might create a new current best solution. Therefore, even if the local search was not able to improve its input, we feed the last found local optimum to the second phase. Only if this does not create a solution that is better than the current best, we consider the local search phase as being not successful.

### 3 Experimental Results

In this section we present experimental experience on the performance of the proposed combination of local search and integer programming techniques.

*Environment* All experiments were conducted on a PC with 99 GB main memory and an Intel Xeon X5650 processor, running with 6 cores at 2.66 GHz and

12MB cache. All code is written in C++ and has been compiled with g++ 4.4.3 and optimization flag -O3. For the integer programming phase, we used the Gurobi optimizer [Inc11] in version 4.5.

*Setup* The conducted experiment was scheduled the following way:

1. The considered benchmark set are the *galaxy* instances from [Tri11]. As instances with team size up to 10 have already been solved to optimality, we only used those which are larger.
2. We construct canonical schedules [dW81] as input solutions using the rotation scheme as described in [WN12]. Of these, only the best three are used per instance.
3. The described algorithm is run for the resulting 45 solutions. The parameter set we used is presented in Table 4. Note that the factor we multiply the infeasibility penalty of the local search with is chosen in a way that penalty increase faster than they decrease.
4. For the best solution found so far for every problem instance, we restart the algorithm with the second parameter set of Table 4.

Teams	12-14	16-22	24-36	38-40
max idle iterations, first	10,000	1,000	500	250
max restarts, first	2			
max idle iterations, second	100,000	4,000	1500	500
max restarts, second	3			
starting inf. penalty	input objective /1,000			
inf. penalty factor	0.97 and 1/0.93			
timelimit HA-opt	1800s			
timelimit Non-HA-opt	3600s			

**Table 4** Parameter choice.

*Results* We summarize the achieved results in Table 5. In the first column, the instance size in terms of number of teams is given, followed by the objective value of the best currently known solution as of January 2012 in column two. We then present the initial objective values of the three best initial solutions, and the improved objective value after the first run of the algorithm, together with the corresponding number of algorithm phases. As described in the *setup* paragraph, these solutions are further improved by restarting the algorithm.

In the last column, we present a lower bound for each instance, and mark bounds that were previously unknown with an asterisk (\*). These bounds are found by calculating an optimal tour for each team separately, and summing up the respective tour lengths. Finding these tours was done using a flow-based integer programming formulation with a timelimit of 3600 seconds, which was hit in only a few cases.

As can be seen, it was possible to further improve the currently best solution in 9 out of 15 cases by at least 0.1%, and up to 3.2%. There seems to be

Teams	Best Known	Initial Solutions	1st Improvement	Number of phases	2nd Improvement	LB
12	7197	8223	7642	2	7555 (5.0%)	6933
		8364	7720	2		
		8408	7639	2		
14	10918	13017	11566	2	11552 (5.8%)	10221
		13047	12008	2		
		13126	11636	2		
16	14900	16257	15769	2	15704 (5.4%)	13619*
		16315	15897	2		
		16372	16094	2		
18	20907	21635	21426	2	21346 (2.1%)	19050*
		21658	21437	2		
		21728	21346	2		
20	26289	28237	26921	4	26749 (1.7%)	23738*
		28332	27121	3		
		28368	26749	4		
22	35516	35832	35624	4	35584 (0.2%)	31461*
		35882	35812	2		
		36021	35584	2		
24	45728	45962	45671	3	<b>45657</b> (-0.2%)	41287*
		46029	45657	2		
		46130	45705	2		
26	60962	61617	58991	4	<b>58991</b> (-3.2%)	53802*
		61634	59889	4		
		61703	59894	4		
28	77577	77683	77381	2	<b>77320</b> (-0.3%)	69992*
		77732	77320	3		
		77736	77361	3		
30	96765	97270	96756	2	<b>96710</b> (-0.1%)	88831*
		97321	96712	2		
		97384	96710	4		
32	120683	122567	120053	3	<b>119996</b> (-0.6%)	108187*
		122655	120130	4		
		122661	119996	4		
34	147742	148194	147644	3	<b>147612</b> (-0.1%)	133976*
		148223	147612	4		
		148363	147763	3		
36	173640	174475	173532	3	<b>173532</b> (-0.1%)	158363*
		174595	173716	3		
		174734	173670	2		
38	209463	212706	205876	3	<b>204980</b> (-2.1%)	188935*
		212809	204980	8		
		213072	205870	7		
40	249002	249976	247017	9	<b>247017</b> (-0.8%)	226794*
		249996	248295	3		
		250081	248223	6		

**Table 5** Results overview.

a connection between the instance size and the number of algorithm phases, which can be explained by the increasingly larger neighborhood, which makes it more difficult for the local search to find an actually close, better solution. Therefore, combining local search with integer programming methods is especially beneficial for large instances.

Of course, many more experimental setups are possible to pursue: As an example, we not only examined the best three initial solutions of galaxy22, but all 22 of them. As a result a solution of objective 35467 was found, which is 0.1% below the current best solution.

#### 4 Conclusion and Outlook

We proposed an algorithm to the traveling tournament problem that is able to overcome local optima arising in local search heuristics by solving integer optimization subproblems. Its applicability is demonstrated by an extensive experiment on a well-known benchmark set, resulting in new best known solutions for all instances of size greater or equal to 24. Experiments with further instance sets are currently being conducted.

It seems promising to use further parameter setups and local search heuristics to find better solutions than presented in this work. Also, we plan to use the described algorithmic ideas to include *robustness* issues in the schedule design, i.e., to find tournament schedules that are insensitive to disruptions like bad weather conditions that may increase travel time between two venues, or even render a stadium unusable for certain days.

#### References

- [AMHV06] Aris Anagnostopoulos, Laurent Michel, Pascal Van Hentenryck, and Yannis Vergados. A simulated annealing approach to the traveling tournament problem. *J. Scheduling*, 9(2):177–193, 2006.
- [AEOP02] Ravindra K. Ahuja, zlem Ergun, James B. Orlin, and Abraham P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(13):75 – 102, 2002.
- [BLR01] T. Benoist, L. Laburthe, and B. Rottembourg. Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems. In *Proceedings of the 3rd International Workshop on the Integration of AI and OR Techniques (CP-AI-OR)*, pages 15–26, 2001.
- [DGS05] Luca Di Gaspero and Andrea Schaerf. A tabu search approach to the traveling tournament problem. In *Proceedings of the 6th Metaheuristics International Conference (MIC-2005)*, Vienna, Austria, August 2005. Available as electronic proceedings.
- [dW81] D. de Werra. Scheduling in Sports. In P. Hansen, editor, *Studies on graphs and integer programming*, volume 11, pages 381–395. Annals of Discrete Mathematics, North Holland, 1981.
- [ENT01] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. *Principles and Practice of Constraint ProgrammingCP 2001*, pages 580–584, 2001.
- [GS07] Luca Di Gaspero and Andrea Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13:189–207, April 2007.
- [Hen04] M. Henz. Playing with constraint programming and large neighborhood search for traveling tournaments. In *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 23–32, 2004.
- [Inc11] Gurobi Optimization Inc. Gurobi optimizer, 2011. Version 4.5.

- [Kir47] Thomas P. Kirkman. On a problem in combinations. *The Cambridge and Dublin Mathematical Journal*, 2:191–204, 1847.
- [KKRU10] Graham Kendall, Sigrid Knust, Celso C. Ribeiro, and Sebastián Urrutia. Invited review: Scheduling in sports: An annotated bibliography. *Comput. Oper. Res.*, 37:1–19, January 2010.
- [LRZ06] A. Lim, B. Rodrigues, and X. Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operations Research*, 174:1459 – 1478, 2006.
- [MMI08] R. Miyashiro, T. Matsui, and S. Imahori. An approximation algorithm for the traveling tournament problem. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2008.
- [Rib11] Celso C. Ribeiro. Sports scheduling: Problems and applications. *International Transactions in Operational Research*, 2011.
- [RT06] Rasmus Rasmussen and Michael Trick. The timetable constrained distance minimization problem. In J. Beck and Barbara Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3990 of *Lecture Notes in Computer Science*, pages 167–181. Springer Berlin / Heidelberg, 2006.
- [Tri11] Michael Trick. Challenge traveling tournament problems benchmark, 2011. <http://mat.gsia.cmu.edu/TOURN/>.
- [TW11] C. Thielen and S. Westphal. Complexity of the traveling tournament problem. *Theoretical Computer Science*, 412(4-5):345–351, 2011.
- [vHV06] P. van Hentenryck and Y. Vergados. Traveling tournament scheduling: A systematic evaluation of simulated annealing. *LNCS*, 3990:228–243, 2006.
- [WN12] S. Westphal and K. Noparlik. A 5.875-approximation for the traveling tournament problem. *Annals of Operations Research*, 2012.
- [YIMM09] D. Yamaguchi, S. Imahori, R. Miyashiro, and T. Matsui. An improved approximation algorithm for the traveling tournament problem. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, volume 5878 of *LNCS*, pages 679–688, 2009.

---

## Real-life Curriculum-based Timetabling

Tomáš Müller · Hana Rudová

June 2012

**Abstract** This paper presents an innovative approach to curriculum-based timetabling. Curricula are defined by a rich model that includes optional courses and course groups among which students are expected to take a subset of courses. Transformation of the curriculum model into the enrollment model is proposed and a local search algorithm generating corresponding enrollments is introduced. This enables curriculum-based timetabling in any existing enrollment-based course timetabling solver. The approach was implemented in a well established enrollment-based course timetabling system UniTime. The system has been successfully applied in practice at the Faculty of Education at Masaryk University for about 7,500 students and 260 curricula. Experimental results related with this problem are demonstrated for two semesters.

**Keywords** Course timetabling · Curriculum-based timetabling · Local search · UniTime

### 1 Introduction

Curriculum-based timetabling belongs to the class of university course timetabling problems (Burke and Petrovic, 2002; Lewis, 2008). Much research has been done in the area of curriculum-based timetabling (Di Gaspero et al, 2007; Bonutti et al, 2012), typically using a base curriculum model. In this

---

T. Müller  
Space Management and Academic Scheduling, Purdue University  
400 Centennial Mall Drive, West Lafayette, USA  
E-mail: muller@unitime.org

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

model, besides the usual classes, instructors and rooms tied together by various constraints (e.g., a room must be of large enough size, or an instructor can only teach one class at a time), there is a set of curricula defined. Each curriculum contains a list of courses that are to be attended by the same students (students of the curriculum). This is usually backed up by a hard constraint ensuring that classes of the same curriculum cannot overlap in time.

In the real world (McCollum, 2007), on the other hand, students are usually not required to attend all courses of a curriculum. Besides compulsory courses (courses that students must, or at least are expected, to take), there are elective courses (usually forming *groups*, where students are expected to take  $n$  of  $m$  courses) and optional courses that students may or may not take. Moreover, for some courses, students may decide during which semester they will take them. Typically, compulsory and elective courses cannot overlap in time, except there may be some overlaps of elective courses that are of the same group. For instance, if students are to take one of the given three courses, these three courses can be timetabled during the same time. Optional courses are usually only required to be at times that are not blocked by some other compulsory or elective course of the same curriculum. It is important to note here that each course may be present in multiple curricula, and it may be required for some curricula and only optional for another.

The whole problem is usually made even more complicated by the fact that courses tend to have multiple *course sections* (Hertz, 1991; Rudová et al, 2011). Courses with many students are usually split into several seminar groups and/or lectures. Furthermore, a course can be offered in various *configurations* (e.g., a lecture only, a lecture and a lab), with multiple lectures and labs available and some restrictions on what combinations of lecture and lab students are allowed to take. There can also be some mapping between curricula and specific classes of a course (e.g., the first lecture of a course may be reserved for students of an engineering major), but often there is none. Even simple course sectioning into several seminars leads to the inability to map curricula onto pairs of classes with no overlap in time. This is a very important aspect of the problem which must lead to more complex models and solutions.

## 1.1 Our Work

In our work, we rely on the UniTime<sup>1</sup> university timetabling system, containing an enrollment-based course timetabling solver which already deals with course sections and configurations of courses. In UniTime, students are assigned to classes based on student course demands (e.g., taken from pre-enrollment or from a previous semester) in a way that tries to keep students with similar courses together (Müller and Murray, 2010). A *class* is understood to be a part of the course which needs a time and a room assignment (e.g., each of the seminar groups or a lecture is a class). The course timetabling

---

<sup>1</sup> <http://www.unitime.org>

process looks for a proper assignment of times and classrooms to classes with the goal to optimize a set of criteria. It aims to minimize the number of student conflicts, i.e., cases when students are not able to attend classes due to their overlap in time or due to their placement of one right after the other in rooms that are too distant. Student conflicts can be decreased by assignment of a proper time as well as by swapping students between alternative classes or configurations of a course. Other important criteria include time and room preferences and restrictions for assignment of particular classes as well as distribution preferences specifying relations among several classes. A detail description of UniTime features and algorithms as well as its application to the Purdue University timetabling problem can be found at (Rudová et al, 2011).

This paper introduces a curriculum model which is applicable to solve real-life curriculum-based timetabling problems at large universities such as Masaryk University (Czech Republic) or Purdue University (USA). We also propose a transformation of this curriculum model into an enrollment model where each course is associated with a set of enrolled students (Lewis et al, 2007). A hybrid model of combining curricula with historic student enrollment data is also possible and briefly discussed in this paper. We describe a local search algorithm (Hoos and Stützle, 2005) which allows us to generate student course demands for the enrollment timetabling problem which respect characteristics of curricula. Given the curriculum model, transformation into the enrollment model, and the algorithm for generating student enrollments, it is possible to enable curriculum-based timetabling in *any* existing enrollment-based timetabling solver.

The proposed curriculum model and algorithm is implemented in UniTime and an application of this approach is presented on real-life curriculum-based timetabling problems from the Faculty of Education at Masaryk University. Timetables generated by UniTime have been used at the college in practice since Fall 2011. Here we have about 260 different curricula for about 7,500 students. This corresponds to timetabling of present, combined and lifelong forms of study. Overall computational results are presented on problems from two semesters, Fall 2011 and Spring 2012. We present results of the curricula to enrollments transformation for each of the three different forms of study, each representing a curriculum model with different characteristics. The number of curricula is very high since most of the programs in the college combine two different majors in order to educate teachers in two different subject areas (e.g., Math and Physics or Physics and Chemistry). A curriculum is defined for each allowed combination. This makes the overall timetabling very complex since there is a wide range of combinations with largely varying numbers of students in them. In particular, there are many combinations such as Math and Music, with only a few students, whose curriculum must still be respected. The initial requirement actually was to create timetables for compulsory and elective courses of all curricula with almost no student conflicts. In the paper, we demonstrate that timetabling of about 1,500 classes was possible with only about 100 student conflicts, i.e., 99.8% of student course demands in curricula were satisfied for compulsory and elective courses. This was accom-



panied by consideration of a set of other problem characteristics including time and room preferences on individual classes as well as various relations among classes. During the timetabling process, the solution generated by fully automated methods was also interactively adjusted to reflect additional needs of the faculty and the students. The main results for the initial automated timetables, as well as for the final interactively corrected timetables that were used at the Faculty of Education in practice, are presented for both semesters.

In the following section we propose a curriculum model and the next chapter specifies a possible mapping between curriculum and enrollment models as a transformation of curricula to student course demands. Consequently we describe a local search algorithm that computes student course demands. Finally we provide experimental results from the application of the resultant system at the Faculty of Education.

## 2 Curriculum Model

We propose a curriculum model that is able to tackle an advanced set of real-life characteristics of timetabling problems. A curriculum, usually tied to students by their academic area (program of study) and one or more majors (further specializations) is split between different semesters. The tuple specifying curriculum and semester is called *classification*. A number of students is associated with each classification. This number may be known or may be estimated from previous semesters using various student projections. In addition to the number of students, each curriculum has an associated set of courses defined for each semester. Each course has a percentage which evaluates to the number of students that are expected to attend the course out of all those in the classification. These course percentages may also be replaced by the number of students from the classification expected to attend the course.

To model the relations between courses in a curriculum, various groups are defined. Each group contains a subset of courses in the curriculum. It is possible to create two types of groups. A *conflicting group* expresses that students are expected to take all courses in the group. A *non-conflicting group* indicates that students are expected to attend just one course in the group. During timetabling this means that courses in a non-conflicting group may overlap in time. Courses in a conflicting group must be timetabled so that all students in one course are able to attend all other courses in the conflicting group.

An illustrative example of a curriculum is presented in Figure 1. A bachelors degree in the curriculum is offered over three years. Students are required to take courses A and B during their first year of study, C and D during their second year of study, and E and F during their last year of study. They are also expected to take course G either the first or the second year (though 80% of students usually take the course during the second year). Similarly, they can take course H during their second or third year. During the first

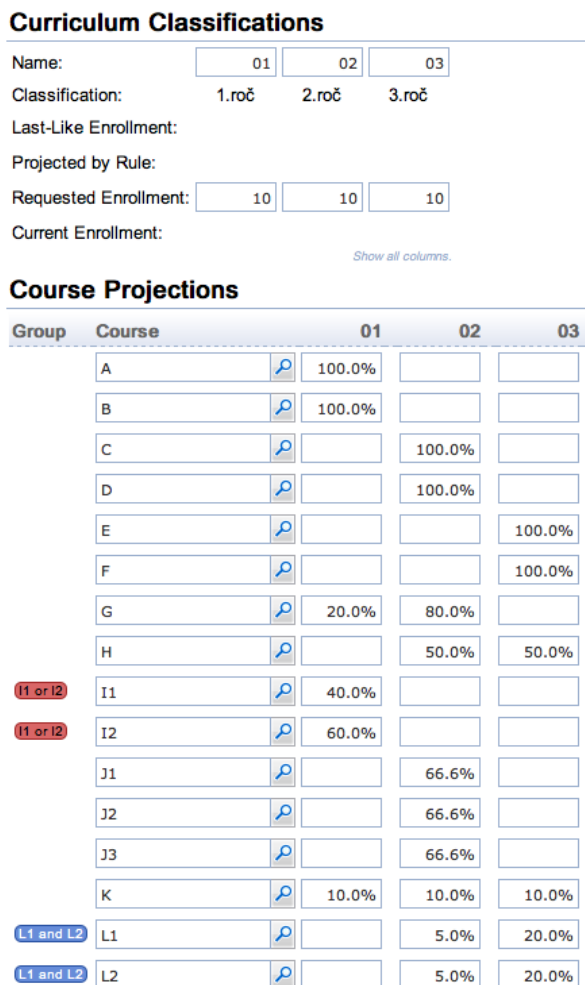


Fig. 1 Example of a curriculum prepared in UniTime timetabling system.

year, they should also take I1 or I2 (note that these two courses are put into a non-conflicting group “I1 or I2”). During their second year, they should also take two of courses J1, J2, J3 (this is solely modeled by the curriculum course percentages). There are also optional courses L1 and L2, which are either not taken at all or are taken together (this is modeled by the conflicting group “L1 and L2”).

It is also possible for a course to be in multiple groups, as demonstrated in Figure 2. Due to the transitive closure relationship between groups discussed in Section 2.1, this allows modeling cases where a student needs to take a certain pair of courses (M1 together with M2 or N1 together with N2 or O1 together with O2) or other more complex cases.

### Course Projections


Group	Course	01	02	03
M1 and M2 <span style="background-color: #FFD700;">N or M or O</span>	M1 	<input type="text"/>	50.0%	<input type="text"/>
M1 and M2	M2 	<input type="text"/>	50.0%	<input type="text"/>
N1 and N2 <span style="background-color: #FFD700;">N or M or O</span>	N1 	<input type="text"/>	30.0%	<input type="text"/>
N1 and N2	N2 	<input type="text"/>	30.0%	<input type="text"/>
O1 and O2 <span style="background-color: #FFD700;">N or M or O</span>	O1 	<input type="text"/>	20.0%	<input type="text"/>
O1 and O2	O2 	<input type="text"/>	20.0%	<input type="text"/>


Fig. 2 Example with courses in multiple groups.

Please note that a curriculum may not contain all courses that a student in the curriculum may take during his/her study, but only courses that are offered in the term that is to be timetabled. For instance, if we are creating the Spring 2012 timetable, only courses that are offered in Spring 2012 will be present, but there will still be students associated with a curriculum that are in different years or semesters of their study. Also, if there has been a curriculum change starting in Spring 2011, students in their third year will take courses from the old curriculum whereas students in their first and the second years will need to follow the new curriculum.

A typical curriculum for the combined form of masters study taking two years is shown in Figure 3. In the *Course Projections* table, the columns titled *01* and *02* contain the expected enrollments for each course and semester. The columns titled *Last* are optionally displayed columns which indicate the number of students enrolled in the course during the last-like semester (Fall 2010). Courses in the conflicting groups P 01 and P 02 represent compulsory courses in the first and the second year respectively. Courses in the non-conflicting groups PV 01, PVJ 01, PV 02 represent elective courses. Students in this curriculum are expected to take two electives in the first year (one from each group PV 01 and PVJ 01) and only one elective in the second year (group PV 02). Optional courses do not belong to any group (column *Group* in the *Course Projections* table is empty). Note that the groups P 01 and P 02 are not necessary as all the courses in these two groups expect attendance by all the first year or second year students respectively.

#### 2.1 Formal Model

More formally, there is a set of courses  $c \in C^a$  for each curriculum  $a \in A$ . The set of all courses in the whole timetabling problem corresponds to  $\bigcup_{a \in A} C^a$ . Since a course may appear in more than one curriculum, sets  $C^a$  and  $C^b$  for curricula  $a, b \in C$  may possibly have a non-empty intersection. A curriculum  $a$  is associated with student counters  $x_1^a, x_2^a, \dots, x_n^a$  where  $n$  is the number of semesters and  $x_i^a$  is the number of students in curriculum  $a$  and semester  $i$ .



## Curriculum Detail <sup>2</sup>

Müller, Tomáš      Podzim 2011 (PDF)  
Administrator      [Click here to change the session / role.](#)

---

### Curriculum Details

[Edit](#)   [Delete](#)   [Print](#)   [Back](#)

---

Abbreviation: KN-PD/SOCP  
 Name: Pedagogika / Soc. ped.  
 Academic Area: Pedagogika  
 Major(s): SOCP - Sociální pedagogika  
 Department: KSocPed - Katedra sociální pedagogiky

### Curriculum Classifications

---

Name:	01	02	03	04	OST
Classification:	1.roč	2.roč	3.roč	4.roč	ostatní
Last-Like Enrollment:	58	65	1	1	9
Requested Enrollment:	59	43			

### Course Projections

---

Group Course	01 Last	02 Last		
<span style="color: red;">P 01</span> SO SC4MK_ET	59	57		
<span style="color: red;">P 01</span> SO SC4MK_PAT2	59	57	3	
<span style="color: red;">P 01</span> SO SC4MK_PPD	59	57	2	
<span style="color: red;">P 01</span> SO SC4MK_SGI	59	57	3	
<span style="color: red;">P 01</span> SO SC4MK_TEOS	59	57		
<span style="color: yellow;">PVJ 01</span> CJ SC4MK_CJAJ	41	20	13	
<span style="color: blue;">PV 01</span> SO SC4MK_PRVC	19	21	5	
<span style="color: yellow;">PVJ 01</span> CJ SC4MK_CJNJ	18	10	5	
<span style="color: blue;">PV 01</span> SO SC4MK_KAZS	18	18	23	
<span style="color: blue;">PV 01</span> SO SC4MK_TVMV	18	19	1	
<span style="color: blue;">PV 01</span> SO SC4MK_REEV	5	5		
SO CJV_ROM1	1	1	1	1
<span style="color: green;">P 02</span> SO SC4MK_ANDR			43	56
<span style="color: green;">P 02</span> SO SC4MK_APPR			43	56
<span style="color: green;">P 02</span> SO SC4MK_KRIN			43	
<span style="color: green;">P 02</span> SO SC4MK_MAN2			43	56
<span style="color: green;">P 02</span> SO SC4MK_PsVL			43	55
<span style="color: green;">P 02</span> SO SC4MK_SPR2			43	56
<span style="color: orange;">PV 02</span> SO SC4MK_MAPX			14	
<span style="color: orange;">PV 02</span> SO SC4MK_INMK			12	14
SO SZ9BP_ZRM1	3	3	11	
SO SZ9BP_DUTL	1	1	11	
SO CJV_ROM2			11	
<span style="color: orange;">PV 02</span> SO SC4MK_MARK			10	12
<span style="color: orange;">PV 02</span> SO SC4MK_ENEV			6	7

Fig. 3 Example of curriculum from Fall 2011 at the Faculty of Education.

Note that the tuple  $(a, i)$  defines a classification. To simplify formulations, we define the set  $AI$  as a set of all classifications  $(a, i)$ .

Furthermore, there is a matrix with values  $e_{c,i}^a$  between 0 and 1. For each course  $c$  and semester  $i$ , the value  $e_{c,i}^a$  defines the proportion of the number of students  $x_i^a$  expected to attend the course  $c$ . Each course  $c \in C^a$  is expected to be attended by  $e_{c,i}^a x_i^a$  students from classification  $(a, i)$ . Finally, there are groups of courses  $G_1^a \subseteq C^a, \dots, G_k^a \subseteq C^a$  representing conflicting groups and  $H_1^a \subseteq C^a, \dots, H_l^a \subseteq C^a$  representing non-conflicting groups in the curriculum.

It is also important to mention that for each pair of courses  $c, d \in C^a$ , we expect the following proportion of the number of students in classification  $(a, i)$  represented by values from interval  $\langle 0, 1 \rangle$ .

$$t_{c,d,i}^a = \begin{cases} 0 & \exists j : c, d \in H_j^a, \\ 1 & \exists j : c, d \in G_j^a, \\ e_{c,i}^a e_{d,i}^a & \text{otherwise.} \end{cases}$$

We call this number the *target share* of a curriculum between the two courses.

For the above example with Figure 1, the target share between courses L1 and L2 is 1 (all students attending L1 are expected to attend L2 and vice versa), it is 0 between courses I1 and I2 (students are taking either I1 or I2, but not both), and it is 0.44 between J1 and J2 for the second year students (44% of students are expected to take both J1 and J2).

If there are courses in multiple groups, a special graph needs to be considered. Here the nodes are represented by courses and the edges are defined by the existence of a (conflicting or non-conflicting) group between the two courses. More precisely, there is an edge between  $c, d \in C^a$  if courses  $c$  and  $d$  are present in the same group. The target share  $t_{c,d,i}^a$  is set to zero if there is a path  $c = c_1, c_2, \dots, c_{m-1}, c_m = d$  in the graph where at least one of the groups defining edges on the path is non-conflicting. The target share  $t_{c,d,i}^a$  is set to one if the path has all the groups conflicting. Note that correct computation of all target share values necessitates computation of the transitive closure in the graph. For the above example with Figure 2, this means that we also expect no students to be between course M1 and N2, between M2 and N1, and between M2 and N2 (and similar for O1 and O2 courses).

### 3 Curriculum to Enrollment Model Transformation

We now show that the proposed curriculum model can be transformed into an enrollment model. In the enrollment model, each course has a set of students enrolled in it. Our goal is to find an assignment of students to courses in the enrollment model such that the curricula are respected. This means that student conflicts in the enrollment model should correspond with the number of broken requests given by curricula. For instance, two compulsory courses in a curriculum with 10 students timetabled at the same time corresponds to 10 student conflicts in the enrollment model.

First, consider the target share between two courses in a curriculum. We propose relating the target share with the number of students who must be able to attend both courses. This corresponds with the number of student conflicts in the enrollment model. Further, particular target shares specify the characteristics of an *ideal* enrollment model in which student conflicts fully correspond with the unmet curricular course requirements in the curriculum model. Having this in mind, we define an optimization criterion evaluating assignments of students to courses in the enrollment model. This criterion evaluates the affinity of this assignment to an ideal enrollment model. Generally, we can summarize distances from optimality based on a comparison of characteristics (target shares) of the ideal enrollment model with characteristics of the assignment. Certainly our goal is to find an assignment with the minimal distance. For instance, if there are two courses with target share of 15 students in a curriculum  $a$  and our assignment has only 14 students that are enrolled in both of these courses from  $a$ , then the contribution to the distance corresponds to 1.

### 3.1 Formal Transformation

In the curriculum model, there are  $x_i^a$  students for each classification  $(a, i)$ . We summarize characteristics of the ideal enrollment model with respect to the curriculum model.

1. There are  $e_{c,i}^a x_i^a$  students from the classification  $(a, i)$  enrolled in the course  $c$ .
2. The target share  $t_{c,d,i}^a$  between courses  $c$  and  $d$  of the classification  $(a, i)$  specifies the number of students  $t_{c,d,i}^a x_i^a$  enrolled in both courses  $c$  and  $d$ .

If we expect that each student is enrolled in only one curriculum<sup>2</sup>, all curricula are independent and do not share any students. Also, it is clear that each student is enrolled in only one semester of the curriculum  $a$ . This means that there are different students in the two semesters  $i, j$  of curriculum  $a$ . This extends characteristics of the ideal enrollment model with respect to the curriculum model.

1. There is a total of  $\sum_{(a,i) \in AI} e_{c,i}^a x_i^a$  students enrolled in the course  $c$ .
2. For each two courses  $c, d \in C^a$ , the total number of students enrolled in both courses corresponds to  $\sum_{(a,i) \in AI} t_{c,d,i}^a x_i^a$ .

Next we consider an assignment  $\theta$  of students to courses defining an enrollment model which will be evaluated with respect to the ideal enrollment model. We also expect that each such student belongs to a curriculum  $a \in A$  and its semester  $i$  and the number of students in curriculum  $a$  equals to  $x_i^a$  for the semester  $i$ . In the assignment  $\theta$ , we denote the number of students  $x_i^a$  in classification  $(a, i)$  belonging to both courses  $c, d \in C^a$  as an *actual share*  $s_{c,d,i}^a$ .

---

<sup>2</sup> This is typical for the vast majority of students in our case. Only students wanting to study two different topics would not satisfy this condition.

The assignment  $\theta$  is evaluated by the *distance*

$$F(\theta) = \sum_{(a,i) \in AI} \sum_{c,d \in C, c \neq d} |t_{c,d,i}^a x_i^a - s_{c,d,i}^a| = \sum_{(a,i) \in AI} F(\theta, a, i) .$$

Clearly an ideal assignment  $\omega$  (assignment with the ideal enrollment model) has a distance  $F(\omega) = 0$ . Since such an assignment may not necessarily exist, our goal is to find an assignment  $\sigma$  with the minimal distance  $F(\sigma)$ . We also defined  $F(\theta, a, i)$  since students in each classification  $(a, i)$  are different from students in all other classifications. Following that, the assignment of students for each classification is independent of assignments in all other classifications and the following statement holds.

$$\min F(\theta) = \sum_{(a,i) \in AI} \min F(\theta, a, i) \quad (1)$$

### 3.2 Using Historical Data

It is also possible to make adjustments to the target share matrix based on historical data, typically from the last-like semester (e.g., last-like semester for Spring 2012 is Spring 2011). For instance, if we know from past experience that students attending J1 are more likely to attend J2 than J3, we can use this information to adjust the values of the matrix to reflect this.

More formally, the target share is  $t_{c,d,i}^a = p r_{c,d,i}^a + (1-p) e_{c,i}^a e_{d,i}^a$  where  $r_{c,d,i}^a$  is the percentage of students from classification  $(a, i)$  that took both courses  $c$  and  $d$  in the last-like semester, and  $p$  a number between 0 and 1 defining how much we want to stick with the past. Note that this new target share only applies to pairs of courses for which we have historical data. In other words, if either course  $c$  or  $d$  is newly offered, the target share is only defined by  $e_{c,i}^a e_{d,i}^a$  matrices and the groups in the previous chapter.

Given this, we can take assignment  $\theta$  of students in the last-like semester, compute its distance  $F(\theta)$  and consider it as an initial assignment when looking for an (sub-)optimal solution to the timetabling problem using the curriculum model.

## 4 Construction of Enrollments

We specified how students should be assigned to courses to respect the curriculum model by minimization of the distance  $F(\theta)$ . In this section, we describe a local search algorithm which allows computation of a “reasonable” sub-optimal assignment  $\theta$  of students to courses with respect to  $F(\theta)$ . As discussed in the second paragraph of Section 3.2 and demonstrated in Equation 1, we expect different students for each classification  $(a, i)$ . This means that the assignment  $\theta$  can be computed per partes, i.e., the local search algorithm is applied to each classification separately.

First, it is important to discuss computation of the target share between two courses  $c, d \in C^a$ . We will concentrate on computation of the number of students  $\bar{t}_{c,d,i}^a$  that are expected to be assigned to both courses  $c$  and  $d$ . Corresponding to Section 2.1, this is counted based on  $t_{c,d,i}^a x_i^a$ . However, it is also bounded by the number of students in courses  $c$  and  $d$ , respectively. To account for this, the number of students in the course  $c$  for classification  $(a, i)$  is denoted  $x_{c,i}^a = \text{round}(e_{c,i}^a x_i^a)$ . For instance, if there are 20 students in the given semester of a curriculum, and courses  $c$  and  $d$  are expected to be attended by 10 and 15 students respectively, the share of the two courses must be between 5 and 10.

$$\bar{t}_{c,d,i}^a = \begin{cases} \max(0, x_{c,i}^a + x_{d,i}^a - x_i^a) & \exists j : c, d \in H_j^a, \\ \min(x_{c,i}^a, x_{d,i}^a) & \exists j : c, d \in G_j^a, \\ \max\left(\min\left(\text{round}(e_{c,i}^a e_{d,i}^a x_i^a), x_{c,i}^a, x_{d,i}^a\right), x_{c,i}^a + x_{d,i}^a - x_i^a\right) & \text{otherwise.} \end{cases}$$

The overall distance  $F(\theta, a, i)$  for classification  $(a, i)$  is counted incrementally for each course as the difference  $\Delta F(\theta, a, i, c, z_{\text{new}}, \perp)$  between the distance before and after assignment of a student  $z_{\text{new}}$  into a course  $c$ . It also allows for a swap of a course between two students  $z_{\text{new}}$  and  $z_{\text{old}}$  (denoted by  $\Delta F(\theta, a, i, c, z_{\text{new}}, z_{\text{old}})$ ). Pseudo-code of this function is presented in Figure 4.

```

1: function  $\Delta F(\theta, a, i, c, z_{\text{new}}, z_{\text{old}})$ 
2:  $f = 0$ 
3: for  $d \in C^a$  such that  $d \neq c$ 
   (for each course other than  $c$ ,  $f$  is increased by the difference
   between target share and actual share before and after the change)
4:    $t := \bar{t}_{c,d,i}^a$ 
5:    $s := s_{c,d,i}^a$ 
6:    $f := f - |t - s|$ 
7:   if  $z_{\text{new}} \in \text{students}(d)$  then  $s := s + 1$ 
8:   if  $(z_{\text{old}} \neq \perp$  and  $z_{\text{old}} \in \text{students}(d))$  then  $s := s - 1$ 
9:    $f := f + |t - s|$ 
10: return  $f$ 

```

**Fig. 4** Pseudo-code of function  $\Delta F(\theta, a, i, c, z_{\text{new}}, z_{\text{old}})$ .

The search for assignment of students to courses for the given classification  $(a, i)$  is processed in two phases as can be seen in Figure 5. In the first phase (lines 3-7) a simple *construction* heuristic is used. In each iteration, a single student is assigned to a particular course until each course has the desired number of students. Courses are ordered dynamically by the number of remaining spaces (if there are two or more courses with the same number of remaining spaces, one is selected randomly — see line 4). For a selected course, all students that are not yet assigned to it are checked, and one of the students that has the best impact on the overall distance is selected randomly (line 5).

In the second phase (lines 8-18) a *great deluge* approach (Dueck, 1993) is used. The initial bound  $UB$  is set to 1.25 of the initial solution's distance  $f$



```

1: procedure SEARCH( $a, i, \alpha$ )
2:  $f := 0$ 
   (construction phase)
3: while  $\exists u \in C^a$  such that  $\|students(u)\| < x_{c,i}^a$ 
4:    $c := \text{RANDOM}(d \in C^a \text{ such that } \text{MAXIMAL}(x_{d,i}^a - \|students(d)\|))$ 
5:    $z := \text{RANDOM}(v \notin students(c) \text{ such that } \text{MINIMAL}(\Delta F(\theta, a, i, c, v, \perp)))$ 
6:    $f := f + \Delta F(\theta, a, i, c, z, \perp)$ 
7:    $students(c) := students(c) \cup \{z\}$ 
   (great deluge phase)
8:    $UB = 1.25 * f$  (upper bound)
9:   while ( $UB \geq 0.75 * f$  and  $f > 0$ )
10:     $c := \text{RANDOM}(d \in C^a \text{ such that } \|students(d)\| < x_{d,i}^a \text{ and } x_{d,i}^a > 0)$ 
11:     $z_{old} := \text{RANDOM}(v \in students(c))$ 
12:     $z_{new} := \text{RANDOM}(v \notin students(c))$ 
13:     $\Delta f := \Delta F(\theta, a, i, c, z_{new}, z_{old})$ 
14:    if ( $\Delta f \leq 0$  or  $f + \Delta f \leq UB$ ) then
15:       $f := f + \Delta f$ 
16:       $students(c) := students(c) \setminus \{z_{old}\}$ 
17:       $students(c) := students(c) \cup \{z_{new}\}$ 
18:       $UB := UB * (1 - \alpha)$ 

```

**Fig. 5** Pseudo-code of the algorithm computing enrollments for classification  $(a, i)$ .

(line 8) and it is decreased by the coefficient  $\alpha$  (typically 0.0001 %) in each iteration (line 18). The search is stopped when a solution with zero distance is found or when the bound reaches 0.75 of the solution's distance (line 9). In each iteration, a possible change of a single student in a course is generated (lines 10-12) and accepted if the resultant solution's distance does not exceed the bound (lines 14-17). Changes that do not increase the distance are also accepted. A course, a student that is removed from this course and a student that is assigned to this course are selected randomly (lines 10, 11, 12, respectively).

When an assignment  $\theta$  with zero distance  $F(\theta, a, i)$  is found during the first phase, the second phase is not executed. When historical data are available and we want to take them into account, the first phase starts with last-like semester enrollments (see Section 3.2).

## 5 Experimental Results

The following experiments are based on Fall 2011 and Spring 2012 data from the Faculty of Education at Masaryk University. Fall 2011 is the first semester for which UniTime was used to build the course timetable for the college. Experiments in Section 5.1 were computed on an Apple MacBook Pro with a 3.06 GHz Intel Core 2 Duo processor and 8 GB RAM, running Mac OS X 10.7.3 and Java 1.6.0. Results in Section 5.2 are presented from the installation for the Faculty of Education and the Faculty of Arts (Rudová and Müller, 2011) running on a virtual machine hosted on a machine with two Intel X5560 processors and 96 GB RAM (8 GB dedicated to the UniTime virtual machine).

### 5.1 Curriculum to Enrollment Transformation

Table 1 shows results from the curriculum to enrollment solver. All of the curricula are split into three groups based on the student's form of study. Results are presented for all curricula together as well as for particular sets. For each set, the number of curricula, classifications and students is specified. We also present the number of students per classification and the number of courses per classification. For these data sets, the average distance  $F(\theta, a, i)$ , the average distance achieved after the construction phase of the search algorithm (see Figure 5), and the average computational time is presented for 10 independent runs.

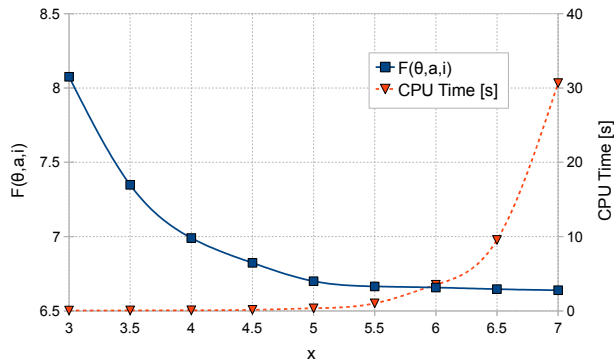
We can see that curricula of the present form of study introduce the largest data set with the highest computed distance. Curricula of the combined form of study can be transformed into a better enrollment model (the distance is lower) and it takes a longer time. Curricula of the lifelong form of study introduce the smallest data set and are easiest to transform. To understand the achieved quality of the solution we note that the distances 7.05 and 6.66 for the present form of study correspond to 0.60% and 0.69% of the worst possible distance, respectively.

Certainly we tried to find the best possible distance  $F(\theta, a, i)$  in a reasonable time. We ran the set of experiments with varying size of  $\alpha$  (see line 18 of Figure 5 and description of the algorithm) influencing progress of the search

Fall 2011	All together	Present (P)	Combined (K)	Lifelong (C)
Spring 2012				
Curricula	265	210	28	27
	258	202	25	31
Classifications	574	470	56	56
	543	442	53	48
Students	7,569	4,301	2,562	706
	6,803	3,852	2,362	589
Students	13.19	9.15	45.75	14.71
per classif.	12.53	8.71	44.57	12.27
Courses	30.61	34.63	18.32	5.67
per classif.	27.44	31.06	15.62	7.21
$F(\theta, a, i)$	$7.05 \pm 0.01$	$8.24 \pm 0.01$	$3.14 \pm 0.03$	$0.00 \pm 0.00$
	$6.66 \pm 0.01$	$8.04 \pm 0.01$	$1.02 \pm 0.03$	$0.13 \pm 0.00$
$F(\theta, a, i)$	$11.97 \pm 0.13$	$13.25 \pm 0.13$	$11.53 \pm 0.14$	$0.04 \pm 0.06$
after 1. phase	$10.87 \pm 0.11$	$11.99 \pm 0.12$	$11.03 \pm 0.12$	$0.31 \pm 0.06$
CPU time [s]	$3.36 \pm 0.06$	$3.08 \pm 0.05$	$8.58 \pm 0.12$	$0.01 \pm 0.00$
	$3.53 \pm 0.07$	$2.88 \pm 0.06$	$12.14 \pm 0.16$	$0.02 \pm 0.03$

**Table 1** Computing enrollments for two semesters.

algorithm. The smaller  $\alpha$  is the slower the upper bound decreases and the great deluge algorithm has more time for optimization. Results of this experiment are available in Figure 6. Here we can see that the algorithm is able to



**Fig. 6** Graph depicting dependency of the distance  $F(\theta, a, i)$  and the computational time on the value  $\alpha = 10^{-x}$  for the problem with all curricula and semester Spring 2012.

improve the distance with a reasonable demands on computational time up to the value  $\alpha = 0.0001\%$  (corresponding to  $x = 6$  in the graph). Decreasing this value further does not achieve significant improvements in the distance in a reasonable computational time.

## 5.2 Course Timetabling

Table 2 contains results from the course timetabling (courses from the present form of study only) at the Faculty of Education. For each semester, there are results from the automated solver as well as the published solution, after a few interactive changes were made. Note that published solutions were used in practice. The table shows the overall number of courses and the number of compulsory and elective courses (in parenthesis). Similarly, the number of classes and the number of student enrollments in each problem are presented. The second part of the table presents the main characteristics of the computed solution. The most important factor of the problem was the number of student conflicts among compulsory and elective courses. As we can see, we have only 112 and 96 conflicts for 1,575 and 1,408 timetabled classes, respectively. This is certainly a very strong result given all of the complexities and the size of both problems (recall from Table 1 that we have 210 and 202 curricula for 4,301 and 3,852 students, respectively). The number of student conflicts among all courses is slightly higher, it is mostly due to overlaps between optional and compulsory or elective courses. These numbers, together with results for time, room, and distribution preferences, correspond with priorities of the school and the importance of particular criteria. The better results for Spring semester were achieved due to a smaller number of classes timetabled into the same

	Fall 2011 automated	Fall 2011 published	Spring 2012 automated	Spring 2012 published
Courses (comp. & elect.)	1,225 (1,156)		900 (870)	
Classes (comp. & elect.)	1,831 (1,575)		1,665 (1,408)	
Enrollments (comp. & elect.)	57,861 (52,396)		45,786 (45,400)	
Student conflicts	418 (0.63 %)	456 (0.69 %)	477 (1.02 %)	417 (0.89 %)
among comp. & elect.	112 (0.17 %)	140 (0.21 %)	96 (0.20 %)	93 (0.20 %)
Time preferences	89.27 %	89.93 %	94.88 %	95.32 %
Room preferences	78.03 %	79.92 %	85.15 %	86.50 %
Distribution preferences	84.50 %	80.41 %	90.49 %	90.49 %
Interactive changes		355		275
of time		183		105
of room		300		218

**Table 2** Results from timetabling at the Faculty of Education for two semesters.

amount of available classrooms. Finally, the number of interactive changes with the initial solution is demonstrated. These low numbers show that it is not necessary to adjust solutions much. Still, it is important to allow some adjustments to the solutions to make them more acceptable for the school.

## 6 Conclusion

We presented a new approach to curriculum-based timetabling and applied it to solve large-scale problems at the Faculty of Education where it is implemented in the UniTime system and used for timetabling since Fall 2011. Automated timetabling simplified the process for the college where about 40 schedule deputies cooperated on creating timetables manually until Spring 2011. Presently, they provide only inputs, such as desirable assignments of times and rooms to classes, and the timetables are constructed by UniTime. It is also important to mention the existence of Information System<sup>3</sup> at Masaryk University where curriculum data are maintained and can be used for timetabling directly. Resultant timetables are also available here<sup>4</sup>.

Further generalization of the approach involves inclusion of students in more than one curriculum. This is fully compatible with the proposed search algorithm where students can be used in different curricula and additional courses will be generated for them from each curriculum. However, this approach is not yet implemented. Also, some reformulation of the curriculum model related to the proposed distance function is necessary. Our intent to include this functionality lies in easier management of existing curricula. Having many curricula composed of two different majors (examples are Math and

<sup>3</sup> <http://is.muni.cz>

<sup>4</sup> <http://is.muni.cz/rozvrh/?fakulta=1441&lang=en>

Physics or Physics and Chemistry as discussed before), it would be easier to maintain the smaller set of majors and an additional set of acceptable combinations of majors defining curricula.

**Acknowledgements** This work is supported by the Grant Agency of Czech Republic under the contract P202/12/0306. The access to the MetaCentrum computing facilities provided under the program "Projects of Large Infrastructure for Research, Development, and Innovations" LM2010005 funded by the Ministry of Education, Youth, and Sports of the Czech Republic is highly appreciated.

## References

- Bonutti A, De Cesco F, Di Gaspero L, Schaerf A (2012) Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research* To appear
- Burke EK, Petrovic S (2002) Recent research directions in automated timetabling. *European Journal of Operational Research* 140:266–280
- Di Gaspero L, McCollum B, Schaerf A (2007) The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Tech. Rep. QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, University, Belfast, United Kingdom
- Dueck G (1993) New optimization heuristics: The great deluge algorithm and the record-to record travel. *Journal of Computational Physics* 104:86–92
- Hertz A (1991) Tabu search for large scale timetabling problems. *European Journal of Operational Research* 54(1):39–47
- Hoos HH, Stützle T (2005) *Stochastic Local Search Foundations and Applications*. Elsevier
- Lewis R (2008) A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum* 30(1):167–190
- Lewis R, Paechter B, McCollum B (2007) Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. *Cardiff Working Papers in Accounting and Finance A2007-3*, Cardiff Business School, Cardiff University
- McCollum B (2007) A perspective on bridging the gap between theory and practice in university timetabling. In: Burke E, Rudová H (eds) *Practice and Theory of Automated Timetabling VI*, Springer-Verlag LNCS 3867, pp 3–23
- Müller T, Murray K (2010) Comprehensive approach to student sectioning. *Annals of Operations Research* 181:249–269
- Rudová H, Müller T (2011) Rapid development of university course timetables. In: *Proceedings of the 5th Multidisciplinary International Scheduling Conference – MISTA 2011*, pp 649–652
- Rudová H, Müller T, Murray K (2011) Complex university course timetabling. *Journal of Scheduling* 14(2):187–207

---

# Schedule Pattern: an Innovative Approach to Structuring Time in Secondary Schools Scheduling

Baiyun Tao, Rick Dwyer

*Follett Software Company/X2 Development Corporation, Hingham, MA, USA*

[btao@FollettSoftware.com](mailto:btao@FollettSoftware.com)

[rdwyer@FollettSoftware.com](mailto:rdwyer@FollettSoftware.com)

## 1 Introduction

This extended abstract describes the structure of time used by secondary schools in the United States, three common scheduling models each utilizing time differently, and an innovative approach using schedule patterns to efficiently and flexibly schedule classes into a master schedule. The content of this extended abstract is based on over 20 years of practical experience scheduling secondary schools in the United States. During that time the authors have built thousands of high school and middle school schedules using various versions of timetabling software we created.

The school year in a typical secondary school in the United States is comprised of 180 days. Each school day consists of about 5½ - 6 hours of instruction resulting in approximately 1000 hours of instructional time per student per school year. Partitioning instructional time between the various curriculum content areas (math, science, social studies, language, art, music, technology, etc.) is a key challenge. The structure of time in the school schedule provides the framework for how this partition is accomplished.

In section 2 we define three fundamental parameters that govern the structure of time in a secondary school schedule: terms per year (TPY), days per cycle (DPC), and periods per day (PPD). The value of these parameters determines the kind of schedules possible.

In section 3 we describe three common scheduling models and some of their variations. Each model makes specific choices regarding the values of TPY, DPC, and PPD which impact not only the student and teacher experience but the challenges involved in scheduling the school.

In section 4 we describe schedule patterns, an innovative approach to grouping time slots into valid schedule instances. By combining schedule patterns into pattern sets and assigning pattern sets to courses we can significantly reduce the work involved in scheduling each course.

In section 5 we describe schedule rotations. A schedule rotation remaps time slots from one schedule matrix (DPC x PPD) to another schedule matrix with the same dimensions or one with different dimensions. Rotations are applied post schedule build. They allow a user to schedule their school using simple, uniform patterns and then transform it into a schedule with more complex varied patterns.

## **2 Structure of time in secondary school schedules**

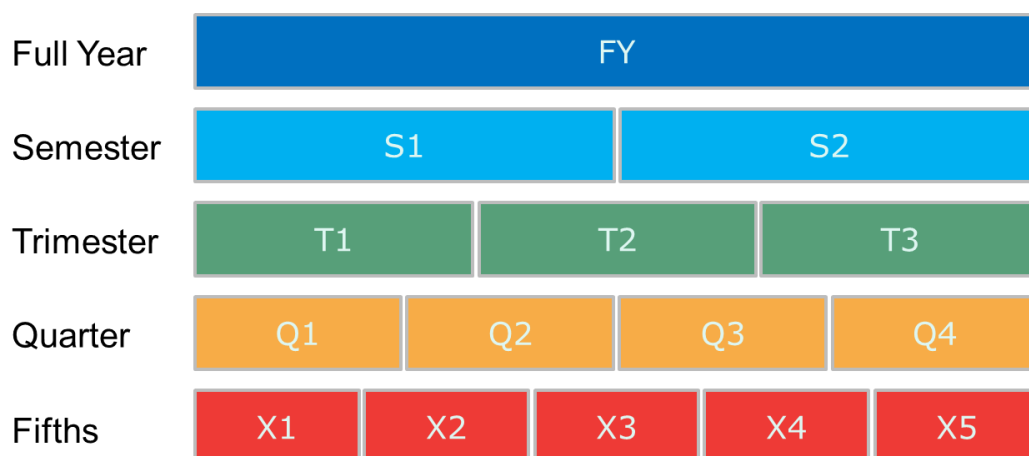
In everyday life time is organized around clocks and calendars. The calendar splits up the year into months, weeks, and days while the clock splits up the day into hours, minutes, and seconds. A week is a block of seven days that repeats throughout the year. These structures provide a universal way to talk about time and schedule our lives. In the same way secondary schools in the United States use analogous time structures to meet their scheduling needs.

### **2.1 School Year**

The school year consists of 180 school days. A school year typically starts in August or September and ends in May or June. The specific start and end dates depend on state requirements as well as the number and duration of school vacations and holidays.

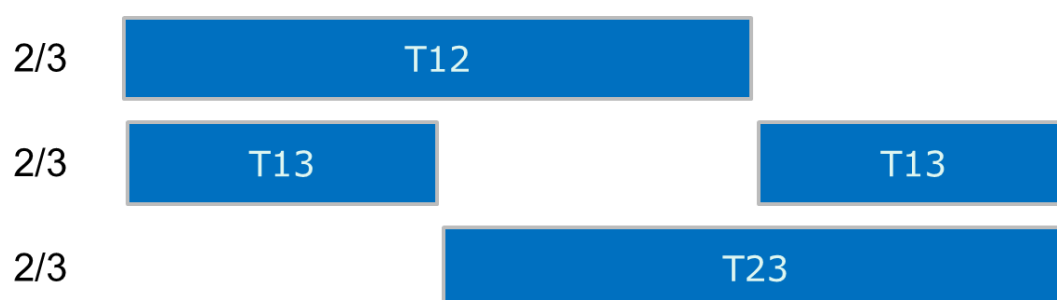
## 2.2 Schedule Term

A schedule term is a portion of a school year. Common schedule terms are full year (180 days), semester (90 days each), and quarter (45 days each) (see Fig.1.1). Schools can define as many different schedule terms as the schedule requires.



**Fig 1.1** Typical scheduling terms

Typically a course is scheduled over a single schedule term. But courses can be schedule over multiple schedule terms. For example an art course could be scheduled over two trimesters (120 days or  $2/3$  of the school year) (see Fig 1.2).



**Fig 1.2** Combine schedule terms consecutively and non-consecutively

## 2.3 Schedule Cycle

The schedule cycle is analogous to a week on a standard calendar. It is a fixed block of days that repeats throughout the school year. The school defines the number of days in the schedule cycle. Most schools do not use a 5 day cycle and



therefore the days in the schedule cycle do not correspond to the days of the week. An even number of days such as a 6 day cycle is more common and allows multiple sections of a partial cycle course to be scheduled in the same period on alternate days.

## **2.4 Day**

The schedule day is simply a day in session. Public schools in the United States are only in session during the week – they do not meet on Saturday or Sunday.

## **2.5 Period**

The school day is broken up into periods. Each day has the same number of periods and each period has the same duration. This uniformity allows courses to be scheduled during any period of the day. Typically a bell rings to mark the start and end of each period. Between each period students usually have 3-5 minutes get to their next class.

The structure of a schedule is therefore defined by three fundamental parameters: the number of term per year (TPY), the number of days per cycle (DPC) and the number of periods per day (PPD).

The number of days per cycle and the number of periods per day ( $DPC \times PPD$ ) defines a simple schedule matrix.

The shape of the schedule matrix ultimately has the biggest impact on the day-to-day experience of students and teachers in a schedule. It governs the amount of time a student spends in each course during a day and the variety of courses encountered during the cycle.

## 3 Three Common Scheduling Models

In this section we look at three common scheduling models: traditional, block, and hybrid.

### 3.1 Traditional Schedule

The most common traditional secondary school schedule has 7 periods per day and 6 days per cycle. Traditional schedules usually vary between 6 – 8 periods per day – the more periods the shorter the duration of each period.

Academic classes (English, math, science, and social studies) in a traditional schedule usually run for a single period every day for the full school year. Non-academic classes (art, music, or physical education) often receive less instructional time. Changing the instructional time a course is schedule is accomplished by reducing the number of days in the cycle or the number of terms per year the course is scheduled. Partial cycle or partial year courses are common in a traditional schedule. For example, physical education could be scheduled every other day rather than every day and may only run a semester rather than a full year. Typically a traditional schedule includes both semester and quarterly courses.

In a traditional (7 x 6) schedule, students take 7 or more classes in the same term. Teachers teach for 5 periods per day and usually have at least one period free for planning the other period could be scheduled as a duty.

### 3.2 Block Schedule

The most common block schedule has 4 periods per day and 1 day per cycle. This is called a 4x4 block schedule [1,2]. Each course meets 90 minutes per day for a semester.

A popular variation on the 4x4 block schedule is the A/B block schedule. The A/B block schedule also has 4 periods per day but each course meets every other day for the entire school year rather than every day for a semester.

With a smaller number of periods per day the block schedule allows extended learning time each day in each class. Students and teachers also have fewer classes meeting on each day.

### **3.3 Hybrid Schedule**

A hybrid schedule is one which combines the traditional schedule and the block schedule. A common hybrid schedule has 8 periods per day and 2 days per cycle. Courses in a hybrid schedule are usually scheduled in 1 period (traditional) or in 2 consecutive periods (block).

## **4 Schedule Patterns**

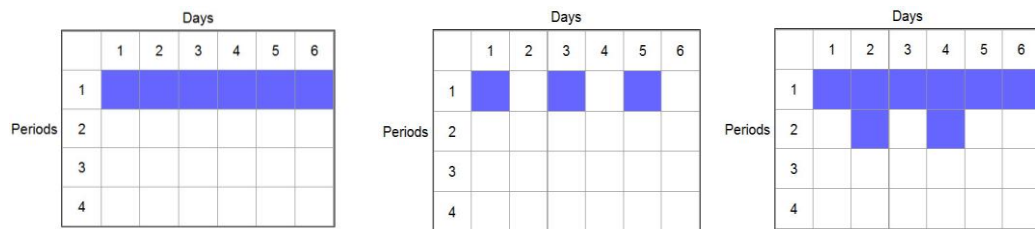
The master schedule builder schedules sections of courses. A course defines the curriculum content while a section defines an instance of that content in the schedule. For example a high school schedule will typically include a geometry course. If 100 students request geometry, multiple sections of geometry will be scheduled. A section represents a specific instance of the course scheduled at a particular time, in a particular room, by a particular teacher.

Schedule patterns represent the valid ways a particular course such as geometry can be scheduled and provide a visual representation of the shape of a course in the schedule. The following are common schedule patterns in a traditional 6 day schedule (1 period every day, 2 periods every day, 1 period every other day, and 1 period on 4 days and 2 periods on 2 days).

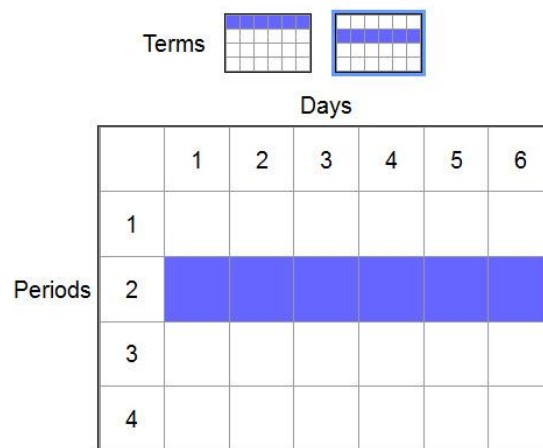
The defining attributes of a schedule pattern are the following:

- Days: number of days scheduled in the cycle
- Periods: number of periods scheduled during the day

- Terms: portion of the school year the pattern covers. By default a schedule pattern is term agnostic so that it can be used in conjunction with any schedule term to represent different time slots cross the school year. (see Fig.2.1) It can also be term aware and specifies the particular terms covered (see Fig.2.2).
- Shape: the distribution pattern of time slots (see Fig.2.1).



**Fig 2.1** Term-agnostic schedule patterns: every day, every other day and lab



**Fig 2.2** Term-aware schedule pattern: period 1 on S1 Period 2 on S2 (focus on S2)

Schedule patterns are grouped together as pattern sets. Each course in the schedule is assigned a pattern set containing the valid schedule patterns for that course. Courses scheduled in the same way typically share the same pattern set. However a schedule patterns can be used by multiple pattern sets. So a course that should only be scheduled in the morning could be associated with a pattern set containing only morning schedule patterns.

Schedule patterns allow the system to efficiently and effectively schedule course sections into valid time slots. For example, there are 20 possible day combinations

for a partial cycle course scheduled on 3 days in a 6 day cycle (in a single period). In reality, however, only 2 of the 20 possible day combinations are typically valid – the odd and even day combinations (1,3,5) and (2,4,6).

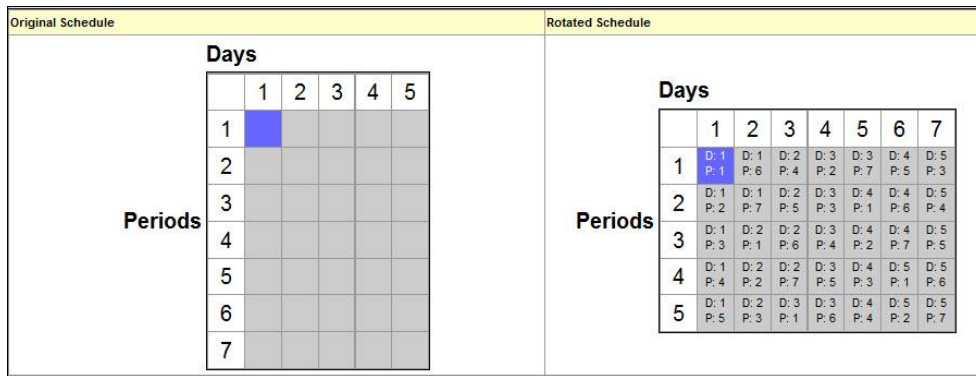
Schedule patterns allow the user to visually “paint” only the valid time slots available for each course. This can significantly reduce the amount of work the schedule builder expends in scheduling each course.

## 5 Schedule Rotations

Another common scheduling practice among secondary schools in the United States, especially in middle schools, is a rotated schedule [3]. A schedule rotation remaps time slots from one schedule matrix (DPC x PPD) to another schedule matrix with the same dimensions or one with different dimensions. Rotations allow users to schedule their schools using simple, uniform patterns and then transform it into a schedule with more complex varied patterns post build.

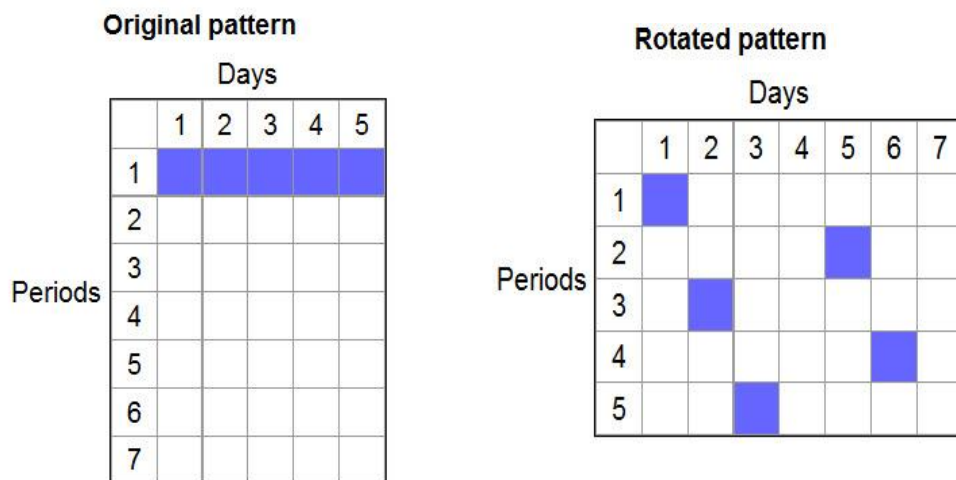
In a flat schedule a course scheduled during the first period of the day is schedule in the first period of the day on each day of the cycle. A simple rotation can vary the period each class is scheduled across the different days in the schedule. This allows a student to learn math at different times of the day on different days in the schedule.

Rotations also allow schools to change the structure of a schedule post build. A common scheduling rotation swaps the periods per day and the days per cycle. For example a 7 x 5 schedule could become a 5 x 7 schedule. In the old schedule an every-day class met for 45 minutes. In the new schedule the same every-day class now only meets 5 days out of 7 but for 63 minutes each meeting. (See Fig.3.1)



**Fig.3.1** Define a rotation: from a 5-day 7-period to a 7-day 5-period schedule

To handle such flexible scheduling modes, simple flat schedule patterns are created and used to build the master schedule. When the scheduling is finished the schedule is transformed into more complex patterns through the rotation. Sections are then being rotated based on the rotation definition. An example of simple flat pattern and the more complex rotated patterns is shown in Fig.3.2.



**Fig.3.2** Original and rotated schedule patterns

## 6 Conclusions

In the United States, secondary schools structure time in a variety of different ways resulting in very different types of schedules. By allowing schools to define the number of periods per day, the number of days per cycle, and the number of terms per year, they have the flexibility to create a broad spectrum of different schedules each meeting variety of different scheduling needs. We looked at three

scheduling models and some variations on these in this extended abstract: the traditional schedule, the block schedules and the hybrid schedules.

The approach reported in this paper is implemented in Aspen master schedule builder, a web-based automate master schedule builder that has been used by many secondary schools in United States and abroad.

## References

1. Irmsher, Karen, Block Scheduling, ERIC Digest, Number 104 <http://www.ericdigests.org/1996-4/block.htm>, 1996
2. John W. Cooper, Block Scheduling: Is this Right for America's Public Schools?, <http://www.johnwcooper.com/papers/blockscheduling.htm>, 2001
3. Williamson, R, Scheduling the Middle Level School to Meet Early Adolescent Needs, Reston VA, National Association of Secondary School Principals, 1993

---

# Using the PEAST Algorithm to Roster Nurses in an Intensive-Care Unit in a Finnish Hospital

Nico Kyngäs<sup>1</sup>, Kimmo Nurmi<sup>1</sup>, Eyjólfur Ingi Ásgeirsson<sup>2</sup>, Jari Kyngäs<sup>1</sup>

<sup>1</sup> Satakunta University of Applied Sciences  
Tiedepuisto 3, 28600 Pori, Finland

<sup>2</sup> Reykjavík University, School of Science and Engineering, Menntavegur 1, 101 Reykjavík, Iceland

**Abstract.** Workforce scheduling has become increasingly important for both the public sector and private companies. Good rosters have many benefits for an organization, such as lower costs, more effective utilization of resources and fairer workloads and distribution of shifts. This paper presents a successful way to roster nurses in an intensive-care unit in a Finnish hospital. The rosters are generated using a population-based local search method called the PEAST algorithm. The algorithm has been integrated into market-leading workforce management software in Finland.

**Keywords:** Nurse Rostering, Staff Scheduling, Workforce Optimization, PEAST algorithm.

## 1 Introduction

Workforce scheduling, also called staff scheduling and labor scheduling, is a difficult and time consuming problem that every company or institution that has employees working on shifts or on irregular working days must solve. Different variations of the problem are NP-hard and NP-complete (Garey and Johnson 1979, Bartholdi 1981, Tien and Kamiyama 1982, Lau 1996), and thus extremely hard to solve. Good overviews of workforce scheduling are published by Alfares (2004), Ernst et al. (2004) and Meisels and Schaefer (2003).

Nurse rostering (Burke 2004) is by far the most studied application area in workforce scheduling. Other successful application areas include airline crews (Dowling et al. 1997), call centers (Beer et al. 2008), check-in counters (Stolletz 2010), ground crews (Lusby et al. 2010), nursing homes (Ásgeirsson 2010), postal services (Bard et al. 2003), retail stores (Chapados et al. 2011) and transport companies (Nurmi et al. 2011).

Most of the workforce scheduling cases in which academic researchers have announced that they have signed a contract with a customer concern nurse rostering (Van Wezel and Jorna 1996, Meyer auf'm Hofe 2001, Diaz et al. 2003, Kawanaka et al. 2003, Bard and Purnomo 2005, Burke et al. 2006, Bilgin et al. 2008, Beddoe et al. 2009). Hospitals tend to be very open about their operational details, enabling easy cooperation with academics who wish to publish the results of their work. However, we believe there is still a gap between academic and commercial solutions. The commercial products may not include the best academic solutions. Yet we have experienced that nurse rostering cooperation between a commercial software vendor and academics does work. According to our experience, the best action plan for real-world nurse rostering research is to cooperate both with a problem owner and a software vendor. Collaboration with software vendors and problem owners allows academics to concentrate on modeling issues and algorithmic power instead of user interfaces, financial management links, customer reports, help desks, etc.

The need for effective commercial workforce scheduling has been driven by the growth in the customer contact center industry and retail sector, in which efficient deployment of labor is of crucial importance. The balance between offering a superior service and reducing costs to generate revenues must constantly be found. There are five basic reasons for the increased interest in nurse rostering optimization. First, hospitals around the world have become more aware of the possibilities in decision support technologies and no longer want to handle the schedules manually. Second, human resources are one of the most critical and most expensive resources for hospitals. Careful planning can lead to significant improvements in productivity. Third, good schedules are very important for the welfare of the staff, resulting in increased happiness and reduction of sick-leaves. Fourth, new algorithms have been developed to tackle previously intractable nurse rostering instances, and, at the same time, computer power has increased to such a level that researchers are able to solve large-scale instances. Finally, one further significant benefit of



automating the scheduling process is the considerable amount of time saved by the administrative nurses involved.

The goal of this paper is to show that the PEAST (Population, Ejection, Annealing, Shuffling, Tabu) algorithm can be used to roster nurses in Finnish hospitals. Section 2 introduces the workforce scheduling process with notes on nurse rostering. It also introduces the necessary terminology. In Section 3 we describe the characteristics of the nurse rostering problems occurring in intensive-care units in Finnish hospitals. Section 4 gives an outline of the PEAST algorithm. Section 5 presents our computational results.

## 2 Workforce Scheduling and Nurse Rostering

Workforce scheduling consists of assigning employees to tasks and shifts over a period of time according to a given timetable. The *planning horizon* is the time interval over which the employees have to be scheduled. Each employee has a total working time that he/she has to work during the planning horizon. Furthermore, each employee has *competences* (qualifications and skills) that enable him/her to carry out certain tasks. Days are divided into *working days* (days-on) and rest days (*days-off*). Each day is divided into periods or timeslots. A *timeslot* is the smallest unit of time and the length of a timeslot determines the granularity of the schedule. A *shift* is a contiguous set of working hours and is defined by a day and a starting period on that day along with a *shift length* (the number of occupied timeslots). Shifts are usually grouped in *shift types*, such as morning (M), day (D) and night (N) shifts. A specific sequence of shifts, such as DDDNN, is called a *stint*. Each shift is composed of a number of *tasks* that should be completed during the shift. A shift or a task requires the employee assigned to it to possess one or more competences. A work schedule for an employee over the planning horizon is called a *roster*. A roster is a combination of shifts and days-off assignments that covers a fixed period of time.

We classify the real-world workforce scheduling process as given in Figure 1. *Workload prediction*, also referred to as demand forecasting or demand modeling, is the process of determining the staffing levels – that is, how many employees are needed for each timeslot in the planning horizon. *Shift generation* is the process of determining the shift structure, tasks to be carried out on particular shifts and the competences needed on different shifts. Traditionally, hospitals work in three shifts – morning, day and night – but in the intensive-care units the customer flow should be considered when constructing the shift structure, as is the case in, e.g. the call center and retail sector businesses. The shifts generated from a solution to the shift generation problem form the input for subsequent phases in the workforce scheduling. Another important goal for shift generation is to determine the size of the workforce required to solve the demand. Note that shifts are created anonymously, so there is no direct link to the employee that will eventually be assigned to the shift.

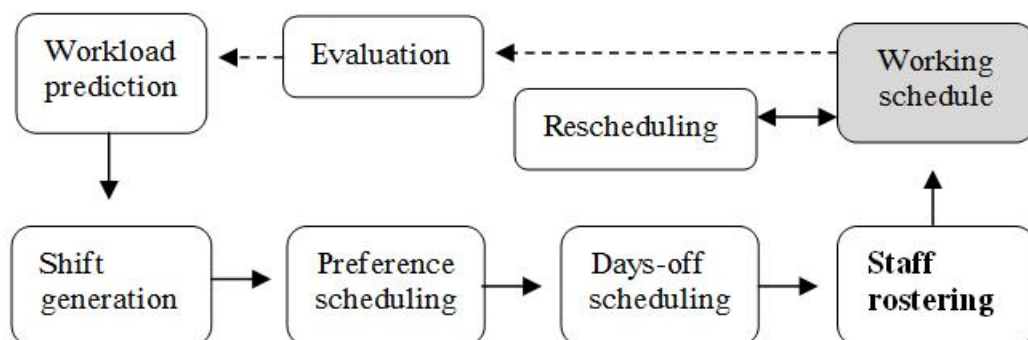


Fig. 1. The real-world workforce scheduling process.

In *preference scheduling*, each employee gives a list of preferences and attempts are made to fulfill them as well as possible. It is very important to pay attention to employee requests. Kellog and Walczak (2007) report that any academic nurse rostering model that does not include some opportunity for preference scheduling will probably not be implemented. Nurses tend to use complex decision-making skills when selecting their personal schedules. The employees' preferences are considered in the days-off scheduling and staff rostering phases.

*Days-off scheduling* deals with the assignment of rest days between working days over a given planning horizon. Days-off scheduling also includes the assignment of vacations and special days, such as union steward duties and training sessions. *Staff rostering*, also referred to as shift scheduling, deals with the assignment of employees to shifts. Days-off and shifts are often scheduled simultaneously. However, if a hospital scheduled days-off every tenth week and rostering staff every second week, the nurses would be able to plan their free time more conveniently.

*Rescheduling* deals with ad hoc changes that are necessary due to sick leaves or other no-shows. The changes are usually carried out manually using some level of computer support. Finally, participation in *evaluation* ranges from the individual employee through personnel managers (administrative nurses) to executives (head nurses). A reporting tool should provide performance measures in such a way that the personnel managers can easily evaluate both the realized staffing levels and the employee satisfaction. When necessary, the workload prediction and/or shift generation can be reprocessed and focused, and the whole workforce scheduling process restarted.

### 3 Nurse Rostering in an Intensive-Care Unit in a Finnish Hospital

We have experience in solving workforce scheduling problems occurring in the transportation industry, see for example (Nurmi and Kyngäs 2011, Nurmi et al. 2011, Kyngäs et al. 2012). Our current research is focused on workforce scheduling in call centers and hospitals. Based on our experiences, we believe that the framework for implementation-oriented staff scheduling we presented in (Ásgeirsson et al. 2011) can be used to model a considerable number of real-world workforce scheduling scenarios. With the help of administrative staff from Finnish hospitals we used the framework to describe the problem occurring in intensive-care units in Finnish hospitals. This problem includes five characteristics that are not always present in the nurse rostering cases reported in the academic literature:

1. The number of nurses is over 100
2. The nurses are grouped in four categories based on their total working hours within the planning horizon (100%, 78.43%, 50% and 40% of the full-time work)
3. Some shifts last more than 14 hours and actually include two consecutive shifts
4. Some nurses should always work on the same shifts
5. The nurses' wishes for days-off and shifts cover as much as 50% of their total work.

The implementation should present a wide variety of real-world constraints and be tractable enough to enable the addition of new constraints. It is important to concentrate on the acceptance by and satisfaction of both the administrative staff and the nurses. Despite the fact that the algorithm should be as robust as possible, no parameter tuning should be expected from the end-users. On the other hand, it should be possible for the end-users to influence different aspects of the algorithm, like weighting between constraints or limiting running times, if he/she wishes to.

We are well aware that it is difficult to incorporate the experience and expertise of the administrative nurses into a nurse rostering system. They often have extremely valuable knowledge, experience and detailed understanding of their specific staffing problem, which will vary from hospital to hospital. To formalize this knowledge into constraints is not an easy task. Still, we believe that the model given in this section builds up a solid foundation for nurse rostering scenarios in hospitals and specifically in intensive-care units.

The most important goal is to minimize understaffing and overstaffing. Low-quality rosters can lead either to an undersupply of nurses with a need to hire part-time nurses or an oversupply of nurses with too much idle time, implicating a loss of efficiency. The overall objective is to meet daily staffing requirements and personal preferences at minimum penalty without violating work contracts and government regulations. The framework presented in (Ásgeirsson et al. 2011) makes no strict distinction between hard and soft constraints; that will be given by the instances themselves. The goal in an instance is to find a feasible solution that is most acceptable for the hospital, that is, a solution that has no hard constraint violations and that minimizes the weighted sum of the soft constraint violations. The weights will also be given by the instances themselves and will vary between hospitals. Still, one should bear in mind that an instance is usually just an

approximation of practice. In reality, hard constraints can turn out to be soft, if necessary, while giving weights to the soft constraints can be difficult.

The framework classifies the constraints into coverage, regulatory and operational requirements, and operational and personal preferences. The coverage requirement ensures that there are a sufficient number of nurses on duty at all times. The regulatory requirements ensure that the nurses' work contract and government regulations are respected. Operational and personal preferences should be met as far as possible; this leads to greater staff satisfaction and commitment, and reduces staff turnover.

We discuss the problem occurring in intensive-care units (ICU) in Finnish hospitals using an example from the Satakunta Hospital District which offers specialized medical care services for the 231,000 residents of the Satakunta region. The number of nurses in the ICU is 130. The problem can be modeled as follows; the constraint numbers refer to the constraints presented in (Ásgeirsson et al. 2011):

#### *Coverage requirement*

- (C1) An employee cannot be assigned to overlapping shifts
- (C2) A minimum number of employees with particular competences must be guaranteed for each shift
- (C4) A balanced number of surplus employees must be guaranteed in each working day

#### *Regulatory requirements*

- (R1) The required number of working days, working hours and days-off within a timeframe must be respected
- (R2) The required number of holidays within a timeframe must be respected
- (R3) The required number of free weekends (both Saturday and Sunday free) within a timeframe must be respected
- (R5) The minimum time gap of rest time between two shifts must be respected
- (R6) The number of special shifts (such as union steward duties and training sessions) for particular employees within a timeframe must be respected
- (R7) Employees cannot work consecutively for more than  $w$  days

#### *Operational requirements*

- (O1) An employee can only be assigned to a shift he/she has competence for
- (O2) At least  $g$  working days must be assigned between two separate days-off breaks
- (O5) An employee assigned to a shift type  $t_1$  must not be assigned to a shift type  $t_2$  on the following day (certain stints are not allowed)

#### *Operational preferences*

- (E1) Single days-off should be avoided
- (E2) Single working days should be avoided
- (E3) The maximum length of consecutive days-off is  $d$
- (E4) A balanced assignment of single days-off and single working days must be guaranteed between the employees
- (E5) A balanced assignment of different shift types must be guaranteed between the employees
- (E7) A balanced assignment of weekdays must be guaranteed between employees
- (E8) Assign or avoid a given shift type before or after a free period (days-off, vacation)

#### *Personal preferences*

- (P1) Assign or avoid assigning given employees to the same shifts
- (P2) Assign a requested day-on or avoid a requested day-off
- (P3) Assign a requested shift or avoid an unwanted shift.

Often, a nurse cannot be assigned to more than one shift per day. However, two consecutive shifts per day are allowed in Finnish hospitals (see shift types C and E described later). The definition of constraint C1 allows two or more shifts to be assigned provided they do not overlap. Employees have seven possible competences: casting skill, intravenous skill, transportation skill, help skill, novice-nurse, intermediate-nurse and top-nurse. It is obvious that a nurse can only be assigned to a shift he/she has competence for (O1). The minimum number of employees of particular competences for time of day (C2) is given in Table 1. Note that the competences may overlap, e.g. a top nurse probably has an intravenous skill as well. Quite often, hospitals and ICUs have more nurses working than are needed to cover the minimum number of nurses each working day. The surplus nurses are used to cover the expected sick leaves and other no-shows. In our example case a balanced number of surplus nurses must be guaranteed in each working day (C4).

**Table 1.** Minimum number of employees with particular competences for each time of day.

	min #emp
Casting skill	1
Intravenous skill	11
Transportation skill	1
Help skill	1
Novice-nurse	0
Intermediate-nurse	5 (at night) 10 (otherwise)
Top-nurse	4

Within the last two years the hospital has started to consider the patient flow as a basis for the shift structure. Even though the shift structure is not near-optimal, it is a good start towards generating the shifts based on real workload prediction in the near future. The shift structure for the ICU is given in Table 2. Note that C and E are so-called double-shifts that last 14 hours and 30 minutes.

**Table 2.** The shift structure.

Code	Description	From	To
A	Morning	07.30	15.15
X	Transport I	07.30	15.15
U	Admin	07.30	15.30
B	Help I	07.30	16.00
C	Double	07.30	22:00
E	Transport D	07.30	22.00
O	Acute I	10.00	18.00
Z	Acute II	12.00	20.00
F	Special	14.00	22.00
I	Evening	15.00	22.00
P	Help II	15.00	23.00
J	Transport II	15.00	23.00
R	Help III	16.00	24.00
D	Acute III	17.00	24.00
Y	Night	21.30	07.45

The planning horizon is six weeks. The total working hours for each full-time nurse are 229 hours and 30 minutes (R1). The working hours can also be 180h, 114h 45min or 181h 30min if a nurse is on part-time pension or has small children. The holidays (R2) and special shifts (R6) are included in the working hours.

The working days and shifts are built up using the following rules. The number of free weekends within a timeframe must be at least two (R3). At least nine hours of rest are required between two shifts (R5). Nurses cannot work consecutively for more than nine days (R7). At least two working days must be assigned between two separate days-off (O2). Single days-off and single working days should be avoided (E1 and E2). The maximum length of consecutive days-off is four (E3). A balanced assignment of single days-off and single working days must be guaranteed between the employees (E4). A nurse assigned to a night shift (code Y) must not be assigned to an early shift (A,X,U,B,E,C) the following day (O5). Furthermore, a night/early shift should be avoided before/after a free period (E8).

Each six-week planning horizon is preceded with a phase where nurses express their wishes for days-off and shifts (P2 and P3). These wishes cover as much as 50% of their total work on average. This is why a balanced assignment of different shift types cannot be guaranteed between the employees as given in constraint E5. The same holds for balancing the assignment of weekdays (E7). A special request is that some nurses should always work on the same shifts because they travel together to work from the nearby cities (P1).

As per the nurse rostering problem classification given in (De Causmaecker and Vanden Berghe 2011), the problem could be classified as *ASBC|V3O|PX*.

The next section gives an outline of the PEAST algorithm that is used to solve the problem occurring in intensive-care units (ICU) in Finnish hospitals and especially in the Satakunta Hospital District. Section 5 presents our computational results.

## 4 The PEAST Algorithm

The usefulness of an algorithm depends on several criteria. The two most important are the quality of the generated solutions and the algorithmic power of the algorithm (i.e. its efficiency and effectiveness). Other important criteria include flexibility, extensibility and learning capabilities. We can steadily note that our PEAST algorithm (Kyngäs 2011) realizes these criteria. The acronym PEAST stems from the methods used as Population, Ejection, Annealing, Shuffling and Tabu. It has been used to solve real-world school timetabling problems (Nurmi and Kyngäs 2007), real-world sports scheduling problems (Kyngäs and Nurmi 2009) and real-world workforce scheduling problems (Kyngäs and Nurmi 2011).

The PEAST algorithm is a population-based local search method. As we know, the main difficulty for a local search is

- 1) to explore promising areas in the search space that is, to zoom-in to find local optimum solutions to a sufficient extent while at the same time
- 2) avoiding staying stuck in these areas for too long and
- 3) escaping from these local optima in a systematic way.

Population-based methods use a population of solutions in each iteration. The outcome of each iteration is also a population of solutions. Population-based methods are a good way to escape from local optima. The PEAST algorithm uses GHCM, the Greedy Hill-Climbing Mutation heuristic introduced in (Nurmi 1998) as its local search method. The outline of the algorithm is given in Figure 2 and the pseudo-code of the algorithm is given in Figure 3.

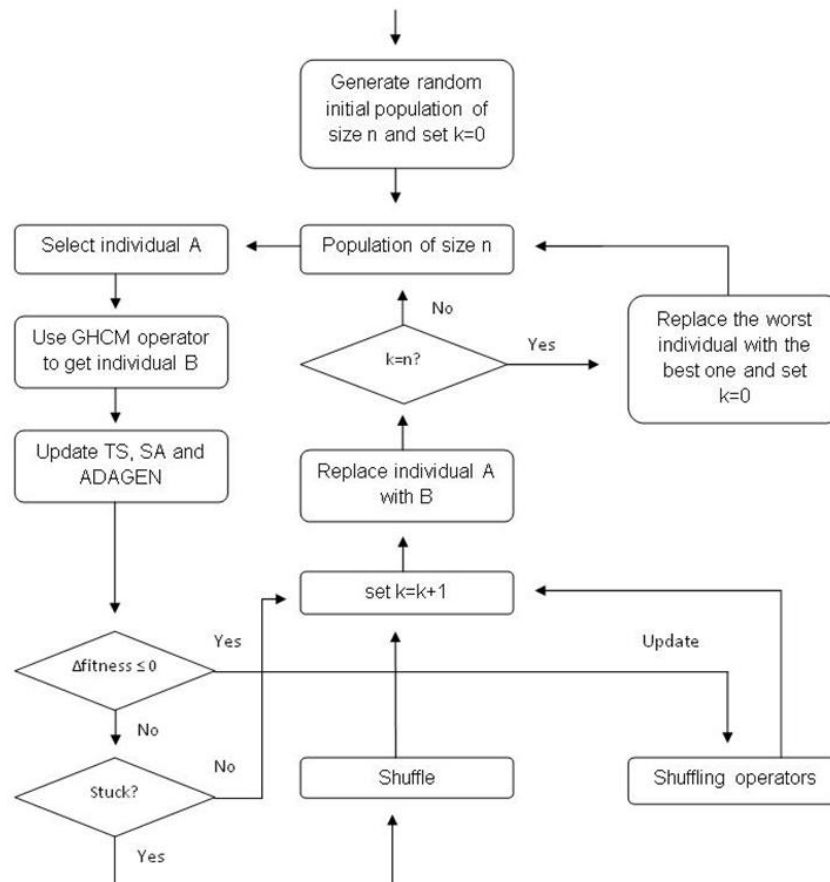


Fig. 2. The outline of the population-based PEAST algorithm.

The reproduction phase of the algorithm is, to a certain extent, based on steady-state reproduction: the new individual replaces the old one if it has a better or equal objective function value. Furthermore, the least fit is replaced with the best one when  $n$  better individuals have been found, where  $n$  is the size of the population. Marriage selection is used to select an individual from the population for a single GHCM operation. In the marriage selection we randomly pick an individual,  $A$ , and then we try at most  $k - 1$  times to randomly pick a better one. We choose the first better individual, or, if none is found, we choose  $A$ .

---

```

Set the time limit  $t$ , no_change limit  $m$  and the population size  $n$ 
Generate a random initial population of individuals
Set  $no\_change = 0$  and  $better\_found = 0$ 
WHILE elapsed_time <  $t$ 
  REPEAT  $n$  times
    Select an individual  $A$  by using a marriage selection with  $k = 3$ 
    (explore promising areas in the search space)
    Apply GHCM to  $A$  to get a new individual  $A'$ 
    Calculate the change  $\Delta$  in objective function value
    IF  $\Delta < 0$  THEN
      Replace  $A$  with  $A'$ 
      IF  $\Delta < 0$  THEN
         $better\_found = better\_found + 1$ 
         $no\_change = 0$ 
      END IF
    ELSE
       $no\_change = no\_change + 1$ 
    END IF
  END REPEAT
  IF  $better\_found > n$  THEN
    Replace the worst individual with the best individual
    Set  $better\_found = 0$ 
  END IF
  IF  $no\_change > m$  THEN
    (escape from the local optimum)
    Apply shuffling operators
    Set  $no\_change = 0$ 
  END IF
  (avoid staying stuck in the promising search areas too long)
  Update simulated annealing framework
  Update the dynamic weights of the hard constraints (ADAGEN)
END WHILE
Choose the best individual from the population

```

---

**Fig. 3.** The pseudo-code of the PEAST algorithm.

The heart of the GHCM heuristic is based on similar ideas to the Lin-Kernighan procedures (Lin and Kernighan 1973) and ejection chains (Glover 1992). The basic hill-climbing step is extended to generate a sequence of moves in one step, leading from one solution candidate to another. The GHCM heuristic moves an object,  $o_1$ , from its old position,  $p_1$ , to a new position,  $p_2$ , and then moves another object,  $o_2$ , from position  $p_2$  to a new position,  $p_3$ , and so on, ending up with a sequence of moves.

Picture the positions as cells, as shown in Figure 4. The initial cell selection is random. The cell that receives an object is selected by considering all the possible cells and selecting the one that causes the least increase in the objective function when only considering the relocation cost. Then, another object from that cell is selected by considering all the objects in that cell and picking the one for which the removal causes the biggest decrease in the objective function when only considering the removal cost. Next, a new cell for that object is selected, and so on. The sequence of moves stops if the last move causes an increase in the objective function value and if the value is larger than that of the previous non-improving move. Then, a new sequence of moves is started. A tabu list prevents reverse order moves in the same sequence of moves. The initial solution is randomly generated.

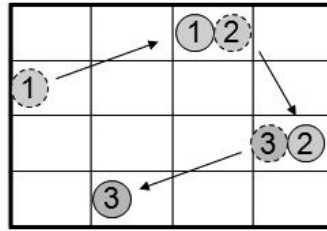


Fig. 4. A sequence of moves in the GHCM heuristic.

In the nurse rostering problem, each *row* corresponds to a nurse, and each *column* to a day. An *object* is a shift. A *move* involves removing a shift from a certain day and inserting it into another day.

The decision whether or not to commit to a sequence of moves in the GHCM heuristic is determined by a refinement (Nurmi 1998) of the standard simulated annealing method (Laarhoven and Aarts 1987). Simulated annealing is useful to avoid staying stuck in the promising search areas for too long. The initial temperature  $T_0$  is calculated by

$$T_0 = 1 / \log(1/X_0) . \quad (1)$$

where  $X_0$  is the degree to which we want to accept an increase in the cost function (we use a value of 0.75). The exponential cooling scheme is used to decrement the temperature:

$$T_k = \alpha T_{k-1} , \quad (2)$$

where  $\alpha$  is usually chosen between 0.8 and 0.995. We stop the cooling at some predefined temperature. Therefore, after a certain number of iterations,  $m$ , we continue to accept an increase in the cost function with some constant probability,  $p$ . Using the initial temperature given above and the exponential cooling scheme, we can calculate the value:

$$\alpha = (-1/(T_0 \log p))^{-m} . \quad (3)$$

We choose  $m$  equal to the maximum number of iterations with no improvement to the cost function and  $p$  equal to 0.0015.

A hyperheuristic (Cowling et al. 2000) is a mechanism that chooses a heuristic from a set of simple heuristics and applies it to the current solution, then chooses another heuristic and applies it, and continues this iterative cycle until the termination criterion is satisfied. We use the same idea, but the other way around. We apply shuffling operators to escape from the local optimum. We introduce a number of simple heuristics that are normally used to improve the current solution but, instead, we use them to shuffle the current solution - that is, we allow worse solution candidates to replace better ones in the current population. In the nurse rostering problem the PEAST algorithm uses two shuffling operations:

- 1) Move a random shift to a random day and repeat this  $l_1$  times.
- 2) Swap two random shifts and repeat this  $l_2$  times.

A random shuffling operation is selected every  $l/20$ th iteration of the algorithm, where  $l$  equals the maximum number of iterations with no improvement to the cost function. The best results were obtained using the values  $l_1 = 5$  and  $l_2 = 3$ .

We use the weighted-sum approach for multi-objective optimization. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. We use ADAGEN, the ADaptive GENetic penalty method introduced in (Nurmi 1998) to assign dynamic weights to the hard constraints. This means that we are searching for a solution that minimizes the (penalty) function

$$\sum_i \alpha_i f_i(x) + \sum_i c_i g_i(x) , \quad (4)$$

where

- $\alpha_i$  = a dynamically adjusted weight for hard constraint  $i$
- $f_i(x)$  = cost of violations of hard constraint  $i$
- $c_i$  = a fixed weight for soft constraint  $i$
- $g_i(x)$  = cost of violations of soft constraint  $i$

The hard constraint weights are updated every  $k$ th generation using the method given in (Nurmi 1998).

## 5 Computational Results

This section presents our results for solving a nurse rostering instance occurring in an intensive-care unit in the Satakunta Hospital District in Finland. The unit has 130 employees. Section 3 outlined the characteristics and constraints of the problem. Table 3 summarizes the hard and soft constraints of the problem. As the hard constraints state, the most important goal is to find a solution that has no overlapping shifts and guarantees a sufficient number of competences for each shift, and where employees do not work consecutively for more than nine days, have sufficient rest time between shifts and are not assigned to a forbidden shift before/after a night shift. As the soft constraint penalties state, the most important goal is to find individual rosters with exactly the required number of working hours. The rosters with less than 229 hours and 30 minutes for full-time nurses are considered as bad as the rosters with more than 229 hours and 30 minutes. The second most important goal is to fulfill the employee's requests.

Table 3 shows the manual solution and the PEASt solution to the problem. Neither solution has any hard constraint violations. The PEASt algorithm only needed 100 employees for generating a feasible and acceptable schedule. Note that the employees on vacation are not counted in this value. The PEASt algorithm was able to find a solution where all but one employee had exactly the required number of working hours. The algorithm also found a solution where 99% of all the employees' wishes were fulfilled even though those wishes covered as much as 50% of the employees' total work on average. Furthermore, the PEASt solution is clearly better at the number of single working days and finding a suitable weekend solution (see R3 and E7).

**Table 3.** The hard and soft constraints of the problem, the penalties for soft constraint violations, the manual solution and the solution obtained by the PEASt algorithm. The solutions indicate the number of violations for the constraints.

Constraint	Description	Penalty	Manual solution	PEAST solution
C1	Overlapping shifts	Hard	0 (108*)	0 (100*)
C2	Number of competences	Hard	0	0
C4	Balanced surplus employees	2	22	4
R1	Working hours / nurse	10	140	3
R3	Free weekends	4	50	16
R5	Sufficient rest time	Hard	0	0
R7	Consecutive working days	Hard	0	0
O1	Sufficient competence	Hard	0	0
O2	Working days in between	1	285	53
O5	Forbidden stints	Hard	0	0
E1	Single days-off	4	274	124
E2	Single working days	2	285	53
E3	Consecutive days-off	2	34	0
E4	Balanced singles	1	18	5
E5	Balanced shift types	1	37	46
E7	Balanced weekdays	1	343	127
E8	Forbidden shifts	4	0	0
P1	Same shifts	5	0	0
P2	Requested days-on	6	76% fulfilled	99% fulfilled
P3	Requested shifts	6	57% fulfilled	99% fulfilled

\* The required number of employees needed for generating the schedule

The PEASt solution was found by generating ten solutions and selecting the best one. The algorithm was run on an Intel Core 2 Extreme QX9775 PC with a 3.2GHz processor and 4GB of random access memory running 64bit Windows Vista Business Edition. The best solution was found in 18 hours of computer time. The time may appear to be long. However, the point here is not to find a solution fast enough and with sufficient quality, but to find a solution of high quality. It is perfectly reasonable to run the algorithm overnight, because the solution is only needed once every six weeks. Note also that the manual solution took three weeks to generate. The detailed data for the instance can be obtained from the authors by email.



The Hospital Board members were very satisfied with our results. We are currently negotiating with them to optimize their overall workforce management process. This includes 1) generating an optimal shift structure based on the predicted patient flow, 2) optimizing the employees' preferences with a centralized self-scheduling system and 3) optimizing the days-off and shift assignments. As was stated in Section 1, the best action plan for real-world nurse rostering research is to cooperate with both a problem owner and a software vendor. We have a business partner that has workforce management software that already includes our optimization component. We are now looking to include nurse rostering in that software as well.

## 6 Conclusions and Future Work

We described an effective method for rostering nurses in an intensive-care unit in a Finnish hospital. The rosters were generated using a population-based local search method called the PEAST algorithm. The acronym PEAST stands for Population, Ejection, Annealing, Shuffling and Tabu, which represent the building blocks of the algorithm. The PEAST algorithm is flexible, easily extended and has good learning capabilities. The algorithm is based on a thorough local search method while still containing a strong global search element through the population based setup and randomized shuffling. The hospital was very satisfied with our results. We are currently negotiating with them to optimize their overall workforce management process. The PEAST algorithm has been integrated into market-leading workforce management software in Finland.

Future work includes modeling the instance presented in this paper using the xml-based modeling format introduced and managed by Tim Curtois (2010). We will also use the PEAST algorithm to solve the benchmark instances in (Curtois 2010). Our direction for future research is to strengthen our competence in workforce optimization concerning contact centers.

## References

- Alfares, H.K. (2004). Survey, categorization and comparison of recent tour scheduling literature. *Annals of Operations Research* 127, 145-175.
- Ásgeirsson, E. I. (2010). Bridging the gap between self schedules and feasible schedules in staff scheduling. In *Proc of the 8th Conference on the Practice and Theory of Automated Timetabling*, Belfast, Ireland.
- Ásgeirsson, E.I., Kyngäs, J., Nurmi, K., Stølevik, M. (2011). A Framework for Implementation-Oriented Staff Scheduling. *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory and Applications*, Phoenix, USA.
- Bard, J. F., Binici, C., Desilva, A. H. (2003). Staff Scheduling at the United States Postal Service. *Computers & Operations Research* 30, 745-771.
- Bard, J., Purnomo H. (2005). Hospital-wide reactive scheduling of nurses with preference considerations. *IIE Trans.* 37(7), 589-608.
- Bartholdi, J.J. (1981). A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering. *Operations Research* 29, 501-510.
- Beddoe, G.R., Petrovic, S., Li, J. (2009). A Hybrid Metaheuristic Case-based Reasoning System for Nurse Rostering. *Journal of Scheduling* 12, 99-119.
- Beer, A., Gaertner, J., Musliu, N., Schafhauser, W., Slany, W. (2008). Scheduling breaks in shift plans for call centers. In *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling*, Montréal, Canada.
- Bilgin, B., De Causmaecker, P., Rossie, B., Vanden Berghe G. (2008). Local Search Neighbourhoods to Deal with a Novel Nurse Rostering Model. In *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling*, Montréal, Canada.
- Burke, E., De Causmaecker P., Petrovic S., Vanden Berghe G. (2006). Metaheuristics for Handling Time Interval Coverage Constraints in Nurse Scheduling. *Applied Artificial Intelligence*, 743-766.
- Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H. (2004). The State of the Art of Nurse Rostering. *Journal of Scheduling* 7(6), 441-499
- Chapados, N., Joliveau, M., Rousseau L-M. (2011). Retail Store Workforce Scheduling by Expected Operating Income Maximization, *CPAIOR*, 53-58.
- Cowling, P., Kendall, G., Soubeiga, E. (2000). A hyperheuristic Approach to Scheduling a Sales Summit. *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, 176-190.
- Curtois, T. (Last update August 2010). Staff Rostering Benchmark Data Sets [Online]. Available: <http://www.cs.nott.ac.uk/~tec/NRP/>.
- De Causmaecker, P., Vanden Berghe, G. (2011). A categorisation of nurse rostering problems. *Journal of Scheduling*, 14 (1), 3-16.

- Diaz, T., Ferber, D., deSouza C., Moura A. (2003). Constructing nurse schedules at large hospitals. *Internat. Trans. Oper. Res.* 10(3), 245–265.
- Dowling, D., Krishnamoorthy, M., Mackenzie, H., Sier, D. (1997). Staff rostering at a large international airport. *Annals of Operations Research* 72, 125-147.
- Ernst, A. T., Jiang H., Krishnamoorthy, M., Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153 (1), 3–27.
- Garey, M.R., Johnson, D.S. (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness*, Freeman.
- Glover, F. (1992). New ejection chain and alternating path methods for traveling salesman problems. *Computer Science and Operations Research: New Developments in Their Interfaces*, edited by Sharda, Balci and Zenios, Elsevier, 449–509.
- Kawanaka, H., Yoshikawa T., Shinogi T., Tsuruoka S. (2003). Constraints and search efficiency in nurse scheduling problem. In *Proc. Internat. Sympos. Comput. Intelligence Robotics Automation* 1, 312–317.
- Kellogg D.L., Walczak S. (2007). Nurse Scheduling: From Academia to Implementation or Not. *Interfaces* 37(4), 355–369.
- Kyngäs, J. (2011). *Solving Challenging Real-World Scheduling Problems*. Dissertation. Dept. of Information Technology, University of Turku, Finland.
- Kyngäs, J., Nurmi, K. (2009). Scheduling the Finnish Major Ice Hockey League. *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling*, Nashville, USA.
- Kyngäs, J., Nurmi, K. (2011). Shift Scheduling for a Large Haulage Company. *Proceedings of the 2011 International Conference on Network and Computational Intelligence*, Zhengzhou, China, 2011.
- Kyngäs, N., Nurmi, K., Kyngäs, J. (2012). Optimizing Large-Scale Staff Rostering Instances. *Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists*, Hong Kong.
- Lau, H. C. (1996). On the Complexity of Manpower Shift Scheduling. *Computers and Operations Research* 23(1), 93-102.
- Lin, S., Kernighan, B. W. (1973). An effective heuristic for the traveling salesman problem. *Operations Research* 21, 498–516.
- Lusby T., Dohn A., Range T., Larsen J. (2010). Ground Crew Rostering with Work Patterns at a Major European Airlines. In *Proc of the 8th Conference on the Practice and Theory of Automated Timetabling*, Belfast, Ireland.
- Meisels, A., Schaerf, A. (2003). Modelling and solving employee timetabling problems. *Annals of Mathematics and Artificial Intelligence* 39, 41–59.
- Meyer auf'm Hofe, H. (2001). Solving rostering tasks by generic methods for constraint optimization. *Internat. J. Foundations Comput. Sci.* 12(5), 671–693.
- Nurmi, K. (1998). *Genetic Algorithms for Timetabling and Traveling Salesman Problems*. Dissertation. Dept. of Applied Math., University of Turku, Finland, 1998. Available: <http://www.bit.spt.fi/cimmo.nurmi/>
- Nurmi, K., Kyngäs, J. (2007). A Framework for School Timetabling Problem. *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications*, Paris, France, 386–393.
- Nurmi, K., Kyngäs, J. (2011). Days-off Scheduling for a Bus Transportation Staff. *International Journal of Innovative Computing and Applications* Volume 3 (1), Inderscience, UK.
- Nurmi K., Kyngäs J., Post G. (2011). Driver Rostering for Bus Transit Companies. *Engineering Letters* 19(2), 125–132.
- Stolletz, R. (2010). Operational workforce planning for check-in counters at airports. *Transportation Research Part E* 46, 414-425.
- Tien J, Kamiyama A. (1982). On Manpower Scheduling Algorithms. *SIAM Rev.* 24 (3), 275–287.
- van Laarhoven, P.J.M., Aarts, E.H.L. (1987). *Simulated annealing: Theory and applications*, Kluwer Academic Publishers.
- Van Wezel, W., Jorna R. (1996). Scheduling in a generic perspective: Knowledge-based decision support by domain analysis and cognitive task analysis. *Internat. J. Expert Systems* 9(3), 357–381.

---

## A Tour Scheduling Problem with Fixed Jobs: use of Constraint Programming

Tanguy Lapègue · Damien Prot ·  
Odile Bellenguez-Morineau

Received: date / Accepted: date

**Abstract** This paper presents a constraint programming approach to solve a specific scheduling problem arising in a company specialized in drug evaluation and pharmacology research. The aim is to build employee timetables covering the demand given by a set of fixed tasks. The optimality criterion concerns the equity of the workload sharing. A solution to this problem is the assignment of all tasks whose resulting working shifts respect tasks requirements as well as legal and organizational constraints. Scheduling problems usually consider a fixed set of shifts which have to be assigned to a given number of employees whereas in our problem shifts are not fixed and must be deduced from the task assignment.

**Keywords** Tour Scheduling Problem · Fixed Job Scheduling Problem · Constraint Programming

### 1 Introduction

Personnel scheduling problems tackle the difficult task of building employee rosters respecting legal and organizational constraints in order to satisfy the demand. These problems are of tremendous importance for services oriented companies, especially for those working around the clock. Consequently, many researchs have been carried out into this area (see [12] for an overview). These complex and highly constrained problems proved to be very difficult to solve in a satisfactory way and even more to optimality.

In this paper, we present a real-world problem which arises in a company specialized in drug evaluation and pharmacology research. The problem at hand is to build fine rosters which respect legal and organizational constraints. This task is currently hand-performed on a weekly basis by the chief nurse, which is very time-consuming.

The remainder of the paper is organized as follows: section 2 is devoted to the description of the problem, section 3 presents some related works, in section 4 we

---

LUNAM Université, École des Mines de Nantes, IRCCyN (UMR CNRS 6597), 44307 Nantes Cedex 3, Nantes (France),  
E-mail: {tanguy.lapegue, damien.prot, odile.morineau}@mines-nantes.fr

propose a modelling of our problem, section 5 describes some branching strategies which aim at finding quickly a good solution. Our approach is validated by experimental results in section 6.

## 2 Problem description

The company carries out clinical studies on behalf of pharmaceutical laboratories which need to control the impact of newly developed drugs on the human body. The company recruits volunteers who are hospitalized during the whole study so that nurses could perform each required task on each volunteer. Laboratories deliver to the company a very specific study procedure which contains a description of all the clinical tasks to be performed, along with their relative starting time and duration. Based on this protocol, the company needs to assign these tasks to qualified and available employees. The individual plannings resulting from this task assignment have to respect a set of legal and organizational constraints, and also, up to a point, they are expected to be as fair as possible, which makes the scheduling task very difficult and time-consuming for the chief nurse.

During a week, employees divide their working time between three kinds of job:

1. **Clinical tasks** are fixed by the protocol and must be performed at the given starting minute, which means that the granularity of the problem drops to the minute. These tasks are fixed whenever during the day of the week and the hour of the day. The chief nurse has to assign one employee to every clinical task.
2. **Compulsory administrative tasks** are also fixed but they are already assigned before the shift-building procedure. This kind of tasks, such as meetings and trainings, are counted as working time, but not as clinical working time, and they could be assigned to more than one employee (in case of meetings within the company). These assignments lead to fixed periods of clinical unavailability that have to be included in designed shifts.
3. **Free administrative tasks**, such as medical reports writing, are not fixed and do not require any specific skills. Each employee has a specific set of administrative tasks to do during the week, and they are free to work on it whenever they want, provided they are not already assigned in the mean time to clinical or compulsory administrative tasks. Consequently, these tasks do not appear in the final timetable.

One characteristic of the problem lies in the lack of fixed shifts: employees could start and end their day whenever it is necessary, provided the resulting shift sequences respect the set of hard constraints due to work regulation, company organization, and nurses agreements. Consequently, working days refer to working periods and they may overlap two calendar days if needed. The main constraints are summarized in the following.

### Organizational constraints

- **HC 1:** Employees cannot perform tasks which require unmastered skills.
- **HC 2:** Employees cannot perform tasks while unavailable.
- **HC 3:** Every clinical task must be assigned to one employee.
- **HC 4:** The assignment of compulsory administrative tasks must be respected.
- **HC 5:** Employees must finish a task before starting another one.

### Work regulation / nurses agreements constraints

- **HC 6:** The daily working load must not exceed 10 hours.
- **HC 7:** The weekly working load must not exceed 48 hours.
- **HC 8:** The duration of a working day must not exceed 11 hours.
- **HC 9:** The duration of a rest period must not be less than 11 hours.
- **HC 10:** The duration of the weekly rest must not be less than 35 hours.
- **HC 11:** Series of consecutive working days must not exceed 6 days.
- **HC 12:** Depending on their working days, nurses have different breaks.

During the task assignment process, the chief nurse takes into account this whole set of hard constraints which might lead to dead end, meaning that the problem admits no solution. In this case, the chief-nurse strengthen the workforce with externals, who abide by the same rules as regular workers.

### Objective function

Hard constraints must be satisfied, but they only ensure the feasibility of schedules. In order to build fine schedules, the chief nurse takes into account multiple criteria based on equity among employees. In this paper, we consider the most important of these criteria: we want to share the workload resulting from clinical and writing tasks in a fair way. However, some nurses are less confident with the writing of medical reports than others. As a consequence, the chief nurse associates to each employee a weekly targeted clinical load: nurses who are less confident with writing tasks have a higher targeted clinical load in order to counterbalance the higher administrative load of nurses who are more confident with writing tasks. The idea of our objective is to fit with this targeted clinical load in order to get "fair" schedules. As a consequence our objective is to minimize the difference between the highest and the lowest nurse's gap value which is defined as the difference between the clinical targeted time and the clinical assigned time. A small difference between the highest gap and the smallest gap means that the workload is well balanced among nurses.

### Dimensions of the problem

A typical problem involves 200 tasks, whose durations range from 5 minutes to 4 hours, which have to be assigned to about 20 nurses whose set of skills is closed to 30 different skills. One characteristic of our problem lies in the time granularity which drops to the minute over a scheduling horizon of a week. This might look like an excess of precision, but it is very important for the company to follow scrupulously the given protocol. To highlight this fact, one can state the use of synchronised clocks in the whole building.

### Problem studied

As a first step towards the resolution of this industrial problem, we propose to focus on the problem of designing schedules and assigning tasks to the regular workforce, i.e. externals are not taken into account. By doing this, we simulate the first step of the hand-performed resolution process which is done by the chief nurse. In order to provide as much information as possible regarding to needs on externals, we relax the constraint **HC 3**, which allows us to find solutions with an incomplete assignment of tasks to workers.

Constraints related to nurses breaks are particularly complicated because they deal with several kind of breaks which depends on the kind of shift performed:

- some of them have to occur on a given time window whereas others are not time constrained,
- some of them are included in the working time whereas others are not,
- some of them last a few minutes whereas others last one hour.

From our point of view, ensuring these breaks would require heavy constraints without improving much schedules. This may be explained in two steps:

1. because of the fixed start and end of clinical tasks, it is highly probable that short breaks will be respected automatically,
2. the chief nurse pointed out that nurses are very flexible regarding to their breaks, and they are free to exchange some tasks in order to improve their schedules.

Consequently, in this paper, we decided to put aside the constraint **HC 12**. However, warnings are displayed in a post-processing step to alert the chief nurse about these breaks.

In this paper, compulsory administrative tasks are considered to be fixed in time because they are scheduled by the chief nurse before the assignment of clinical tasks, which is the part of the problem we focus on. It would have been possible to give some freedom to these tasks in order to schedule them during the resolution, but it would have make the problem much harder to solve. Consequently, we consider them as data, which is exactly the way it is done in the company.

### 3 Related work

Over the years, many approaches have been proposed in order to model and solve Personnel Scheduling Problems (see [12] for an overview). Mathematical models, usually based on the Dantzig set-covering formulation, often achieve the lowest cost solutions but they are difficult and time-consuming to implement. In particular, specific constraints and objectives may be difficult to express easily. Consequently, research often focus both on simplified problems and general methods. In [3] for instance, the authors proposed a general mathematical model covering several personnel scheduling problems. Another important trend has been to develop implicit modeling in order to tackle more complex problems. In [2] for instance, the authors present an implicit integer linear programming formulation for the inclusion of meal/rest-break flexibility. On the contrary, metaheuristics, such as Tabu Search, Simulated Annealing and Genetic Algorithms, offer the opportunity to incorporate problems specificities. These approaches do not guarantee the optimality of the solution, but they are quite robust and they could be adapted even to the smallest problem specificity. They have been successfully tested to varying kind of real-world problems (see for instance [5] and [8]). Constraint Programming (CP) offers a promising alternative: CP is very close to a form of declarative programming, and consequently, it offers very powerful tools to state complex problems and produce flexible implementations, which is desirable in a business context. For instance, the Nurse Rostering Problem (NRP), and also the Crew Rostering Problem (CRP), which are some of the most constrained personnel

scheduling problems, belong to the set of problems that could be effectively stated as a constraint satisfaction problem (see [21] and [9]).

### **Tour Scheduling Problem**

Personnel scheduling problems could be addressed in two steps: the first step, referred to as the days-off problem, aims at assigning days-off to workers whereas the second one, referred to as the shift scheduling problem, consists of assigning shift sequences to workers. The combined version of these two problems is referred to as the Tour Scheduling Problem ([17]). This last problem offers a wide range of combinations which enables substantial improvements in the labor utilization. Consequently, many approaches have been proposed and tested (see [1] for an overview).

Loucks and Jacobs ([16]) present a heuristic approach to the dual problem of Tour Scheduling and Task Assignments involving workers who differ in their availabilities and qualifications. In this problem, two ranked objectives are considered: the first one is the minimization of the total man-hours of overstaffing, and the second one is the minimization of the sum of the squared differences between the number of tour hours scheduled and the number targeted for the workers. However, the approach requires to divide the horizon into one-hour periods, which is far too big for our own problem. More generally, to our knowledge, the Tour Scheduling Problem with one-minute periods has not been studied yet. This may be explained in three steps, as mentioned by [16]: first of all, working with one-minute periods means that the demand is known with a one-minute precision, which makes no sense if the demand comes from forecasting methods. Besides, managers would be under the obligation to make efforts so that employees respect their schedules. Finally, managers often prefer to keep some flexibility in schedules in order to cope with potential delays, and other uncertainties.

### **Fixed Job Scheduling Problem**

Given a set of jobs along with their fixed starting times and processing times, the Fixed Job Scheduling Problem (FJSP), also known as the Interval Scheduling Problem, consists of deciding whether or not to accept a job, knowing that chosen jobs have to be assigned to available resources (see [15] for an overview). The assignment part of our problem could consequently be seen as a FJSP, where every job has to be accepted. In this last case, deciding whether a feasible schedule exists is NP-Complete ([15]).

As pointed out in [15], the FJSP constitutes the core of a variety of applications, such as the well-known Crew Rostering Problem. For instance, in [9] the authors propose a mixed approach based on CP and OR techniques to solve a specific CRP. However, they consider only the rostering step, meaning the sequencing of the given duties, and not the scheduling step, meaning the generation of duties. Moreover, many CRPs integrate some constraints on the feasible sequences of tasks in order to take into account the geographical location of the tasks, which is not relevant in our case.

However, CRPs also consider the scheduling of ground station personnel, which is closer to our problem since the geographical location is absent. In [11] the authors present a decision support system designed for the aircraft maintenance department of KLM Royal Dutch Airlines. However the scheduling problem of KLM is different from our own problem in several points: first of all, ground station personnel operates in a four-shift system with fixed shifts. Besides, members of the same team are assigned to

the same shifts. In [4] the authors present an approach in two steps based on column generation and simulated annealing in order to schedule the work of the ground station personnel in airport. In order to solve this problem, they used a 15-minutes period interval on a weekly horizon, which is too coarse-grained for our problem. Moreover, they assume that shifts of the same tour are the same, which means that employees are assigned to the same shift during the whole week.

### Nurse Rostering Problem

The core of NRPs is to assign shifts to nurses over a scheduling period so that the resulting rosters respect a set of constraints and cover the demand. Most NRPs are NP-Complete [13] and real-world applications, due to their large set of specific constraints, are very challenging and hard to solve.

Our problem clearly shares some characteristics with the NRP as described in [6]: constraints **HC 1** and **HC 2** are common personnel constraints of the NRP, constraints **HC 3** to **HC 5** can be seen as coverage constraints and constraints **HC 6** to **HC 11** are common work regulation constraints of the NRP. However, the core of the problem is different since both the demand and the work assignment are different: in the NRP, the demand is given by a number of nurses for each skill category and for each demand period, whereas in our problem, it is given by a set of tasks with any possible starting and ending times. Moreover, in the NRP, the work assignment corresponds to a shift assignment with usually a very limited number of shifts (see for instance [7], [14] and [10]), whereas in our problem, the work assignment is given by a tasks assignment from which shifts have to be deduced. In addition to these differences, NRPs usually consider patterns to penalize or favour, such as consecutive night shifts, stand-alone day-off shift or complete weekends, which are not taken into account in our problem. Consequently, our problem is much closer to the Tour Scheduling Problem ([17]) and from the Fixed Job Scheduling Problem ([15]) than from the NRP.

The problem of designing schedules by taking into account skills, availabilities and work regulation constraints in order to cover personnel requirements has been widely studied in several business environments (NRP, TSP). In these problems, personnel requirements are usually given for a set of fixed time slots/shifts, whereas in our problem personnel requirements are given by a set of fixed tasks which cannot be preempted. Translating our personnel requirements into the classical time index representation would lead to allow preemption which is not possible. On the contrary, the problem of assigning fixed tasks to resources (FJSP) do not take into account some basic constraints related to work regulations, such as the minimal resting time between two worked days. Finally, the design of schedules for ground stationed personnel seems to be the closest problem in the related literature ([4]). However, in practice, additional constraints related to the internal organization of airports are often taken into account. Besides, even if the equity among workers may be an interesting objective in these kind of problems it is also often important to minimize iddle times in order to improve the productivity of workers, whereas in our problem it is required to let some iddle time so that nurses could work on their free administrative tasks.

On the whole, even if some related problems are relatively close to the one we consider, none of them allow to grasp its full complexity (the time granularity which drops to the minute, the lack of fixed shifts and the optimality criterion are good



Table 1: Data

Data	Definition
$\mathcal{N} = \{1, \dots, N\}$	Set of nurses
$\mathcal{D} = \{1, \dots, D\}$	Set of days
$\mathcal{T} = \{1, \dots, T\}$	Set of clinical tasks
$\mathcal{A} = \{T + 1, \dots, A\}$	Set of compulsory administrative tasks
$\forall n \in \mathcal{N}, \mathcal{A}_n \subset \mathcal{A}$	Set of tasks assigned to the nurse $n$
$\mathcal{O} = \{O_1, \dots, O_{\text{card}(\mathcal{O})}\}$	Set of sets of overlapping tasks
$\forall t \in \mathcal{T} \cup \mathcal{A}, s[t]$	Starting minute (over the week) of task $t$
$\forall t \in \mathcal{T} \cup \mathcal{A}, e[t]$	Ending minute (over the week) of task $t$
$\forall n \in \mathcal{N}, Wo[n]$	Targeted clinical working time over the week
$\forall n \in \mathcal{N}, Lo[n]$	Number of worked days since the previous day-off
$\forall n \in \mathcal{N}, U[n] = \{u_0, \dots, u_k\}$	Periods of unavailability for employee $n$

Table 2: Variables

Variables	Definition
$Tu \subset \mathcal{T}$	Set of unassigned tasks
$\forall n \in \mathcal{N}$	For every employee $n$
$Ta[n] \subset \mathcal{T} \cup \mathcal{A}$	Set of weekly assigned tasks
$Bs[n] \in \llbracket 0; 7980 \rrbracket$	Start of the weekly break
$Ww[n] \in \llbracket 0; 2880 \rrbracket$	Weekly working load
$Go[n] \in \llbracket -2880; 2880 \rrbracket$	Gap between $Ww[n]$ and $Wo[n]$
$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}$	For every employee $n$ and every day $d$
$Da[d][n] \subset \mathcal{T} \cup \mathcal{A}$	Set of daily assigned tasks
$At[d][n] \in \llbracket 0; 660 \rrbracket$	Daily attendance time
$Wd[d][n] \in \llbracket 0; 600 \rrbracket$	Daily working load
$Sh[d][n] \in \{\text{Off}; \text{Worked}\}$	Daily assigned shift
$Fi[d][n] \in \{\omega\} \cup \{s[t], t \in \mathcal{T} \cup \mathcal{A}\}$	Start of the daily working period
$La[d][n] \in \{\alpha\} \cup \{e[t], t \in \mathcal{T} \cup \mathcal{A}\}$	End of the daily working period

examples). Consequently, we propose in the following a dedicated method based on Constraint Programming. The use of CP is motivated by recent works which highlight its ability to tackle highly constrained problems with very specific objectives.

#### 4 Constraints Modeling

Data and variables notations related to our model are presented in Tables 1 and 2. The main idea of the model is to assign tasks to nurses using set variables. Since every task is fixed, the search of a solution amounts to finding a weekly set of tasks for each nurse: the vector  $Ta$  gives, for each nurse, the set of assigned tasks. However, most of the constraints deal with daily work instead of weekly work, which requires additional variables. Consequently, the matrix  $Da$  gives, for each nurse and for each day, the set of assigned tasks. In order to check constraints over shift sequences, the matrix  $Sh$ , gives for each nurse and day whether it is a working day or a day-off. Matrices  $Fi$  and  $La$  stand for the starting and ending time of each day of each nurse. Domains

of  $F_i$  (respectively  $L_a$ ) correspond to the starting (respectively ending) time of tasks. In order to deal with days off, domains of  $F_i$  (respectively  $L_a$ ) are completed with a constant  $\omega = 8 \times 24 \times 60$  (respectively  $\alpha = -24 \times 60$ ). The matrix  $At$  gives for each nurse and for each day the minimal attendance time (i.e. the attendance time resulting from clinical and compulsory administrative tasks). Finally, the vector  $Bs$  gives for each nurse the starting time of the weekly break.

Based on these variables, organizational and legal constraints could then be written. Some constraints refer directly to legal or organizational constraints whereas others simply ensure the consistency between variables. The first set will be referred to as *business constraints* whereas the second set will be denoted by *channeling constraints*. Finally, some legal constraints could be ensured from the creation of the variables, by reducing their domain of definition. This last set of constraints will be referred to as *preprocessing constraints* since once stated, they do not impact the solver anymore. In the following, *preprocessing constraints*, *business constraints* and finally *channeling constraints* are explained.

### Preprocessing constraints

The domains of  $Ta$  and  $Da$  are reduced during the creation of variables by comparing both the mastered skills of each employee with the required skills of each task and the starting and ending times of each task with the availabilities of each employee. Basically, a task  $t$  which requires the skill  $s$  will be removed from the domain of the variable  $Ta[n]$  if the nurse  $n$  does not master  $s$ . Besides, if the processing interval of  $t$ , which is given by  $[s[t]; e[t]]$  overlaps any periods of unavailability of employee  $n$ , which are given by  $U[n]$ , then  $t$  will also be removed from  $Ta[n]$ . The same process holds for  $Da$ . Moreover, the domain of the variables of  $Da$  could be even more reduced by taking into account the day of the week: tasks which are fixed on Wednesday, for instance, could not be performed on Monday, and so on. More precisely, a working period  $d$  could gather every task whose starting time belongs to the interval:  $[60 \times (6 + 24 \times d); 60 \times (6 + 24 \times (d + 1))]$ , which corresponds to a period of 24 hours starting every day at 6 am. Consequently, employees who start working around 9 pm a day could finish their work at 6 am the following day. Thus, constraints **HC 1** and **HC 2** are verified from the creation of the problem, without any cost (i.e. the solver will not have to check these constraints during the resolution). The bounds of working period intervals have been fixed to correspond to the earliest starting time and the latest ending time applied by the company. Domains of  $Wd$  and  $Ww$  variables are also bounded from their creation, so that employees could neither be assigned to more than 10 hours of work over a day (**HC 4**), nor to more than 48 hours of work over the week (**HC 5**).

### Business constraints

Constraints (1a) to (1c) aim at assigning the exact number of required employees to each task (**HC 3**). More precisely, constraint (1a) aims at assigning at least one employee to every task. Constraint (1b) ensures that tasks which have to be assigned to one employee, are not assigned to several employees. Constraint (1c) ensures that employees do not perform the same task several days. Constraint (2) ensures that the assignment of compulsory administrative tasks is respected (**HC 4**). Constraint (3) prevents employees from starting a new task before finishing the previous one (**HC 5**). Constraints (4) and (5) aim respectively at respecting the daily maximum attendance time (**HC 8**) and the minimum resting time between two working days (**HC 9**). The

constraint (6a) aim at building a weekly break of at least 35 hours (**HC 10**) by ensuring that no work is assigned to employees on an interval of 35 hours. Constraint (7) ensures that the maximal number of consecutive worked days does not exceed 6 days (**HC 11**).

$$Tu \cup \bigcup_{n \in \mathcal{N}} Ta[n] = \mathcal{T} \cup \mathcal{A} \quad (1a)$$

$$\forall (n_1, n_2) \in \mathcal{N}^2 \mid n_1 \neq n_2, \quad Ta[n_1] \cap Ta[n_2] = \mathcal{A}_{n_1} \cap \mathcal{A}_{n_2} \quad (1b)$$

$$\forall n \in \mathcal{N}, \forall (d_1, d_2) \in \mathcal{D}^2 \mid d_1 \neq d_2, \quad Da[d_1][n] \cap Da[d_2][n] = \emptyset \quad (1c)$$

$$\forall n \in \mathcal{N}, \quad \mathcal{A}_n \subset Ta[n] \quad (2)$$

$$\forall d \in \mathcal{D}, \forall n \in \mathcal{N}, \forall O_i \in \mathcal{O}, \quad \text{card}(O_i \cap Da[d][n]) \leq 1 \quad (3)$$

$$\forall d \in \mathcal{D}, \forall n \in \mathcal{N}, \quad At[d][n] \leq 60 \times 11 \quad (4)$$

$$\forall d \in \mathcal{D} \setminus \{D\}, \forall n \in \mathcal{N}, \quad Fi[d+1][n] - La[d][n] \geq 60 \times 11 \quad (5)$$

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad (Fi[d][n] \geq Bs[n] + 35 \times 60) \vee (La[d][n] \leq Bs[n]) \quad (6a)$$

$$\forall n \in \mathcal{N}, \quad \text{card}\{d \in \llbracket 0; 6 - Lo[n] \rrbracket \mid Sh[d][n] = \text{Off}\} \geq 1 \quad (7)$$

### Channeling constraints

Constraint (8) ensures that the daily and the weekly assignments of each employee are consistent. Constraints (9) and (10) ensure that the weekly working load and the daily working load of each employee correspond to the task assignment. Constraint (11) ensures that the matrix  $Sh$  is consistent with the matrix  $Da$ . Constraints (12a) and (12b) aim at finding the beginning and the end of the working days of each employee when daily assigned sets are not empty. Empty sets, which refer to days-off, are handled by constraints (13a) and (13b). This last point might need some deeper explanations: first of all, empty sets have to get a starting and ending times because of the construction of  $Fi$  and  $La$ . In order to respect constraints **HC 8** and **HC 9**,  $Fi$  and  $La$  are assigned respectively to the end and the beginning of the week. This setting is also consistent with constraint (6a). Constraint (14) aims at computing the daily attendance time of each employee: for non-empty sets, the attendance time corresponds to the difference between the end and the start of the corresponding day, for empty sets, the attendance time corresponds to 0. Finally, constraints (15a) to (15d) calculate the objective value. In constraint (15a) we subtract compulsory administrative time from the weekly workload, in order to keep only the clinical workload.

$$\forall n \in \mathcal{N}, \quad \bigcup_{d \in \mathcal{D}} Da[d][n] = Ta[n] \quad (8)$$

$$\forall n \in \mathcal{N}, \quad Ww[n] = \sum_{t \in Ta[n]} e[t] - s[t] \quad (9)$$

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad Wd[d][n] = \sum_{t \in Da[d][n]} e[t] - s[t] \quad (10)$$

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad \text{card}(Da[d][n]) > 0 \Leftrightarrow Sh[d][n] = \text{Worked} \quad (11)$$

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad \text{card}(Da[d][n]) > 0 \Rightarrow Fi[d][n] = \min_{t \in Da[d][n]} s[t] \quad (12a)$$

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad \text{card}(Da[d][n]) > 0 \Rightarrow La[d][n] = \max_{t \in Da[d][n]} e[t] \quad (12b)$$

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad \text{card}(Da[d][n]) = 0 \Leftrightarrow Fi[d][n] = \omega \quad (13a)$$

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad \text{card}(Da[d][n]) = 0 \Leftrightarrow La[d][n] = \alpha \quad (13b)$$

$$\forall d \in \mathcal{D}, \forall n \in \mathcal{N}, \quad At[d][n] = \max(La[d][n] - Fi[d][n], 0) \quad (14)$$

$$\forall n \in \mathcal{N}, \quad Go[n] = (Ww[n] - \sum_{t \in \mathcal{A}_n} e[t] - s[t]) - Wo[n] \quad (15a)$$

$$\forall n \in \mathcal{N}, \quad Gmin \leq Go[n] \quad (15b)$$

$$\forall n \in \mathcal{N}, \quad Gmax \geq Go[n] \quad (15c)$$

$$Obj = 10\,000 \times \text{card}(Tu) + Gmax - Gmin \quad (15d)$$

### Choice of the model

This model is oriented from a task assignment point of view, meaning that our concern is to assign tasks to nurses. Based on this assignment, additional information (such as starting times, ending times, daily load, weekly load, etc...) could then be easily deduced. Another way of modeling the problem is to assign one nurse to every task. However, a lot of constraints which are easily written with task sets, seem more difficult to write with such a model. Consequently, we decided to focus on the task assignment instead of the nurse assignment. Another common way of modeling personnel scheduling problems with constraint programming is based on the use of a general matrix giving the activity of each nurse (on lines) for each time period (on rows). For instance, The Nurse Rostering Problem has been stated and solved by constraint programming in such a way ([21]). However, this kind of approach requires to consider identical time periods, or slots. Since the time granularity of our problem drops to the minute over a planning horizon of a week, the use of identical time periods would lead either to an explosion of the number of variables or to some approximations on the duration of the tasks, which is not desirable. Consequently, our approach, based on task assignment seems to handle the complexity of the problem in a more promising way.

## 5 Variable and value strategies

When using constraint programming in order to build a good solution, it is very important to implement a dedicated search strategy. Basically, CP aims at finding a satisfying

solution, not the optimal one. However CP could still be used to find the optimum by solving the given problem in an iterative way: at each iteration, the cost of the solution is constrained to be less (when minimizing) than the cost of the last solution.

The aim of a dedicated search strategy is to avoid backtracking time in the huge space of unfeasible and unsatisfying solutions, by using the specificities of the problem. Once the filtering process is over, if every variable is not yet instantiated, the solver has to branch on a variable. The variable ordering strategy (VarOS) aims at choosing the most promising variable in order to branch on this variable, and the value ordering strategy (ValOS) aims at starting the branching with the most promising value. This value is chosen from the domain of the chosen variable. In our model, set variables are represented by two sets: the kernel and the envelope. The first one represents the set of values which belong to every solution whereas the second one refers to the set of values which belong to at least one solution. Consequently, the kernel of a set variable is a subset of its envelope. The chosen value must belong to the envelope, but not to the kernel. This set of possible values is called the open domain.

Concerning set variables, the *default* strategy select the variable with the smallest open domain, and among this domain, it selects the smallest value. However, this strategy do not use the specificities of the problem. Consequently, two VarOS and five ValOS are proposed and discussed in the following.

**VarOS** Among the variables of the vector  $Ta$ :

- Choose the variable corresponding to the nurse whose numerical difference between the assigned clinical working time and the targeted clinical working time is the smallest. The idea of this VarOS, which will be referred to as *LW (Less Working)*, is to start assigning tasks to the less working nurse, as soon as possible, in order to improve the solution.
- Find the variable corresponding to the nurse whose weekly working load is the highest among those which are under a fixed limit, controlled by a parameter  $\bar{l}$ , expressed in minutes. More precisely, among the variables respecting the following inequality:

$$W_w[n] - W_o[n] \leq \bar{l}$$

choose the variable which maximizes  $(W_w[n] - W_o[n])$ . If such a variable does not exist, choose a variable randomly. This strategy, which will be referred to as *MW (Most Working)*, could be used in two different ways depending on the value of  $\bar{l}$ . For instance, setting  $\bar{l}$  to a positive value allow the solver to keep assigning tasks to nurses who are already above their targeted clinical load, in order to keep room for further assignments and consequently, avoid dead ends. Consequently, this will not lead to well-balanced solutions, but the idea is to evaluate the number of feasible solutions which could be found by this way. On the contrary, setting  $\bar{l}$  to a negative value aim at ensuring a minimum working load for each nurse. Even if setting  $\bar{l}$  to a negative value is also a way of finding good solutions, there is an important difference between *LW* and *MW*: *LW* tries to improve the solution at every node whereas *MW* allow the solver to deteriorate the solution, hoping to improve it in

the end.

**ValOS** Among the *open domain* (referred to as  $O(Ta[n_c])$ ) of the chosen variable:

- Find the task which produces the highest increase of work density. The work density of a day corresponds to the working load of this day divided by the attendance time of the day. If the highest increase of work density is higher than a given limit controlled by a parameter  $\underline{\delta}$ , then choose the corresponding task, else choose the first possible task. More formally, for each possible task  $t$ , we can compute the new daily working load  $W_d^{t+}[d][n_c]$  and the new attendance time  $At^{t+}[d][n_c]$  resulting from the addition of  $t$  to the corresponding working day of  $n_c$ . We refer to the increase of density produced by  $t$  as  $\delta[t]$ :

$$\delta[t] = \frac{W_d[d][n_c]}{At[d][n_c]} - \frac{W_d^{t+}[d][n_c]}{At^{t+}[d][n_c]} \in [-1; 1]$$

Among tasks whose corresponding  $\delta[t]$  is above  $\underline{\delta}$ , choose the task  $t$  which maximizes  $\delta[t]$ . If such a task does not exist, choose the first possible task. The idea of this ValOS, which will be referred to as *ID (Increase Density)*, is to produce compact schedules, in order to avoid wasting time.

- Choose the task which belongs to the biggest set of overlapping tasks. The idea of this ValOS, which will be referred to as *BO (Biggest Overlap)*, is to assign as soon as possible tasks which are fixed on activity peaks. Since employees cannot work simultaneously on overlapping tasks, choosing such a task may also be interesting because of the filtering process which may lead to many deductions (the employee could not anymore perform the other overlapping tasks).
- Choose the task which can be performed by the smallest number of nurses. The idea of this ValOS, which will be referred to as *LN (Lack of Nurses)*, is to avoid backtracking procedure by avoiding dead ends.
- Find the task with the biggest duration. If this duration is higher than a given limit controlled by a parameter  $\underline{p}$ , then choose this task, else choose the first possible task. More precisely, among the tasks which respect the following inequality:

$$e[t] - s[t] \geq \underline{p}$$

choose the longest task. The idea of this ValOS, which will be referred to as *BT (Biggest Task)*, is to assign biggest tasks as soon as possible.

### Mixed Strategy

Based on these simple heuristics we implement a more complex one, which will be referred to as *Mx (Mixed)*. The idea of this ValOS, which is presented in detail in Algorithm 1, is to use the previously described heuristics in a combined way. More precisely, the global idea of *Mx* is to avoid dead ends by using *Lack of Nurses* and *Biggest Overlap*, then, if possible choose an interesting task by using *Biggest Task* and *Increase Density*, otherwise, choose a task which has a big impact on the search by using *Biggest Overlap*.

In the first step of *Mx* (lines 1 to 5), we compare the number of employees available for a task  $t$  with the number of tasks overlapping  $t$ . The idea is to avoid the assignment of employees available for  $t$  to other tasks. In the second step (lines 6 to 10), we compare

the highest processing time with the parameter  $p$ . The idea of this threshold is to focus on the duration of tasks only when it is a highly distinguishing criterion. In the third step (lines 11 to 15), we compare the highest increase of density with the parameter  $\underline{\delta}$ . Again, this comparison aims to avoid making a decision on a hardly distinguishing criterion. Finally (lines 16 to 18), we choose the task which corresponds to the highest activity peak.

---

**Algorithm 1** Mx ValOS: combine simple heuristics to choose the most promising task

---

**Require:**

$p \in \mathbb{N}$ ,  $\underline{\delta} \in [-1; 1]$   
 $\forall t \in O(Ta[n_c])$ ,  $available[t]$ : the number of available employees for task  $t$   
 $\forall t \in O(Ta[n_c])$ ,  $overlapping[t]$ : the number of tasks overlapping task  $t$   
 $findIndex(value, vector)$ : returns the index of  $value$  in the given  $vector$ .

```

1: for all  $t \in O(Ta[n_c])$  do
2:   if  $available[t] \leq overlapping[t]$  then
3:     return  $t$ 
4:   end if
5: end for
6:  $maxP \leftarrow \max_{t \in O(Ta[n_c])} e[t] - s[t]$ 
7:  $tMaxP \leftarrow findIndex(maxP, p)$ 
8: if  $maxP \geq p$  then
9:   return  $tMaxP$ 
10: end if
11:  $maxD \leftarrow \max_{t \in O(Ta[n_c])} \delta[t]$ 
12:  $tMaxD \leftarrow findIndex(maxD, \underline{\delta})$ 
13: if  $maxD \geq \underline{\delta}$  then
14:   return  $tMaxD$ 
15: end if
16:  $maxO \leftarrow \max_{t \in O(Ta[n_c])} overlapping[t]$ 
17:  $tMaxO \leftarrow findIndex(maxO, overlapping)$ 
18: return  $tMaxO$ 

```

---

## 6 Experimental results

We implemented our model with Choco, a Java Constraint Satisfaction Problem Solver [20]. Each instance has been runned on an Intel Core i3 (3.06 GHz) with a time limit of 5 minutes under *default* (see section 5) and dedicated ordering strategies, in optimization. The time limit has been fixed to 5 minutes because the company would like to use the method as a simulation tool, which requires great responsiveness.

### 6.1 Instances generation

In order to test our model, we have generated 720 instances gathered in 24 sets of 30 instances. Each set of instances corresponds to a specific combination of three parameters: the number of tasks, the kind of skills required and the tightness of the workload compared to the work capacity. The number of tasks ranges from 100 to 400 which

allows us to check the behavior of the method for normal activity (200/300 tasks) and extreme cases (100/400 tasks). Moreover we have generated two kinds of skills requirements: the first one considers only common skills, meaning that each skill is mastered by most of the nurses whereas the second one considers also rare skills, meaning that some skills are mastered by only a few nurses. For each instance the targeted clinical workload of each employee is generated in order to fit to the global workload (+/- 5h). The tightness is defined as the average clinical working time of nurses: a usual tightness does not exceed 700 minutes. The first set of instances has a tightness of 600 minutes, which corresponds to a usual workload. The second one has a tightness of 800 minutes which corresponds to a big workload. The tightness of the last set of instances amounts to 1000 minutes which aims at testing the limit of the model. Finally, the task distribution and profile are also based on realistic data: tasks are distributed all along the week, with a density peak around 8 a.m. and three kinds of tasks, with specific probabilities of occurrence:

1. small tasks ranging from 5 to 15 minutes, with a probability of occurrence of 5%.
2. medium tasks with a processing time close to one hour, with a probability of occurrence of 65%.
3. big tasks ranging from 2 to 5 hours, with a probability of occurrence of 30%.

Then, a simple procedure computes the required number of workers along with their personal data (skills, availabilities, etc...). It ensures that the tightness of the instance is respected but it does not ensure the feasibility of the instance. Consequently some instances do not admit a complete assignment. In the following, we define the size of an instance as its number of tasks.

## 6.2 Parameters design

Parameter  $\bar{l}$  (in minutes) has been tested with values: -180, 0 and 180. Increasing  $\bar{l}$  leads some ValOS, such as BO, to a higher number of complete assignments and a smaller number of unassigned tasks. However this improvement is obtained at the expense of the mean equity value. Moreover, some strategies such as *BT* and *Mx* do not profit from this increase of  $\bar{l}$ , on the contrary, it leads only to worsen the equity.

Parameters  $\underline{p}$  and  $\underline{\delta}$  have been tested separately within the *BT* (*Biggest Task*) and the *ID* (*Increase Density*) ValOS respectively.

Parameter  $\underline{\delta}$  has been tested from -1 to 0 with a range of 0.1. We did not try to set  $\underline{\delta}$  to strictly positive values, because it seems unlikely to be possible to use this strategy during the whole search. Results show no significant differences for these various settings, which means that making a big increase of density is not so important compared to completing working days (whether it is by increasing or decreasing the density). In order to get a well balanced *Mx* strategy, we set  $\underline{\delta}$  to 0, but when used inside *ID* we set it to -1.

Parameter  $\underline{p}$  (in minutes) has been tested with values 5, 30, 50, 120, 180 and 240 which enables us to consider either every task or only subsets of tasks. Increasing  $\underline{p}$  leads to a higher number of complete assignments and a smaller number of unassigned tasks, but it worsen the equity. The best solutions (considering equity) are obtained with values 5 and 30. Above 30 the equity value starts to decrease. Consequently, we set  $\underline{p}$  to 30. By doing this, we put aside small tasks whose durations range from 5 to



30 minutes, but when used inside *BT* we set it to 5.

### 6.3 Results analysis

In order to evaluate our results, which are presented in detail in Table 3, we used 4 indicators:

1. "Complete": the number of solutions with a complete assignment.
2. "Equity": the mean equity value of best solutions, in minutes (over complete assignments only).
3. "Left": the average number of unassigned tasks.
4. "Time": the mean computation time of best solutions, in seconds (over complete and incomplete assignments).

Table 3: Results (time limit: 5 min)

Strategy	Indicator	Size				
		100	200	300	400	All
LW BO	Complete	27/180	43/180	45/180	52/180	167/720
	Equity	304	301	295	293	297
	Left	6	8	9	11	8
	Time	44	23	23	41	32
LW BT( $\underline{p} = 5$ )	Complete	30/180	43/180	64/180	61/180	198/720
	Equity	49	41	39	34	39
	Left	5	6	7	6	6
	Time	22	26	19	36	26
LW ID( $\underline{\delta} = -1$ )	Complete	57/180	89/180	104/180	108/180	358/720
	Equity	248	268	290	297	280
	Left	4	5	3	3	4
	Time	60	27	32	39	37
LW LN	Complete	54/180	75/180	90/180	93/180	312/720
	Equity	227	261	271	277	263
	Left	4	5	5	5	5
	Time	70	30	31	36	39
LW Mx( $\underline{p} = 30, \underline{\delta} = 0$ )	Complete	33/180	50/180	64/180	72/180	219/720
	Equity	42	43	40	36	40
	Left	5	6	6	6	6
	Time	35	5	16	37	23
LW All	Complete	71/180	101/180	116/180	124/180	412/720
	Equity	119	146	121	117	126
	Left	3	3	2	2	2
	Time	109	53	50	53	66
MW( $\underline{i} = -180$ ) BO	Complete	43/180	56/180	60/180	64/180	223/720
	Equity	389	471	500	540	483

Strategy	Indicator	Size				
		100	200	300	400	All
	Left Time	5 73	8 40	7 54	8 95	7 66
MW( $\bar{l} = -180$ ) BT( $\underline{p} = 5$ )	Complete	41/180	60/180	70/180	66/180	237/720
	Equity	286	363	390	414	372
	Left	5	7	5	5	6
	Time	96	58	59	78	71
MW( $\bar{l} = -180$ ) ID( $\underline{\delta} = -1$ )	Complete	79/180	102/180	126/180	125/180	432/720
	Equity	354	439	514	544	476
	Left	4	5	3	3	4
	Time	75	74	73	106	83
MW( $\bar{l} = -180$ ) LN	Complete	68/180	82/180	96/180	112/180	358/720
	Equity	349	477	551	579	505
	Left	4	5	4	4	4
	Time	101	89	94	100	96
MW( $\bar{l} = -180$ ) Mx( $\underline{p} = 30, \underline{\delta} = 0$ )	Complete	45/180	63/180	75/180	77/180	260/720
	Equity	312	369	375	419	376
	Left	5	6	5	5	5
	Time	66	69	77	75	73
MW( $\bar{l} = -180$ ) All	Complete	92/180	109/180	135/180	144/180	480/720
	Equity	283	365	414	458	380
	Left	3	3	2	2	2
	Time	146	129	127	147	137
MW( $\bar{l} = 180$ ) BO	Complete	59/180	71/180	78/180	81/180	289/720
	Equity	802	1056	1113	1185	1056
	Left	4	6	6	7	6
	Time	151	109	148	142	138
MW( $\bar{l} = 180$ ) BT( $\underline{p} = 5$ )	Complete	63/180	85/180	96/180	97/180	341/720
	Equity	903	1102	1217	1284	1150
	Left	5	6	5	4	5
	Time	146	138	134	140	139
MW( $\bar{l} = 180$ ) ID( $\underline{\delta} = -1$ )	Complete	91/180	108/180	128/180	128/180	455/720
	Equity	783	1003	1132	1213	1054
	Left	3	4	3	3	3
	Time	141	143	135	148	142
MW( $\bar{l} = 180$ ) LN	Complete	90/180	108/180	127/180	144/180	469/720
	Equity	763	989	1122	1190	1043
	Left	4	4	3	3	4
	Time	168	155	139	162	156
MW( $\bar{l} = 180$ ) Mx( $\underline{p} = 30, \underline{\delta} = 0$ )	Complete	69/180	88/180	101/180	107/180	365/720
	Equity	908	1104	1212	1282	1149
	Left	5	6	5	4	5
	Time	140	143	139	141	141
MW( $\bar{l} = 180$ ) All	Complete	103/180	123/180	139/180	152/180	517/720
	Equity	732	952	1063	1152	975

Strategy	Indicator	Size				
		100	200	300	400	All
	Left	2	2	1	2	2
	Time	222	226	223	226	224
All	Complete	106/180	127/180	142/180	153/180	528/720
All	Equity	205	260	210	238	228
	Left	2	2	1	1	2
	Time	256	246	244	251	249

Table 3 gives for each size the value of the various indicators depending on the branching strategy. Results obtained with  $\bar{l} = 0$  are not presented because they stand between those obtained with  $\bar{l} = -180$  and  $\bar{l} = 180$ . The column "All" gives for each configuration the sum of the indicator "Complete" along with the mean values of indicators "Equity", "Left" and "Time". The line *LW-All* (respectively *MW-All*) gives the results which can be found by using in parallel each ValOS with *LW* (respectively *MW*).

### Results regarding to feasibility

The *default* strategy gives similar results on each size: it finds around 70 instances with a complete assignment with a mean equity of 1600 minutes in 90 seconds. Compared to the *default* strategy, dedicated strategies find more solutions, especially with the *MW* VarOS, which illustrates the importance of using dedicated strategies.

Generally speaking, the number of solutions is bigger with the *MW* VarOS than with the *LW* strategy, which is coherent with the idea of these strategies. The *ID* ValOS gives the highest number of solutions for both VarOS, which means that building compact schedules is a good lead to get a feasible solution. On the contrary, *BO* gives few solutions for both strategies. This comes from the data specificities: the activity peak turns around 8 a.m. every day, consequently, by systematically choosing these tasks over the others we set to many shifts around the same time slot, which makes the assignment of night tasks much more difficult. The average number of unassigned tasks turns around 5 which is quite small.

On the whole, small instances turned out to be more difficult than bigger instances, which comes from the variations of the number of employees: instances with 100 tasks which corresponds to a small industrial activity have a smaller number of available workers than instances with 400 tasks which corresponds to a big activity. On the whole, tested ValOS perform differently on each instance, which means that they can complete one another, at least to some extent. Consequently, a practical way of finding solutions is to solve the problem in series with various ValOS, which is highlighted by the lines *LW-All*, *MW-All* and *All-All*. For instance, using each ValOS in parallel with *LW* increases the number of found solutions by 54. Another way to make up for the lack of solutions is to combine simple heuristics, as we do with *Mx*.

Instances with rare skills are harder to solve than those with only common skills. On average the relative difference of the number of complete assignments between these two sets turns around 10 to 20%. However, the average number of unassigned tasks does not change between these two sets. Instances with a high tightness are much

more difficult to solve than those with a common tightness. On the whole, 96%, 85% and 39% of the instances with a respective tightness of 600, 800 and 1000 minutes are solved with a complete assignment. This means that the method encounters difficulties to cope with heavy workload. However the number of unassigned tasks do not increase much.

### Results regarding to equity

On average, between first solutions and best solutions, the improvement of the objective is not very big, which may be explained by the choice of the method: CP aims more at finding a solution rather than improving a solution. Consequently it is very important to find a good solution from the beginning, especially for the biggest instances. *LW-BT* finds the best results regarding to the equity, but only on a very small subset of instances. The *LW-Mx* strategy finds more solutions than the *LW-BT* strategy, and their value is relatively close to the best known value. Consequently, this heuristic is a successful combination of the more simple heuristics.

### 6.4 Operational point of view

On the whole, the method is quite fast: the first solution is found within a few seconds and the best solution is found in less than 3 minutes, which fits the requirements of the company. Given that some instances are not feasible without externals, the method finds a relatively good number of instances with a complete assignment and the number of unassigned tasks is very small. Depending on the running configuration, the number of complete assignments along with the value of the equity may be very different: *LW-BT* and *LW-Mx* strategies perform very well on equity but they have some difficulties to find complete assignments. On the contrary, *LW-ID* and *LW-LN* strategies perform relatively well on the assignment but not so well regarding to the equity.

## 7 Concluding remarks

We have presented a CP model along with several branching strategies in order to solve a real-world problem which shares similarities both with the Tour Scheduling Problem and the Fixed Job Scheduling Problem. For the sake of clarity we did not mention the problem into its full complexity, but this model could be extended in order to deal with additional legal constraints such as break constraints but also more complex equity objectives such as the distribution of night or weekend shifts which are also important for the company. This would lead to the interesting problem of defining a good solution when facing multiple objectives. This approach deals simultaneously with the task assignment problem and the design of personnel scheduling. We intend to compare this approach with a sequential one which deals with the design of personnel schedules before the task assignment problem.

We believe that our model could be improved in two ways. The first one is to work on the branching strategies which clearly have a big impact on results. For instance, it may be interesting to design a more complex variable ordering strategy, able to mediate between the need to improve a solution and the need to keep room to avoid

dead ends. The second lead is to strengthen the filtering process by using redundant constraints: set variables are a powerful tool which makes constraints easy to write, but they suffer from a weak filtering. Consequently, adding integer variables and channeling constraints may increase the filtering and therefore the performances of the model.

Our results show that branching strategies can successfully complete one another by being used, either in series or in a combined way. A more thorough study of the impact of parameters would allow to get sharpened conclusions, but the main conclusions remain: many results are good enough to be used by the company whereas others need to be improved. Since the model encounters some difficulties to improve a solution, it may be interesting to use another method such as a Large Neighbourhood Search ([19]), as a following of our method. Since the number of unassigned tasks is on average very small, it may be much more efficient to explore some targeted neighbourhood rather than following the basic tree exploration in order to find complete assignments. In order to evaluate in a better way the quality of our approach, future work may also focus on the search of optimal solutions and lower bounds regarding to the equity and the number of externals required to perform each tasks. However our optimal criterion makes this task difficult. Working on the sequential approach (shifts are fixed) makes this task a bit easier. In this context, we proposed two lower bounds for the equity value ([18]). However they do not correspond to the lower bound for the general problem studied in this paper and hence need to be adapted which seems very challenging.

## References

1. Alfares, H.K.: Survey, Categorization, and Comparison of Recent Tour Scheduling Literature. *Annals of Operations Research* 127, 145–175 (2004)
2. Bechtold, S.E., Jacobs, L.W.: Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science* 36(11) (1990)
3. Brucker, P., Qu, R., Burke, E.K.: Personnel scheduling: Models and complexity. *European Journal of Operational Research* 210(3), 467–473 (2011)
4. Brusco, M.J., Jacobs, L.W., Bongiorno, R.J., Lyons, D.V., Tang, B.: Improving personnel scheduling at airline stations. *Operations Research* 43(5), 741–751 (1995)
5. Burke, E.K., Cowling, P.: A Memetic Approach to the Nurse Rostering Problem. *Applied Intelligence* 15(3), 199–214 (2001)
6. Burke, E.K., De Causmaecker, P., Berghe, G.V., Landeghem, H.V.: The state of the art of nurse rostering. *Journal of Scheduling* 7, 441–499 (2004)
7. Burke, E.K., Li, J., Qu, R.: A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research* 203(2), 484–493 (2010)
8. Cai, X., Li, K.N.: A genetic algorithm for scheduling staff of mixed skills under multi-criteria. *European Journal Of Operational Research* 125, 359–369 (2000)
9. Caprara, A., Focacci, F., Lamma, E., Mello, P., Milano, M., Toth, P.: Integrating Constraint Logic Programming and Operations Research Techniques for the Crew Rostering Problem. *Software Practice and Experience* (28), 49–76 (1998)
10. Cheng, B.M.W., Lee, J.H.M., Wu, J.C.K.: Speeding Up Constraint Propagation By Redundant Modeling. In: 2nd Int. Conf. on Principles and Practice of Constraint Programming (1996)
11. Dijkstra, M.C., Kroon, L.G., Salomon, M., Van Nunen, J.A.E.E., van Wassenhove, L.N.: Planning the Size and Organization of KLM's Aircraft Maintenance Personnel. *Interfaces* 24(6), 47–58 (1994)

12. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153(1), 3–27 (2004)
13. Garey, M.R., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman (1979)
14. Jaumard, B., Semet, F., Vovor, T.: A generalized linear programming model for nurse scheduling. *European Journal Of Operational Research* 2217(97) (1998)
15. Kolen, A.W.J., Lenstra, J.K., Papadimitriou, C.H., Spieksma, F.C.R.: Interval Scheduling: A Survey. *Naval Research Logistics* 54, 530–543 (2007)
16. Loucks, J.S., Jacobs, R.F.: Tour Scheduling and Task Assignment of a Heterogeneous Work Force: A Heuristic Approach. *Decision Sciences* 22(4), 719–738 (1991)
17. Mabert, V.A., Watts, C.A.: A Simulation Analysis of Tour-Shift Construction Procedures. *Management Science* 28(5), 520–532 (1982)
18. Prot, D., Lapègue, T., Bellenguez-Morineau: Lower bounds for a fixed job scheduling problem with an equity objective function. In: 25th European Conference on Operational Reserach (2012)
19. Shaw, P.: Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: *Principles and Practice of Constraint Programming (CP'98)*, volume 1520 of LNCS. pp. 417–431 (1998)
20. Team, C.: *choco: an open source java constraint programming library*. Research report, Ecole des Mines de Nantes (2010)
21. Weil, G., Heus, K., Patrice, F., Poujade, M.: Constraint Programming for Nurse Scheduling. *Engineering in Medicine and Biology Magazine, IEEE* 14(4), 417–422 (1995)

---

## Fairness in Academic Course Timetabling

Moritz Mühlenthaler · Rolf Wanka

**Abstract** We consider the problem of creating *fair* course timetables in the setting of a university. Our motivation is to improve the overall satisfaction of individuals concerned (students, teachers, etc.) by providing a fair timetable to them. The central idea is that undesirable arrangements in the course timetable, i. e., violations of soft constraints, should be distributed in a fair way among the individuals. We propose two formulations for the fair course timetabling problem that are based on max-min fairness and Jain's fairness index, respectively. Furthermore, we present and experimentally evaluate an optimization algorithm based on simulated annealing for solving max-min fair course timetabling problems. The new contribution is concerned with measuring the energy difference between two timetables, i. e., how much worse a timetable is compared to another timetable with respect to max-min fairness. We introduce three different energy difference measures and evaluate their impact on the overall algorithm performance. The second proposed problem formulation focuses on the tradeoff between fairness and the total amount of soft constraint violations. Our experimental evaluation shows that the known best solutions to the ITC2007 curriculum-based course timetabling instances are quite fair with respect to Jain's fairness index. However, the experiments also show that the fairness can be improved further for only a rather small increase in the total amount of soft constraint violations.

**Keywords** Curriculum-based Course Timetabling, Max-Min Fairness, Fairness Index

### 1 Introduction

We consider the problem of creating fair course timetables in the setting of a university. In academic timetabling, the courses need to be assigned to a limited number of resources (rooms and timeslots) such that certain constraints are satisfied. There are typically two kinds of constraints called hard and soft constraints. The hard constraints are basic requirements, so a timetable which does not satisfy all hard constraints is considered useless. The

---

Research funded in parts by the School of Engineering of the University of Erlangen-Nuremberg.

Moritz Mühlenthaler, Rolf Wanka  
Department of Computer Science, University of Erlangen-Nuremberg, Germany  
E-mail: {moritz.muehlenthaler, rwanka}@cs.fau.de

soft constraints characterize certain undesirable properties of a timetable for students and teachers, as well as for abstract entities such as courses and curricula. The quality of a timetable is determined by the extent to which soft constraints are violated: the fewer soft constraint violations, the better. The usual approach is to use an objective function which penalizes the soft constraint violations so the goal is to find a timetable with a minimal total penalty. Situations may arise however, in which a large part of penalty hits only a small group of individuals, who would thus receive a poor timetable in comparison to others. In other words, a timetable may be unfair due to an unequal distribution of penalty.

Fairness and inequality in the distribution of resources are a major concern for instance in economics [15,36] and computer networks [4–6,17,18,20,31,35]. In the area of operations research, fairness criteria have been applied for example to the aircraft landing problem [37]. To the best of our knowledge, no previous paper on academic timetabling addresses fairness explicitly. In this paper, we investigate two approaches that avoid unfair distributions of penalty in the context of academic course timetabling. The first approach uses a purely qualitative measure of fairness, i. e., given two timetables the fairness measure determines which of the two is better. In contrast, the second approach is based on a quantitative fairness measure, which represents the fairness of a timetable as a number between zero and one.

The first approach considers groups of students ranked by the quality of the course timetable from the students' perspective. The goal is to improve the student satisfaction by imposing the following fairness conditions: The courses should be assigned to rooms and timeslots such that the worst course schedule for any of the students is as good as possible with respect to the various soft constraints. Under this condition, the second-worst course schedule for any student should be as good as possible, and so forth. This fairness concept is called lexicographic max-min fairness. For the sake of succinctness, we will refer to this fairness concept just as max-min fairness. In the literature, max-min fairness has for example been applied to network bandwidth allocation problems [31,35]. In this work, we propose the MMF-CB-CTT problem model, a max-min fair variant of the popular curriculum-based course timetabling (CB-CTT) problem formulation from [13]. We further propose MAXMINFAIR\_SA, an optimization algorithm based on simulated annealing (SA), for solving max-min fair optimization problems. Although our evaluation of MAXMINFAIR\_SA focuses on MMF-CB-CTT problems, the algorithm can be tailored to other max-min fair problems by choosing an appropriate neighborhood exploration mechanism and a suitable evaluation function. A delicate part of the algorithm is the energy difference function, which quantifies how much worse one solution is compared to another solution. We propose three different energy difference functions and evaluate their impact on the performance of MAXMINFAIR\_SA on the 21 standard instances from [13].

The fairness conditions imposed by max-min fairness are rather strict in the sense that no tradeoff arises between fairness and total penalty. When creating course timetables for a university, however, it may be desirable to pick a timetable from a number of solutions with varying tradeoffs between fairness and total penalty. Our second proposed approach offers this flexibility. The approach is based on a bi-criteria problem formulation which includes fairness as an option, but does not enforce it like max-min fair optimization. In this problem formulation, referred to as JFI-CB-CTT, we use Jain's fairness index, an inequality measure proposed by Jain et al. in [17]. The fairness index allows us to quantify the fairness of a timetable explicitly. Please note that since max-min fairness is a purely qualitative fairness measure, it is not applicable in this setting. We investigate the tradeoffs between fairness and total penalty for the six standard instances from [13] whose known best solutions have the highest total penalty compared to the other instances. Our motivation for this choice of



instances is simply that if the total penalty of a timetable is very small, then there is not much gain in distributing the penalty in a fair way. Our conclusion regarding this approach is that, although the known best solutions for the six instances are quite fair, we can improve the fairness further with only a rather small increase in total penalty. For a theoretical treatment of the price of fairness on so-called convex utility sets with respect to proportional fairness and max-min fairness, see the recent work by Bertsimas et al. [6].

The remainder of this paper is organized as follows. In Section 2, we will provide a brief review of the curriculum-based course timetabling (CB-CTT) problem model as well as the two fairness concepts max-min fairness and Jain's fairness index. In Section 3 we will propose two fair variants of the CB-CTT model, and in Section 4, we will introduce the optimization SA-based algorithm MAXMINFAIR\_SA for solving max-min fair allocation problems. Section 5 is dedicated to our experimental evaluation of the fairness of the known best solutions to 21 standard instances from [13] with respect to max-min fairness and Jain's fairness index, and the performance of the MAXMINFAIR\_SA algorithm.

## 2 Preliminaries

In this section, we provide a brief review of the curriculum-based course timetabling problem formulations as well as relevant definitions concerning max-min fairness and Jain's fairness index.

### 2.1 Curriculum-based Course Timetabling Problems

Curriculum-based Course Timetabling (CB-CTT) is a problem formulation for a class of optimization problems which arise when creating course schedules in the setting of a university. A central entity in the problem model is the *curriculum*. Each curriculum consists of a set of courses which must be attended by a common group of students and thus must not be held simultaneously. Our experimental evaluation of fairness in academic timetabling is based on the CB-CTT problem formulation introduced for Track 3 of the Second International Timetabling Competition (ITC2007) [27]. This formulation has emerged as one of the standard problem formulations in academic timetabling – both in research and in practice.

CB-CTT problems are NP-hard and a lot of effort has been devoted to finding heuristic approaches which provide high quality solutions within reasonable time. A wide range of techniques has been employed for solving CB-CTT instances including but not limited to approaches based on Max-SAT [2], mathematical programming [24, 8], local search [12, 26], evolutionary computation [1] as well as hybrid approaches [29]. There has been a lot of progress in terms of the achieved solution quality in the recent years. Interestingly however, there seems to be no single approach which is superior to the other approaches on all (or even most) ITC2007 instances (see [13] for current results).

A CB-CTT instance consists of the following data: We are given a set of days and each day is divided into a fixed number of timeslots. A pair composed of a day and a timeslot will be referred to as *period*. A period in conjunction with a room is called a *resource*. Additionally, we are given sets of teachers, courses, rooms and curricula. Each course has a teacher and consists of a number of lectures; each *curriculum* is a set of courses. For each room, we are provided with the maximum number of students it can accommodate. Now, given a CB-CTT instance  $I$ , the task is to create a course timetable  $\tau$ , i. e., to find an assignment of lectures to resources such that hard constraints are satisfied and soft constraint

violations are minimal. A comprehensive description of the problem model and the rationale behind it can be found in [11].

A timetable which satisfies all hard constraints is called *feasible*. The hard constraints ensure that no student and no teacher has to be present at two lectures at the same time and no two lectures can occupy a room at the same time. Additionally, all lectures have to be assigned to a suitable resource and, for each lecture, the teacher has to be available in the given period. For our purpose of creating fair course timetables, we will consider the distribution of soft constraint violations among the curricula. The CB-CTT formulation features the following four soft constraints:

- S1 *RoomCapacity*: Each lecture should be assigned to a room of sufficient size.
- S2 *MinWorkingDays*: The lectures of each course should be distributed over a certain minimum number of days.
- S3 *IsolatedLectures*: For each curriculum, all lectures associated to the curriculum should be scheduled in adjacent timeslots.
- S4 *RoomStability*: The lectures of each course should be assigned to the same room.

Each violation of one of the soft constraints results in a penalty for the timetable. The objective function aggregates individual penalties by taking their weighted sum. Detailed descriptions of how hard and soft constraints are evaluated and how much penalty is applied for a particular soft constraint violation can be found in [11].

## 2.2 Fairness in Resource Allocation

Fairness issues typically arise when scarce resources are allocated to a number of individuals with demands. E. g., fair resource allocation has received much attention in economic theory [15], but also occurs in a wide range of applications in computer science including bandwidth allocation in networks [5] and task scheduling [34]. In many optimization problems related to resource allocation, the goal is to maximize the amount of resources allocated to each individual. Fairness conditions can be imposed implicitly or explicitly in order to prevent unfair distributions of the allocated resources.

Consider a resource allocation problem with  $n$  entities or, in the following, *individuals* receiving resources. A particular resource allocation (an admissible solution) induces an allocation vector  $X = (X_1, \dots, X_n)$ , where each item  $X_i$ ,  $1 \leq i \leq n$ , corresponds to the amount of resources allocated to individual  $i$ . There are various approaches to determining the fairness of a resource allocation from the corresponding allocation vector. A fairness concept which allows for a qualitative comparison of two allocation vectors is (lexicographic) max-min fairness. It has received attention in the area of network engineering, in particular in the context of flow control [4, 20, 35, 41]. Another class of approaches are inequality measures such as the Gini index [16] and Jain's fairness index [17, 25]: an unequal distribution of resources is considered unfair. The inequality of a resources distribution is typically represented as a number, which allows for a quantitative comparison of the fairness of resource allocations. Inequality measures have been studied in economics [15], in particular in the context of income distribution. Furthermore, in many resource allocation problems, the notion of fairness is implicitly contained in the objective function. Depending on the notion of fairness, the task can be to find allocations maximizing or minimizing the sum of the individual allocations, the mean allocation, the root mean square (RMS), the smallest allocation, and so forth [30, 37].

Our evaluation of fairness in academic course timetabling focuses on the two fairness criteria max-min fairness and Jain's fairness index.

*Max-min Fairness.* Max-min fairness can be stated as iterated application of Rawls's Second Principle of Justice [33]:

“Social and economic inequalities are to be arranged so that they are to be of greatest benefit to the least-advantaged members of society.” (the Difference Principle)

Once the status of the least-advantaged members has been determined according to the difference principle, it can be applied again to everyone except the least-advantaged group in order to maximize the utility (in the economic sense) for the second least-advantaged members, and so on. The resulting utility assignment is called max-min fair. A max-min fair utility assignment implies that no member can improve its utility at the expense of any other member who received less utility. A max-min fair resource allocation is Pareto-optimal.

In order to define max-min fairness more formally, we introduce some notation. Let  $X$  be an allocation vector. Then let  $\vec{X}$  be the corresponding vector containing the values of  $X$  arranged in nondecreasing order. Let  $Y$  be another allocation vector. We write  $X \preceq_{mm} Y$  if  $X$  is at least as good as  $Y$  in the max-min sense, that is if  $\vec{Y} \preceq_{lex} \vec{X}$ , where  $\preceq_{lex}$  is the usual lexicographic comparison. A resource allocation  $X$  is called max-min optimal, if  $X \preceq_{mm} Y$  holds for all feasible resource allocations  $Y$ . Since the allocations are sorted, max-min fairness does not discriminate between individuals, but only between the amounts of resources assigned to them.

A weaker version of max-min fairness results if the fairness conditions are not applied iteratively as stated in the above definition. This means that we are just concerned with choosing the best possible outcome for the least-advantaged individuals. In the literature, related optimization problems are referred to as bottleneck optimization problems [14, 32]. In the context of academic timetabling however, this weaker fairness concept does not lead to desirable results: although, from the perspective of the least-advantaged group a timetable may be optimal, the quality of the timetable from the perspective of other stakeholders is not considered.

*Jain's Fairness Index.* While max-min fairness enforces a certain efficiency in resource utilization and provides a qualitative measure of fairness, Jain's fairness index [17] quantifies the inequality of a given resource distribution. An equal distribution of resources is considered fair, while an unequal distribution is considered unfair. It is the crucial fairness measure that is applied in the famous AIMD algorithm used in TCP Congestion Avoidance [10]. The fairness index  $J(X)$  of an allocation vector  $X$  is defined as

$$J(X) = \frac{\left( \sum_{1 \leq i \leq n} X_i \right)^2}{n \cdot \sum_{1 \leq i \leq n} X_i^2} . \quad (1)$$

It has several useful properties like population size independence, scale and metric independence, it is bounded between 0 and 1, and it has an intuitive interpretation. In particular  $J(X) = 1$  means that  $X$  is a completely fair allocation, i. e., the allocation is fair for every individual, and if  $J(X) = 1/n$  then all resources are occupied by a single individual. Furthermore, if  $J(X) = x\%$  then the allocation  $X$  is fair for  $x$  percent of the individuals.

### 3 Fairness in Academic Course Timetabling

Course timetabling problems fit quite well in the framework of fair resource allocation problems described in the previous section: A timetable is an allocation of resources (rooms,

timeslots) to lectures. In this section, we will define two fair versions of the CB-CTT problem model. The first one, MMF-CB-CTT, is based on max-min fairness. Since max-min fairness enforces efficiency (maximum utility) as well as fairness at least to some extent, it is not a suitable concept for exploring the tradeoff between fairness and efficiency. Therefore we propose a second fair variant of CB-CTT called JFI-CB-CTT that is based on Jain's fairness index.

In order to employ the fairness concepts mentioned in the previous section, we need to define how to determine the allocation vector for a timetable. The central entities in the CB-CTT problem model are the curricula. Therefore, in this work, we are interested in a fair distribution of the penalty values assigned to the curricula. Please note that we interpret utility as the opposite of penalty. Hence, a timetable which receives less penalty than another timetable has a higher utility. We achieve the transformation from penalty to utility by simply changing the signs of the penalty values. Let  $I$  be a CB-CTT instance with curricula  $c_1, c_2, \dots, c_k$  and let  $f_c$  be the usual CB-CTT objective function from [11], which evaluates (S1)-(S4) restricted to curriculum  $c$ . This means  $f_c$  determines soft constraint violations only for the courses in curriculum  $c$ . For a timetable  $t$  the corresponding allocation vector is given by the allocation function

$$A(t) = (-f_{c_1}(t), -f_{c_2}(t), \dots, -f_{c_k}(t)) . \quad (2)$$

**Definition 1 (MMF-CB-CTT)** Given a CB-CTT instance  $I$ , the task is to find a feasible timetable  $t$  such that  $A(t)$  is max-min optimal.

If a feasible timetable corresponds to a max-min optimal allocation, then any curriculum  $c$  could receive less penalty only at the expense of other curricula which receive more penalty than  $c$ . Since each student is struck only by the penalty assigned to his or her curriculum the group of students with the worst timetable receive the best possible timetable and under this condition, the students with the second-worst timetable receive the best possible timetable, and so on.

In order to explore the tradeoff between efficient and fair resource allocation in curriculum-based timetabling, we propose another fair variant of CB-CTT called JFI-CB-CTT that is based on Jain's fairness index [17]. In order to get meaningful results from the fairness index however, we need a different allocation function. Consider an allocation  $X$ , where all penalty is allocated to a single curriculum while the remaining  $k - 1$  curricula receive no penalty. Then  $J(X) = 1/k$ , which means that only one curriculum is happy with the allocation (see [17]). In our situation however, the opposite is the case:  $k - 1$  curricula are happy since they receive no penalty at all. The following allocation function shifts the penalty values such that the corresponding fairness index in the situation described above becomes  $(k - 1)/k$ , which is in much better agreement with our intuition:

$$A'(t) = (f_{max} - f_{c_1}(t), f_{max} - f_{c_2}(t), \dots, f_{max} - f_{c_k}(t)) , \quad (3)$$

with

$$f_{max} = \max_{1 \leq i \leq k} \{f_{c_i}(t)\} .$$

**Definition 2 (JFI-CB-CTT)** Given a CB-CTT instance  $I$ , the task is to find the set of feasible solutions which are Pareto-optimal with respect to the two objectives of the objective function

$$F(t) = (f(t), 1 - J(A'(t))) , \quad (4)$$

where  $f$  is the CB-CTT objective function from [11] and  $J$  is defined in Eq. (1).

In a similar fashion, other classes of timetabling problem such as post-enrollment course timetabling, exam timetabling and nurse rostering can be turned into fair optimization problems. For example, for post-enrollment course timetabling, the central entities of interest would likely be the individual students, not the curricula as for CB-CTT problems. Therefore, the goal would be a fair distribution of penalty over all students. Once an appropriate allocation function has been defined, we immediately get the corresponding fair optimization problems.

Our proposed problem formulations are concerned with balancing the interests within one group of individuals, namely the students. In practice however, there are often several groups of individuals with possibly conflicting interest, for example students, lecturers and administration. The proposed models can be extended to multiple groups of stakeholders in a straight-forward manner: The penalty values for all individuals (in all groups) are stored in the allocation vector, and since all values are just penalty values, the fairness concepts can be applied as proposed above. This approach requires some additional thought however, since the interest of all individuals are considered to be equally important, which may or may not be intended in practice (weighting can be applied of course). Another approach, which avoids the problem of giving explicit priorities to the interests of different groups extends the problem formulation based on Jain's fairness index: The fairness index can be determined independently for each group and the problem model can be extended to a  $(d + 1)$ -objective optimization problem, where  $d$  is the number of groups of stakeholders under consideration. The set of Pareto-optimal solutions characterizes the tradeoffs between the interests of the different groups and the total penalty.

#### 4 Simulated Annealing for Max-Min Fair Course Timetabling and Three Measures for Energy Difference

Simulated Annealing (SA) is a popular local search method which works surprisingly well on many problem domains [19]. SA has been applied successfully to timetabling problems [21, 38] and some of the currently known best solutions to CB-CTT instances from the ITC2007 competition were discovered by simulated annealing-based methods [13]. Our SA for max-min fair optimization problems shown in Algorithm 1 below (algorithm MAXMINFAIR\_SA) is conceptually very similar to the original SA algorithm proposed by Kirkpatrick et al. [19]. Since max-min fairness only tells us which of two given solutions is better, but not how much better, the main challenge in tailoring SA to max-min fair optimization problems is to find a suitable energy difference function, which quantifies the difference in quality between two candidate solutions. In the following, we propose three different energy difference measures for max-min fair optimization and provide details on the acceptance criterion, the cooling schedule, and the neighborhood exploration method used for the experimental evaluation of MAXMINFAIR\_SA in the next section.

*Acceptance Criterion.* Similar to the original SA algorithm proposed by Kirkpatrick et al. in [19], algorithm MAXMINFAIR\_SA accepts an improved or equally good solution  $s_{\text{next}}$  with probability 1. If  $s_{\text{next}}$  is worse than  $s_{\text{cur}}$  then the acceptance probability depends on the current temperature level  $\vartheta$  and the energy difference  $\Delta E$ . The energy difference measures the difference in quality of the allocation induced by  $s_{\text{next}}$  compared to the allocation induced

**Algorithm 1:** MAXMINFAIR\_SA

---

**input** :  $s_{cur}$ : feasible timetable,  $\vartheta_{max}$ : initial temperature,  $\vartheta_{min}$ : final temperature, timeout  
**output**:  $s_{best}$ : Best feasible timetable found so far

$s_{best} \leftarrow s_{cur}$   
 $\vartheta \leftarrow \vartheta_{max}$   
**while** *timeout not hit* **do**  
     $s_{next} \leftarrow \text{neighbor}(s_{cur})$   
    **if**  $P_{accept} \geq \text{random}()$  **then**  $s_{cur} \leftarrow s_{next}$   
    **if**  $A(s_{cur}) \preceq_{mm} A(s_{best})$  **then**  $s_{best} \leftarrow s_{cur}$   
     $\vartheta \leftarrow \text{next\_temperature}(\vartheta)$   
**end**  
**return**  $s_{best}$

---

by the current solution  $s_{cur}$ . The acceptance probability  $P_{accept}$  is defined as:

$$P_{accept} = \begin{cases} 1 & \text{if } s_{next} \preceq_{mm} s_{cur} \\ \exp\left(-\frac{\Delta E(X, Y)}{\vartheta}\right) & \text{otherwise,} \end{cases}$$

where  $X = A(s_{cur})$  and  $Y = A(s_{next})$ . With max-min fair optimization in mind, we propose the following three energy difference measures for the energy difference,  $\Delta E_{lex}$ ,  $\Delta E_{cw}$ , and  $\Delta E_{ps}$ , which are based on lexicographic comparison, component-wise ratios and the ratios of the partial sums of the sorted allocation vectors, respectively. Our experiments presented in the next section indicate that choosing one energy difference function over another has a clear impact on the performance of Algorithm MAXMINFAIR\_SA. Hence the choice of the energy difference function is a critical design choice.

For two allocation vectors  $X$  and  $Y$  of length  $n$ , let the energy difference  $\Delta E_{lex}$  be (note  $\vec{X}_i$  denotes the  $i$ th entry after sorting the entries of  $X$ ,  $\vec{Y}_i$  is defined analogously)

$$\Delta E_{lex}(X, Y) = 1 - \frac{1}{n} \cdot \left( \min_{1 \leq i \leq n} \{i \mid \vec{X}_i > \vec{Y}_i\} + 1 \right). \quad (5)$$

$\Delta E_{lex}$  determines the energy difference between  $X$  and  $Y$  from the index of the sorted allocation vectors at which the comparison  $X \preceq_{mm} Y$  shows that  $X$  is better than  $Y$ . Thus, sorted allocation vectors which differ at the most significant indices have a higher energy difference than those which differ at later indices. In order to make the numerical range of  $\Delta E_{lex}$  independent of the actual size of the allocation vector, which may vary from instance to instance, the result is normalized by the length  $n$  of the allocation vectors.

$\Delta E_{lex}$  only considers the earliest index at which two sorted allocation vectors differ but ignores how much the actual entries differ. We additionally propose the two energy difference measures  $\Delta E_{cs}$  and  $\Delta E_{ps}$  which take this information into account. These two energy difference measures were inspired by the definitions of approximation ratios for max-min fair allocation problems given by Kleinberg et al. in [20]. An approximation ratio is a measure for how much worse the quality of a solution is relative to a possibly unknown optimal solution. In our case, we are interested in how much worse one given allocation is relative to another given allocation. Despite the different context, we can use the same general ideas. Let  $\mu_{X, Y}$  be the smallest value of the two allocation vectors  $X$  and  $Y$  offset by a parameter  $\delta > 0$ , i. e.,

$$\mu_{X, Y} = \min\{\vec{X}_1, \vec{Y}_1\} - \delta. \quad (6)$$

The component-wise energy difference  $\Delta E_{cw}$  of two allocations  $X$  and  $Y$  is defined as:

$$\Delta E_{cw}(X, Y) = \max_{1 \leq i \leq n} \left\{ \frac{\mu_{X,Y} - \bar{X}_i}{\mu_{X,Y} - \bar{Y}_i} \right\} - 1 \quad (7)$$

Unlike  $\Delta E_{lex}$ , the component-wise energy difference does not take into account explicitly which entries of the sorted allocation vectors are responsible for  $X \preceq_{mm} Y$ . There is however a bias towards the ratios of entries which occur early in the sorted allocation vectors. Since all entries are subtracted from  $\mu_{X,Y}$ , the ratios of the most significant entries with respect to  $\preceq_{mm}$  tend to govern the value component-wise energy difference. Consider for example the situation that  $Y$  is much worse than  $X$ , say,  $\min\{\bar{X}_1, \bar{Y}_1\}$  occurs more often in  $X$  than in  $Y$ . Then for  $\delta \ll 1$  the energy difference  $\Delta E_{cw}(X, Y)$  becomes large. On the other hand, if  $X$  is nearly as good as  $Y$  then the ratios are all close to one and thus  $\Delta E_{cw}(X, Y)$  is close to zero.

The third proposed energy difference measure  $\Delta E_{ps}$  is based on the ratios of the partial sums  $\sigma_i(X)$  of the sorted allocation vectors.

$$\sigma_i(X) = \sum_{1 \leq j \leq i} X_j .$$

The intention of using partial sums of the sorted allocations is to give the individuals who receive the most penalty, and hence occur early in the sorted allocation vectors, more influence on the resulting energy difference compared to  $\Delta E_{cw}$ . The energy difference  $\Delta E_{ps}$  is defined as

$$\Delta E_{ps}(X, Y) = \max_{1 \leq i \leq n} \left\{ \frac{i \cdot \mu_{X,Y} - \sigma_i(\bar{X})}{i \cdot \mu_{X,Y} - \sigma_i(\bar{Y})} \right\} - 1 . \quad (8)$$

*Cooling Schedule.* In algorithm MAXMINFAIR\_SA, the function `next_temperature` updates the current temperature level  $\vartheta$  according to the cooling schedule. We use a standard geometric cooling schedule

$$\vartheta = \alpha^t \cdot \vartheta_{max} ,$$

where  $\alpha$  is the cooling rate and  $t$  is the elapsed time. Geometric cooling schedules decrease the temperature level exponentially over time. It is a popular class of cooling schedules which is widely used in practice and works well in many problem domains including timetabling problems [23, 22, 39]. Geometric cooling was chosen due to its simplicity, since the main focus of our evaluation in Section 5 is the performance impact of the different energy difference functions. We have made a slight adjustment to the specification of the geometric cooling schedule in order to make the behavior more consistent for different timeouts. Instead of specifying the cooling rate  $\alpha$ , we determine  $\alpha$  from  $\vartheta_{max}$ , the desired minimum temperature  $\vartheta_{min}$  and the timeout according to:

$$\alpha = \left( \frac{\vartheta_{min}}{\vartheta_{max}} \right)^{\frac{1}{\text{timeout}}} . \quad (9)$$

Thus, at the beginning ( $t = 0$ ) the temperature level is  $\vartheta_{max}$  and when the timeout is reached ( $t = \text{timeout}$ ), the temperature level becomes  $\vartheta_{min}$ . We chose to set a timeout rather than a maximum number of iterations since this setting is compliant with the ITC2007 competition conditions, which are a widely accepted standard for comparing results.

Table 1: Fairness of the known best timetables from [13] for the ITC2007 CB-CTT instances.

Instance	Curricula	$f(s_{best})$	$J(A'(s_{best}))$	$-\bar{A}(s_{best})$
comp01	14	5	0.8571	$5^2, 0^{12}$
comp02	70	24	0.9515	$4, 2^{10}, 0^{59}$
comp03	68	66	0.9114	$13, 10^3, 9, 7^2, 6^4, 5^{13}, 4, 2^6, 0^{37}$
comp04	57	35	0.8964	$7, 6^3, 5^4, 4^2, 2, 0^{46}$
comp05	139	291	0.8277	$41^2, 36^7, 35^5, 32^5, 31^6, 30^9, 28, 27^7, 26^2, 25^{14}, \dots, 2, 0^3$
comp06	70	27	0.9657	$12, 7^2, 5^4, 2^3, 0^{60}$
comp07	77	6	0.9870	$6, 0^{76}$
comp08	61	37	0.9020	$7, 6^3, 5^4, 4^2, 2^2, 0^{49}$
comp09	75	96	0.8047	$10^5, 9, 7^{10}, 6^6, 5^{10}, 4, 2, 0^{41}$
comp10	67	4	0.9701	$2^2, 0^{65}$
comp11	13	0	—	$0^{13}$
comp12	150	300	0.9128	$45, 30^{14}, 28, 27^2, 26^5, 25^{19}, 22^4, 21^6, 20^8, 19, \dots, 2^2, 0^3$
comp13	66	59	0.8830	$8, 7, 6^5, 5^7, 4^2, 2^3, 0^{47}$
comp14	60	51	0.9023	$8^4, 7, 5^2, 2^6, 0^{47}$
comp15	68	66	0.8495	$10^3, 9^3, 7, 6^4, 5^{13}, 4, 2^7, 0^{36}$
comp16	71	18	0.9176	$7^2, 5^7, 4, 0^{61}$
comp17	70	56	0.9248	$10^2, 6^3, 5^9, 2^4, 0^{52}$
comp18	52	62	0.9009	$17, 15, 14, 13, 11, 10, 9^2, 5^{19}, 2^2, 0^{23}$
comp19	66	57	0.9612	$13, 7, 6^4, 5^2, 4, 2^7, 0^{50}$
comp20	78	4	0.9744	$2^2, 0^{76}$
comp21	78	76	0.8838	$12, 11, 10^4, 9, 7^4, 6^4, 5^{12}, 4, 2^3, 1^2, 0^{45}$

*Neighborhood.* In our max-min fair SA implementation, the function `neighbor` picks at random a neighbor in the Kempe-neighborhood of  $s_{cur}$ . The Kempe-neighborhood is the set of all timetables which can be reached by performing a single Kempe-move, which is a well-known and widely used operation for swapping events in a timetable [7, 26, 28, 39, 40]. A prominent feature of Kempe-moves is that they preserve the feasibility of a timetable. Since the algorithm MAXMINFAIR\_SA only uses Kempe-moves to modify timetables the output is guaranteed to be feasible if the input timetable is feasible. In the future, more advanced neighborhood exploration methods similar to those proposed for example in [12, 26] could be used, which may well lead to an improved overall performance of MAXMINFAIR\_SA.

## 5 Evaluation

In this section, we will first address the question how fair or unfair the known best timetables for the ITC2007 CB-CTT instances are with respect to Jain's fairness index and max-min fairness. Table 1 shows our measurements of how fair the best existing solutions to the CB-CTT instances `comp01`, `comp02`, ..., `comp21` are (see [13] for instance data). Please note that the known best timetables were not created with fairness in mind, but the objective was to create timetables with minimal total penalty. In Table 1,  $s_{best}$  refers to the known best solution for each instance.  $A$  and  $A'$  refer to the allocation functions given in (2) and (3), respectively. The data indicates that the timetables with a low total penalty are also rather fair. This can be explained by the fact that these timetables do not have a large amount of penalty to distribute over the curricula. Thus, most curricula receive little or no penalty and consequently, the distribution is fair for most curricula. We will show in Section 5.2



Table 2: The performance of MAXMINFAIR\_SA with  $\Delta E = \Delta E_{cw}$  for different values of  $\delta$ .

$\delta$	$10^0$	$10^{-2}$	$10^{-3}$	$10^{-6}$
$10^0$	–	02	02,05	02
$10^{-2}$	10,19,20	–	09	19
$10^{-3}$	01,10,19,20	–	–	03,19
$10^{-6}$	01,10,20	–	–	–

however, that for timetables with a comparatively large total penalty there is still some room for improvement concerning fairness.

The rightmost column of Table 1 contains the sorted allocation vectors of the best solutions. For a more convenient presentation, all entries of the sorted allocation vectors are multiplied by -1. The exponents denote how often a certain number occurs. For example, the sorted allocation vector  $(-5, -5, 0, 0, 0)$  would be represented as  $5^2, 0^3$ . The sum of the values of an allocation vector is generally much larger than the total penalty shown in the second column. The reason for this is that the penalty assigned to a course is counted for each curriculum the course belongs to. With a few exceptions the general theme seems to be that the penalty is assigned to only a few curricula while a majority of curricula receives no penalty. In the next section we will show that the situation for the curricula which receive the most penalty can be improved with max-min fair optimization for 15 out of 21 instances.

### 5.1 Max-Min Fair Optimization

In Section 4, we presented a SA-based algorithm for solving max-min fair resource allocation problems. A crucial part of this algorithm is the energy difference measure which determines how much worse a given solution is compared to another solution, i.e. the energy difference of the solutions. We evaluate the impact of the three energy difference measures (5), (7) and (8) on the performance of MAXMINFAIR\_SA.

Our test setup was the following: For each energy difference function we independently performed 50 runs with MAXMINFAIR\_SA. The temperature levels were determined experimentally, we set  $\vartheta_{max} = 5$  and  $\vartheta_{min} = 0.01$ ; the cooling rate  $\alpha$  was set according to (9). In order to establish consistent experimental conditions for fair optimization, we used a timeout, which was determined according to the publicly available ITC2007 benchmark executable. On our machines (i7 CPUs running at 3.4GHz, 8GB RAM), the timeout was set to 192 seconds. The MAXMINFAIR\_SA algorithm was executed on a single core. We generated feasible initial timetables for MAXMINFAIR\_SA as a preprocess using standard sequential heuristics [9]. The soft constraint violations were not considered at this stage. Since the preprocess was performed only once per instance (not per run), it is not counted in the timeout. However, the time it took was negligible compared to the timeout (less than 1 second per instance).

Table 2 shows the impact of the parameter  $\delta$  on the performance of MAXMINFAIR\_SA with energy difference measure  $\Delta E_{cw}$ . For each pair of values we performed the one-sided Wilcoxon Rank-Sum test with a significance level of 0.01. The data indicates that for best performance,  $\delta$  should be small, but not too small. For  $\delta = 1$ , MAXMINFAIR\_SA beats the other shown configurations on instance comp02 but performs worse than the other configurations on instances comp10 and comp20. For  $\delta = 10^{-6}$  the overall performance is better than for  $\delta = 1$ , but worse than for the other configurations. With  $\delta = 10^{-2}$  and  $\delta = 10^{-3}$ ,

Table 3: The performance of MAXMINFAIR\_SA with energy difference measures  $\Delta E_{lex}$ ,  $\Delta E_{ps}$  and  $\Delta E_{cw}$ .

$\Delta E$	$\Delta E_{lex}$	$\Delta E_{ps}$	$\Delta E_{cw}$
$\Delta E_{lex}$	–	–	–
$\Delta E_{ps}$	all except 01, 06, 08, 11, 17	–	18
$\Delta E_{cw}$	all except 11	06, 07, 08, 17, 21	–

MAXMINFAIR\_SA shows the best relative performance. Thus, for our further evaluation we set  $\delta = 10^{-3}$ .

Table 3 shows the relative performance of Algorithm MAXMINFAIR\_SA for the proposed energy difference measures (5), (7) and (8). The table shows for any choice of two energy difference measures  $i$  and  $j$ , for which instances MAXMINFAIR\_SA with measure  $i$  performs significantly better than MAXMINFAIR\_SA using measure  $j$ . Again, we used the Wilcoxon Rank-Sum test with a significance level of 1 percent. The data shows that  $\Delta E_{cw}$  is the best choice among the three alternatives, since it is a better choice than  $\Delta E_{lex}$  on all instances except comp11 and a better choice than  $\Delta E_{ps}$  on five out of 21 instances. However, although  $\Delta E_{cw}$  shows significantly better performance than the other energy difference measures, it did not necessarily produce the best timetables on all instances. For the instances comp03, comp15, comp05 and comp12 for example, the best solution found with  $\Delta E = \Delta E_{ps}$  was better than with  $\Delta E = \Delta E_{cw}$ .

The data in Table 4 shows a comparison of the sorted allocation vectors of the known best solutions from [13] with the best solutions found by the 50 runs of MAXMINFAIR\_SA with  $\Delta E = \Delta E_{cw}$ . First of all, for instances comp01 and comp11, the allocation vectors of the best existing solutions and the best solution found by MAXMINFAIR\_SA are identical. This means that MAXMINFAIR\_SA finds reasonably good solutions despite the certainly more complex fitness landscape due to max-min fair optimization. We can also observe that the maximum penalty any curriculum receives is significantly less for most instances and the penalty is more evenly distributed across the curricula. This means that although max-min fair timetables may have a higher total penalty, they might be more attractive from the students' perspective, since in the first place each student notices an unfortunate arrangement of his/her timetable, which is tied to the curriculum. Furthermore, we can observe that if the total penalty of a known best solution is rather low, then it is also good with respect to max-min fairness. For several instances in this category, (comp01, comp04, comp07, comp10 and comp20), the solution found by MAXMINFAIR\_SA is not as good as the known best solution with respect to max-min fairness. We can conclude that if there is not much penalty to distribute, it is not necessary to bother about a fair distribution of penalty.

## 5.2 The Tradeoff Between Fairness and Efficiency

We proposed the JFI-CB-CTT problem formulation in Section 3, which allows us to investigate the tradeoff between fairness and efficiency which arises in course timetabling. We can observe in column 4 of Table 1 that for all of the best solutions from [13] the fairness index (1) is greater than 0.8, i. e., the known best solutions are also fair for more than 80 percent of the curricula. In order to solve the corresponding JFI-CB-CTT instances, we use the multi-objective optimization algorithm AMOSA proposed in [3] that is based on simulated annealing like Algorithm MAXMINFAIR\_SA. Since we do not expect from a general multi-objective optimization algorithm to produce solutions as good as the best CB-CTT

Table 4: Comparison of the sorted allocation vectors of the known best solutions from [13] with the allocation vectors found by MAXMINFAIR\_SA with respect to max-min fairness.

Instance	Known best solution	MAXMINFAIR_SA ( $\Delta E = \Delta E_{cw}$ )
comp01	$5^2, 0^{12}$	$5^2, 0^{12}$
comp02	$4, 2^{10}, 0^{59}$	$4^2, 2^{31}, 1^7, 0^{30}$
comp03	$13, 10^3, 9, 7^2, 6^4, 5^{13}, 4, 2^6, 0^{37}$	$6^4, 4^{11}, 2^{22}, 1^3, 0^{28}$
comp04	$7, 6^3, 5^4, 4^2, 2, 0^{46}$	$6^4, 4^2, 2^4, 1, 0^{46}$
comp05	$41^2, 36^7, 35^5, 32^5, 31^6, 30^9, 28, \dots, 2, 0^3$	$19^2, 18^3, 17^3, 16^5, 15^2, 14^{15}, 13^5, \dots, 4^8, 3^3, 2$
comp06	$12, 7^2, 5^4, 2^3, 0^{60}$	$12, 4^2, 2^{30}, 1^{13}, 0^{24}$
comp07	$6, 0^{76}$	$6, 2^{23}, 1^{24}, 0^{29}$
comp08	$7, 6^3, 5^4, 4^2, 2^2, 0^{49}$	$6^4, 4^2, 2^7, 1^5, 0^{43}$
comp09	$10^5, 9, 7^{10}, 6^6, 5^{10}, 4, 2, 0^{41}$	$6^9, 4^{14}, 2^{17}, 0^{35}$
comp10	$2^2, 0^{65}$	$2^{19}, 1^6, 0^{42}$
comp11	$0^{13}$	$0^{13}$
comp12	$45, 30^{14}, 28, 27^2, 26^5, 25^{19}, 22^4, \dots, 2^2, 0^3$	$10^3, 9^6, 8^{31}, 7^7, 6^{43}, 5^2, 4^{36}, 3^2, 2^{16}, 1, 0^3$
comp13	$8, 7, 6^5, 5^7, 4^2, 2^3, 0^{47}$	$6^6, 4^4, 2^{13}, 1^6, 0^{37}$
comp14	$8^4, 7, 5^2, 2^6, 0^{47}$	$8^4, 4^2, 3, 2^{18}, 0^{35}$
comp15	$10^3, 9^3, 7, 6^4, 5^{13}, 4, 2^7, 0^{36}$	$6^4, 4^{11}, 2^{23}, 1^2, 0^{28}$
comp16	$7^2, 5^7, 4, 0^{61}$	$4^5, 2^{16}, 1^4, 0^{46}$
comp17	$10^2, 6^3, 5^9, 2^4, 0^{52}$	$10^2, 6^2, 4^7, 3, 2^{25}, 1^7, 0^{26}$
comp18	$17, 15, 14, 13, 11, 10, 9^2, 5^{19}, 2^2, 0^{23}$	$4^{20}, 2^{11}, 1^5, 0^{16}$
comp19	$13, 7, 6^4, 5^2, 4, 2^7, 0^{50}$	$6^4, 4^6, 2^{15}, 1^{14}, 0^{27}$
comp20	$2^2, 0^{76}$	$4^5, 3^3, 2^{31}, 1^7, 0^{32}$
comp21	$12, 11, 10^4, 9, 7^4, 6^4, 5^{12}, 4, 2^3, 1^2, 0^{45}$	$10, 6^4, 5, 4^{15}, 3, 2^{36}, 1^3, 0^{17}$

solvers, we will consider the following scenario to explore the tradeoffs between fairness and efficiency: starting from the known best solution we examine how much increase in total penalty we have to tolerate in order to increase the fairness further. We will take as examples the six instances with the highest total amount of penalty, comp03, comp05, comp09, comp12, comp15 and comp21.

The temperature levels for the AMOSA algorithm were set to  $\vartheta_{max} = 20$  and  $\vartheta_{min} = 0.01$ ;  $\alpha$  was set according to (9) with a timeout determined by the official ITC2007 benchmark. The plots in Figure 1 show the (Pareto-) non-dominated solutions found by AMOSA. The arrows point to the starting point, i.e. the best available solutions to the corresponding instances. For instances comp05 and comp21 solutions with a lower total cost than the previously known best solutions were discovered by this approach. The plots show that the price for increasing the fairness is generally not very high – up to a certain level, which depends on the instance. In fact, for comp09 and comp21, the fairness index can be increased by 3.5 percent and 1.4 percent, respectively, without increasing the total penalty at all.

In Figure 1, the straight lines that go through the initial solutions show a possible tradeoff between fairness and efficiency: the slopes were determined such that a 1 percent increase in fairness yields a 1 percent increase in penalty. For the instances shown in Figure 1, the solutions remain close to the tradeoff lines up to a fairness of 94 to 97 percent, while a further increase in fairness demands a significant increase in total cost. For the instances comp05, comp09 and comp15, there are several solutions below the tradeoff lines. Picking any of the solutions below these lines would result in an increased fairness without an equally large increase in the amount of penalty. This means picking a fairer solution might well be an attractive option in a real-world academic timetabling context. For comp05 for example, the

fairness of the formerly best known solution with a total penalty of 291 can be increased by 5.4 percent at 302 total penalty, which is a 3.8 percent increase.

In summary, improving the fairness of an efficient timetable as a post-processing step seems like a viable approach for practical decision making. Using a very efficient solution as a starting point means that we can benefit from the existing very good approaches to creating timetables with minimal total cost and provide improved fairness depending on the actual, instance-dependent tradeoff.

## 6 Conclusion

In this paper we introduced two new problem formulations for academic course timetabling based on the CB-CTT problem model from track three of the ITC2007, MMF-CB-CTT and JFI-CB-CTT. Both problem formulations are aimed at creating fair course timetables in the setting of a university but include different notions of fairness. Fairness in our setting means that the penalty assigned to a timetable is distributed in a fair way among the different curricula. The MMF-CB-CTT formulation aims at creating max-min fair course timetables while JFI-CB-CTT is a bi-objective problem formulation based on Jain's fairness index. The motivation for the JFI-CB-CTT formulation is to explore the tradeoff between a fair penalty distribution and a low total penalty.

Furthermore, we proposed an optimization algorithm based on simulated annealing for solving MMF-CB-CTT problems. A critical part of the algorithm is concerned with measuring the energy difference between two timetables, i.e., how much worse a timetable is compared to another timetable with respect to max-min fairness. We evaluated the performance of the proposed algorithm for three different energy difference measures on the 21 CB-CTT benchmark instances. Our results show clearly that the algorithm performs best with  $\Delta E_{CW}$  as energy difference measure.

Additionally, we investigated the fairness of the known best solutions of the 21 CB-CTT instances with respect to max-min fairness and Jain's fairness index. These solutions were not created with fairness in mind, but our results show that all of the solutions have a fairness index greater than 0.8. This means they can be considered quite fair. Nevertheless, our results show that some improvements are possible with respect to both max-min fairness and Jain's fairness index. The timetables produced by our proposed MAXMINFAIR\_SA algorithms are better than the known best ones with respect to max-min fairness for 15 out of 21 instances. Our investigation of the tradeoff between fairness and the total amount of penalty using the JFI-CB-CTT problem formulation shows that the fairness of the known best timetables can be increased further with only a small increase of the total penalty.

## Acknowledgements

We would like to thank the anonymous referees for their valuable remarks on this paper.

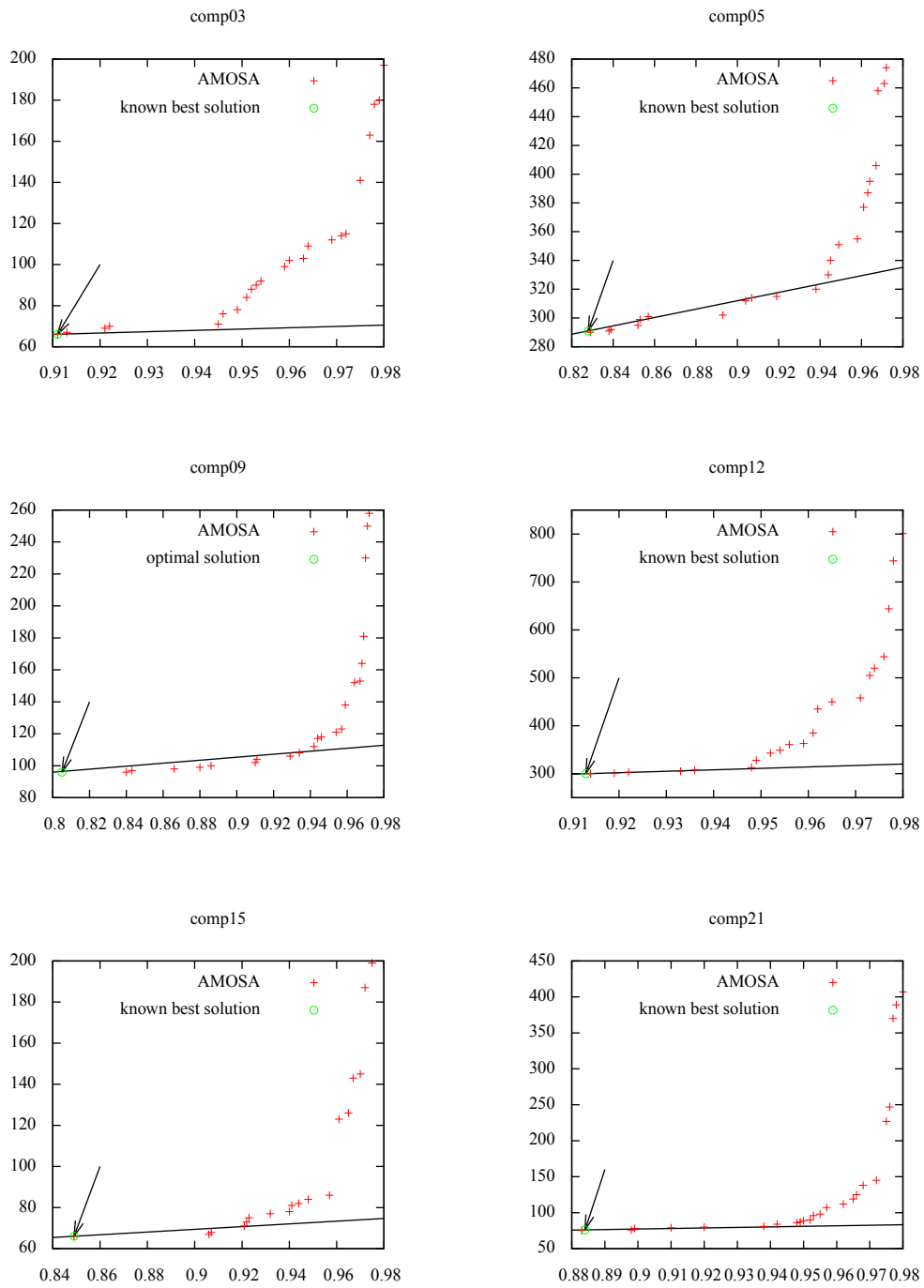


Fig. 1: Non-dominated solutions found by the AMOSA algorithm for the JFI-CB-CTT versions of instances comp03, comp05, comp09, comp12, comp15 and comp21. All graphs show the fairness index on the horizontal axis and the amount of penalty on the vertical axis.

## References

1. Salwani Abdullah, Edmund K. Burke, and Barry McCollum. A hybrid evolutionary approach to the university course timetabling problem. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1764–1768, 2007.
2. Roberto Asín Acha and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. In *Proc. 8th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 42–56, 2010.
3. Sanghamitra Bandyopadhyay, Sriparna Saha, Ujjwal Maulik, and Kalyanmoy Deb. A simulated annealing-based multiobjective optimization algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*, 12:269–283, 2008.
4. Yair Bartal, Martin Farach-Colton, Shibu Yooseph, and Lisa Zhang. Fast, fair and frugal bandwidth allocation in ATM networks. *Algorithmica*, 33(3):272–286, 2002.
5. Dimitri P. Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, 2nd edition, 1992.
6. Dimitris Bertsimas, Vivek F. Farias, and Nikolaos Trichakis. The price of fairness. *Operations Research*, 59(1):17–31, February 2011.
7. Edmund K. Burke, Adam J. Eckersley, Barry McCollum, Sanja Petrovic, and Rong Qu. Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research*, 206(1):46–53, 2010.
8. Edmund K. Burke, Jakub Mareček, Andrew J. Parkes, and Hana Rudová. A branch-and-cut procedure for the udine course timetabling problem. *Annals of Operations Research*, 194(1):71–87, 2011.
9. Edmund K. Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.
10. Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.
11. Luca Di Gaspero, Barry McCollum, and Andrea Schaerf. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (Track 3). Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0/1, School of Electronics, Electrical Engineering and Computer Science, Queens University, Belfast (UK), August 2007.
12. Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5:65–89, 2006.
13. Luca Di Gaspero and Andrea Schaerf. Curriculum-based course timetabling site. <http://satt.diegm.uniud.it/ctt/>, January 2012.
14. Jack Edmonds and D.R. Fulkerson. Bottleneck extrema. *Journal of Combinatorial Theory*, 8(3):299–306, 1970.
15. Allan M. Feldman and Roberto Serrano. *Welfare economics and social choice theory*. Springer, New York, NY, 2nd edition, 2006.
16. Corrado Gini. Measurement of inequality of incomes. *The Economic Journal*, 31(121):124–126, January 1921.
17. Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report DEC-TR-301, Digital Equipment Corporation, September 1984.
18. Frank Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society*, volume 49, 1998.
19. Scott Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
20. Jon Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. *Journal of Computer and System Sciences*, 63(1):2–20, 2001.
21. Philipp Kostuch. The university course timetabling problem with a three-phase approach. In *Proc. 5th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 109–125. Springer, 2004.
22. Christos Koulamas, Solomon Antony, and R. Jaen. A survey of simulated annealing applications to operations research problems. *Omega*, 22(1):41–56, 1994.
23. P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, 1987.
24. Gerald Lach and Marco E. Lübbecke. Curriculum based course timetabling: new solutions to Udine benchmark instances. *Annals of Operations Research*, pages 1–18, 2010.
25. Tian Lan, David Kao, Mung Chiang, and Ashutosh Sabharwal. An axiomatic theory of fairness in network resource allocation. In *INFOCOM*, pages 1343–1351. IEEE, 2010.

26. Zhipeng Lü and Jin-Kao Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.
27. Barry McCollum, Andrea Schaefer, Ben Paechter, Paul McMullan, Rhys Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22:120–130, 2010.
28. Liam T. G. Merlot, Natashia Boland, Barry D. Hughes, and Peter J. Stuckey. A hybrid algorithm for the examination timetabling problem. In *Proc. 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 207–231. Springer, 2002.
29. Tomáš Müller. ITC2007 solver description: a hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.
30. Włodzimierz Ogryczak. Bicriteria models for fair and efficient resource allocation. In *Proceedings of the 2nd International Conference on Social Informatics (SocInfo)*, pages 140–159. Springer, 2010.
31. Włodzimierz Ogryczak and Adam Wierzbicki. On multi-criteria approaches to bandwidth allocation. *Control and Cybernetics*, 33:427–448, 2004.
32. Abraham P. Punnen and Ruonan Zhang. Quadratic bottleneck problems. *Naval Research Logistics (NRL)*, 58(2):153–164, 2011.
33. John Rawls. *A Theory of Justice*. Belknap Press of Harvard University Press, revised edition, 1999.
34. Thomas Repantis, Yannis Drougas, and Vana Kalogeraki. Adaptive resource management in Peer-to-Peer middleware. In *Proc. 19th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2005.
35. Ronaldo M. Salles and Javier A. Barria. Lexicographic maximin optimisation for fair bandwidth allocation in computer networks. *European Journal of Operational Research*, 185(2):778 – 794, 2008.
36. Amartya Sen and James E. Foster. *On economic inequality*. Clarendon Press ; Oxford University Press, Oxford : New York :, enl. ed., with a substantial annexe "on economic inequality after a quarter century" / James Foster and Amartya Kumar Sen. edition, 1997.
37. M. J. Soomer and G. M. Koole. Fairness in the aircraft landing problem. In *Proceedings of the Anna Valicek Competition 2008*, 2008.
38. Jonathan Thompson and Kathryn A. Dowsland. General cooling schedules for a simulated annealing based timetabling system. In *Proc. 1st Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 345–363, 1996.
39. Jonathan M. Thompson and Kathryn A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25(7-8):637–648, 1998.
40. Mauritsius Tuga, Regina Berretta, and Alexandre Mendes. A hybrid simulated annealing with Kempe chain neighborhood for the university timetabling problem. In *Proceedings of the 6th ACIS International Conference on Computer and Information Science (ACIS-ICIS)*, pages 400–405, 2007.
41. Liang Zhang, Wen Luo, Shigang Chen, and Ying Jian. End-to-end maxmin fairness in multihop wireless networks: Theory and protocol. *Journal of Parallel and Distributed Computing*, 72(3):462–474, 2012.

---

## The effect of neighborhood structures on examination timetabling with artificial bee colony

Asaju La'aro Bolaji · Ahamad Tajudin  
Khader · Mohammed Azmi Al-Betar ·  
Mohammed A. Awadallah · J. Joshua  
Thomas

the date of receipt and acceptance should be inserted later

**Abstract** Artificial Bee Colony (ABC) algorithm is among the most effective nature-inspired algorithms for solving the combinatorial optimization problems. In this paper, ABC is adopted for university examination timetabling problems (UETP) using a *defacto* dataset established by Carter et al. (1996). ABC has three main operators that drive the search toward the global minima: employed bee, onlooker bee, and scout. For UETP, the employed bee and onlooker bee operators are manipulated to be workable where three neighborhood structures are employed: move, swap and Kempe chain. The effect of these neighborhood structures on the behaviour of ABC for UETP is studied and analyzed in this paper. The experimental design is intentionally made with various convergence cases of different neighborhood structure. The result suggests that the ABC combined with the three neighborhood structures is an effective method for UETP. Comparative evaluation with previous methods is also provided. The results produced by the proposed method are competitive in

---

Asaju La'aro Bolaji

School of Computer Sciences, Universiti Sains Malaysia, 11800, Pulau Pinang, Malaysia and  
Department of Computer Science, University of Ilorin, Ilorin, Nigeria.  
E-mail: abl10\_sa0739@student.usm.my

Ahamad Tajudin Khader

School of Computer Sciences, Universiti Sains Malaysia, 11800, Pulau Pinang, Malaysia.  
E-mail: tajudin@cs.usm.my

Mohammed Azmi Al-Betar

School of Computer Sciences, Universiti Sains Malaysia, 11800, Pulau Pinang, Malaysia and  
Department of Computer Science, Jadara University, Irbid, Jordan.  
E-mail: mobetar@cs.usm.my

Mohammed A. Awadallah

School of Computer Sciences, Universiti Sains Malaysia, 11800, Pulau Pinang, Malaysia.  
E-mail: mama10\_com018@student.usm.my

J. Joshua Thomas

School of Computer Sciences, Universiti Sains Malaysia, 11800, Pulau Pinang, Malaysia.  
E-mail: joshopever@yahoo.com



comparison with state of the art methods. Theoretically, this study contributes to the examination timetabling community through an ABC template which is both efficient and flexible for UETP.

**Keywords** artificial bee colony · nature-inspired algorithm · examination timetabling problem

## 1 Introduction

Timetabling is an hard scheduling problem faced by many institutions across the globe. Such problem involves assigning a set of events (i.e. courses and exams) to a limited set of resources (i.e. rooms and timeslot), subject to satisfying a set of constraints. The production of a high quality timetabling solution with minimum violations of constraints is one of the major concerns of almost all institutions. Considerable efforts have been exerted by the researchers in the scheduling field to develop an effective techniques to tackle the timetabling problems. Generally, these problems are referred to as NP-hard (Garey and Johnson, 1979) and have been extensively studied in the last three decades (Abramson and Abela, 1991; Burke et al, 1996; Carter and Laporte, 1996; Kostuch, 2005; Burke et al, 2010).

The timetabling problems comes in different forms: educational timetables, nurse scheduling, sport timetables and transportation timetables. The most common example of the educational timetabling problems is the University Course Timetabling Problem (UCTP) and the University Examination Timetabling Problem (UETP) which is the focus of this paper. Both have minor differences in their constraints, where for example, in UCTP, only one course can be assigned to a room at a specific timeslot while for UETP, two or three exams can take place in the same room and timeslot as long as all constraints are met.

The university examination timetabling problem can be defined as the assignment of exams to a limited number of time periods and rooms, subject to a set of hard and soft constraints (Qu et al, 2009b). The main purpose is to generate high-quality timetabling solution that satisfies the hard constraints and reduces the violations of soft constraints as much as possible. A timetable is feasible, if the hard constraints are satisfied and the quality of the timetabling solution is measured by the violations of soft constraints. Examination timetabling problems can be divided into capacitated or uncapacitated with respect to room constraints (Qu et al, 2009b). The uncapacitated examination timetabling problem is the focus of this paper.

Several techniques have been proposed for tackling uncapacitated examination timetabling in the literature, since there is no one single technique that can provide an exact solution (Millar and Kiragu, 1998). One of the most successful methods used to tackle UETP is the meta-heuristic based techniques. These could be divided into local-based search methods as Great Deluge (GD) (Burke et al, 2004; Burke and Newall, 2003; McCollum et al, 2009), Simulated Annealing (Thompson and Dowsland, 1996, 1998), Tabu

Search (Di Gaspero and Schaerf, 2001; White and Xie, 2001; White et al, 2004; Kendall and Hussin, 2005), Variable Neighbourhood Search (VNS) (Ahmadi et al, 2003; Burke et al, 2010)[8] and population-based method such as Ant Colony (Dowland and Thompson, 2004; Eley, 2006), Evolutionary Algorithms (Paquete and Fonseca, 2001; Côté et al, 2005), Particle Swarm Optimization (Fealko and Adviser-Mukherjee, 2006), Harmony Search Algorithm (Al-Betar and Khader, 2008; Al-Betar et al, 2010a,b, 2011b) and memetic algorithms (Burke et al, 1996; Merlot et al, 2003).

A new nature-inspired algorithm called Artificial Bee Colony (ABC) has been recently proposed by Karaboga who was inspired by imitating the intelligent behaviour of honey bee (Karaboga, 2005). It has been successfully applied to a wide variety of optimisation problems as shown in the survey paper (Karaboga and Akay, 2009b).

The ABC as a stochastic search algorithm firstly begins with an initial population stored in Food Source Memory (FSM). At every iteration, new food sources (solutions) are generated from the neighbouring of the existing population using three operators: Employed bee, Onlooker bee and Scout bee. The new food sources are then evaluated against an objective function and replaced the old population, if their fitnesses are better. This process is repeated until the termination criteria is reached.

Defining efficient neighborhood structures that appropriate to the nature of the combinatorial optimization problem is a big challenge that influences the performance of the algorithm (Aladag & Hocaoglu, 2007). Three neighbourhood structures are incorporated into the employed and onlooker operators namely, move, swap and kemp chain. In this study, an extensive analysis of the effect of neighbourhood structure on the behaviour of ABC for UETP is conducted. Then, the effectiveness of ABC with each and combined neighbourhood operators is evaluated on 13 standard benchmark datasets reflecting real-world examination timetabling instances which were introduced by Carter and Laporte (1996). The ABC with three neighbourhood structures achieved comparably competitive results.

The rest of the paper is organized as follows: Section 2 gives descriptions and formulations of the examination timetabling problem while section 3 presents the fundamentals of ABC. Section 4 describes ABC approach for examination timetabling and the neighbourhood structures is presented in section 5. Section 6 provides an explanation to the experimental results while the last section is devoted for conclusion and some future works.

## 2 Problem descriptions and formulations

Tackling the exam timetabling problem involves scheduling a set of exams, each taken by a set of students, to a set of time periods (timeslots) subject to hard and soft constraints. The main objective is to obtain a timetable that satisfies the hard constraint (H1) with the minimum penalty of the soft constraint violation (S1). The hard and soft constraints are as follows:

**Table 1** The symbols used in the description of UETP

Symbols	Definition
$N$	The total number of exams.
$M$	The total number of students.
$P$	The total number of time periods.
$E$	Set of exams
$S$	Set of students
$T$	Set of time periods
$\mathbf{x}$	A timetable solution is given by $(x_1, x_2, \dots, x_M)$ .
$x_i$	The timeslot of exam $i$ .
$a_{i,j}$	Proximity coefficient matrix element : whether the timetable $\mathbf{x}$ is penalized based on the distance between time period of exam $i$ , and time period of exam $j$
	$a_{i,j} = \begin{cases} 2^{5- x_i-x_j } & \text{if } 1 \leq  x_i - x_j  \leq 5 \\ 0 & \text{Otherwise.} \end{cases}$
$u_{i,j}$	Student-exam matrix element: if student $s_i$ is taking exam $j$
	$u_{i,j} = \begin{cases} 1 & \text{if student } i \text{ is sitting for exam } j \\ 0 & \text{Otherwise.} \end{cases}$
$c_{i,j}$	Conflict matrix element: total number of students sharing exam $i$ and exam $j$ .
	$c_{i,j} = \sum_{k=1}^N u_{k,i} \times u_{k,j} \quad \forall i, j \in E$

- H1: no student can sit for two exams simultaneously.
- S1: the exams taken by the same student should be spread out evenly across a timetable.

A detailed description of the problem is summarized by Qu et al (2009b). A timetabling solution is represented by a vector  $\mathbf{x} = (x_1, x_2, \dots, x_M)$  of exams, where the value of  $x_i$  is the timeslot for exam  $i$ . The proximity cost function is used in the evaluation of the timetable by Qu et al (2009b) and refers to the ratio of the penalty assigned to the total number of soft constraint violations and the total number of students. The formulation for the proximity cost function is given in equation (1), while the notation of the variable used is shown in Table 1.

$$\min f(x) = \frac{1}{N} \times \sum_{i=1}^{M-1} \sum_{j=i+1}^M c_{i,j} \times a_{i,j} \quad (1)$$

**H1:** No student can sit for two exams simultaneously

$$x_i \neq x_j \quad \forall x_i, x_j \in X \wedge c_{i,j} \geq 1$$

It is important to note that the value of the proximity cost function  $f(x)$  is referred to as the fitness cost of a feasible timetable (Carter and Laporte, 1996).

The Carter dataset used in this study consists of 13 datasets, which reflect the real-world examination timetabling problems. For the purpose of our study,

**Table 2** Table 2: Characteristics of Uncapacitated Exam Dataset

Dataset	Time-Periods	Exams	Students	Density
CAR-S-91-I	35	682	16,925	0.13
CAR-F-92-I	32	543	18,419	0.14
EAR-F-83-I	24	190	1125	0.27
HEC-S-92-I	18	81	2823	0.42
KFU-S-93	20	461	5349	0.06
LSE-F-91	18	381	2726	0.06
RYE-S-93	23	481	11,483	0.07
STA-F-83-I	13	139	611	0.14
TRE-S-92	23	261	4360	0.18
UTA-S-92-I	35	622	21,266	0.13
UTE-S-92	10	184	2750	0.08
YOR-F-83-I	21	181	941	0.29

12 datasets circulated in the literature were used. The characteristics of Carter datasets, varying in size and complexity, are shown in Table 2. The last column illustrates the density of the conflict matrix, which is the ratio between the number of elements of values  $c_{i,j} > 0$  and the total number of elements in the conflict matrix (Qu et al, 2009b).

### 3 Artificial Bee Colony Algorithm

Artificial Bee Colony (ABC) is a swarm metaheuristic algorithm which was originally introduced in 2005 by Karaboga for tackling numerical optimization problems (Karaboga, 2005). This algorithm is considered a stochastic optimization algorithm based on the model proposed by Teodorović and Del-Orco (2005) for the foraging manners of honey bee in their colonies. The ABC consists of three vital components: employed, unemployed foraging bees, and food sources. The first two components i.e., employed and unemployed forager search for rich food sources, which is the third component. The two principal modes of behaviour which are necessary for self-organization and collective intelligence are also described by the model. In practice, such mode includes the recruitment of foragers to the rich food sources resulting in positive feedback and abandonment of poor food sources by foragers causing negative feedback.

In the colony of ABC there are three groups of bees: employed, onlooker and scout bees. Associated with particular food source is employed bee whose behaviour is studied by the onlookers to select the desired food source while the scout bee searches for new food sources randomly once it is exhausted. Both onlookers and scouts are considered as unemployed foragers. The position of a food source in ABC corresponds to the possible solution of the problem to be optimized and the nectar amount of a food source represents the fitness (quality) of the associated solution. The number of employed bees is equal to the number of food sources (solutions), since each employed bee is associated with one and only one food source (Karaboga, 2005).

The key phases of the algorithm as proposed by Karaboga and Akay (2009a) are as follows:

- Generate the initial population of the food sources randomly.
- REPEAT
  - Send the employed bees onto the food sources and calculate the fitness cost.
  - Evaluate the probability values for the food sources
  - Send the onlooker bees onto the food sources depending on probability and calculate the fitness cost.
  - Abandon the exploitation process, if the sources are exhausted by the bees.
  - Send the scouts into the search area for discovering new food sources, randomly
  - Memorize the best food source found so far.
- UNTIL (requirements are met).

#### 4 Artificial Bee Colony for UETP

The implementation of ABC for the UETP includes the followings six steps:

1. **Step 1: Initialization of the ABC and UETP parameters:**  
 The parameters of UETP are normally extracted from the problem instances. These parameters include the set of exams, the set of timeslots, the set of rooms, etc. The main decision variable of UETP is the exams. Each exam can be assigned to a feasible timeslot in the timetable solution. A set of all feasible timeslots can be considered as the available range of such exams. In fact, the feasible timeslot of each exam changes during the search in the neighbourhood of ABC. The proximity cost function described in (1) is used to evaluate each solution. Here, the parameters of the ABC used for UETP are initialized. That is, the Solution Number (SN) which is similar to the population size in genetic algorithms; Maximum Cycle Number (MCN) which is similar to the number of iterations and Limit which determine when a solution will be abandoned. These parameters will be explained in more detail in the next steps.
2. **Step 2: Initialize the Food Source Memory (FSM):**  
 The Food Source Memory (FSM) is an augmented matrix of size  $SN$  comprising a vector in each row representing a timetable solution as in (2). Note that the vectors in FSM are generated with a method that combines the saturation degree (SD) and backtracking algorithms as used previously (Al-Betar et al, 2010b; Bolaji et al, 2011). Here, SD starts with an empty timetable, where the exam with the least number of valid timeslots in the scheduled list is assigned first. The next selected exam to be scheduled is based on the number of available timeslots; where some exams may not be assigned because of non-availability of the timeslots, then the backtracking algorithm (BA) is applied to re-assign unscheduled exams. The process, SD and BA, is repeated several times until all exams are assigned to feasible timeslots. Using this techniques, the feasibility of all the solutions is guaranteed and sorted in ascending order according to their objective function

values.

$$\mathbf{FSM} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{SN} & x_2^{SN} & \cdots & x_N^{SN} \end{bmatrix} \begin{bmatrix} f(\mathbf{x}^1) \\ f(\mathbf{x}^2) \\ \vdots \\ f(\mathbf{x}^{SN}) \end{bmatrix} \quad (2)$$

**3. Step 3: Send the employed bees to the food sources:**

Here, the employed bee operator selects a timetabling solution from the population one by one and applies the three neighbourhood structures to generate new solutions. The fitness of each offspring solution is calculated. If it is better than that of parent solution, then the offspring replaces the parent in FSM. This process is iteratively repeated until all solutions have been explored (see algorithm: 1 for details). where  $\mathbf{x}^i$  is the solution and

---

**Algorithm 1** Employed Bee Phase

---

```

1: for  $i = 1 \dots SN$  do
2:    $rand \in \{1, 2, 3\}$ 
3:   if  $rand = 1$  then
4:      $\mathbf{x}^{i(new)} = Move(\mathbf{x}^i)$ 
5:   else
6:     if  $rand = 2$  then
7:        $\mathbf{x}^{i(new)} = Swap(\mathbf{x}^i)$ 
8:     else
9:       if  $rand = 3$  then
10:         $\mathbf{x}^{i(new)} = Kempe(\mathbf{x}^i)$ 
11:       end if
12:     end if
13:   end if
14:   if  $\mathbf{x}^{i(new)}$  is better than  $\mathbf{x}^i$  then
15:      $\mathbf{x}^i = \mathbf{x}^{i(new)}$ 
16:   end if
17: next  $i$ 
18: end for

```

---

$\mathbf{x}^{i(new)}$  is the new neighbouring solution.

**4. Step 4: Send the onlooker bees:**

The onlooker bee has the same food sources (timetabling solutions) of the employed bees. It initially calculates the selection probability of each food source generated by the employed bee in the previous step. The fittest food sources are selected by the onlooker using roulette wheel selection mechanism. The process of selection in the onlooker phase works as follows:

- Assign to each employee bee a selection probability as follows:

$$p_j = \frac{f(\mathbf{x}^j)}{\sum_{k=1}^{SN} f(\mathbf{x}^k)}$$

Note that the  $\sum_{i=1}^{SN} p_i$  is unity.

- The onlooker sample the fitness of the food source of each employed bee and selects the one with highest probability. It then adjust the selected food source to its neighbourhood using the same strategy as the employed bee. The fitness of the new solution is calculated and if it is better it replaces the current one.
5. **Step 5: Send the Scout to search for possible new food sources:**  
This is known to be the colony explorer. It works once a solution is abandoned, i.e. if a solution in the FSM has not improved for certain number of iterations. ABC generates a new solution randomly and substitutes the abandoned one. Memorize the fitness of the best food source  $x_{best}$  found so far in FSM.
  6. **Step 6: Stopping Criteria:**  
Steps 3 to 5 are repeated until a stop criterion is met. This is originally determined using MCN value.

## 5 Neighbourhood Structure (NS)

In this section, the neighbourhood structures (Move, swap and kempe chain) used in the employed and onlooker phases given in section 4 shall be described in details. Neighbourhood structure (NS) is a commonly used technique in solving timetabling problems. NS can be used to obtain a new set of neighbor solutions by applying a small perturbation to a given solution and each neighbourhood solution is reached immediately from a given solution by a move (Glover and Laguna, 1998). The neighbourhood structure begins with an initial solution and progressively explores the neighborhood of the solution for improvement. Thus, the current solution is iteratively replaced by one of its neighbors (often improving) until a specific stopping condition is met (Lü et al, 2011). The ABC operators such as employed and onlooker bees, use three different neighbourhood structures to explore the solution space thoroughly in order to enhance the quality of the solution and thus reduce the redundancy or ineffectiveness of using a particular type alone. The three neighbourhood structures are: move, swap, and kempe chain. They have been used by other researchers and proven to be very efficient for exam timetabling problems (Al-Betar et al, 2010a; Thompson and Dowsland, 1998; Burke et al, 2010).

- **Neighbourhood Move (NM):** moves selected exam to a feasible period and room randomly i.e. replace the time period  $x'_i$  of exam  $i$  by another feasible timeslot.
- **Neighbourhood Swap (NS):** swap two selected exams at random i.e. select exam  $i$  and event  $j$  randomly, swap their time periods  $(x'_i, x'_j)$ .
- **Neighbourhood Kempe Chain (NK):** Firstly, select the timeslot  $x'_i$  of exam  $i$  and randomly select another  $q'$  timeslot. Secondly, all exams that have the same timeslot  $x'_i$  that are in conflict with one or more exams timetabled in  $q_i$  are entered to chain  $\delta$  where  $\delta = \{j | x'_j = x'_i \wedge t_{i,q'} = 0 \wedge \forall j \in E\}$ . Thirdly, all exams that have the same timeslot  $q'$  that are conflicting with one or more exams timetabled in  $x'_i$  are entered to a chain

$\delta'$  where  $\delta' = \{k | x'_k = q' \wedge t_{k,x'_i} = 0 \wedge \forall k \in E\}$  and Lastly, simply assign the exams in  $\delta$  to  $q'$  and the exams in  $\delta'$  to  $x'_i$ .

## 6 Experimental Results and Analysis

The method is coded in Microsoft Visual C++ 6.0 on Windows 7 platform on Intel 2 GHz Core 2 Quad processor with 2 GB of RAM. The ABC required a maximum of 7 hours to obtain the recorded result, although the computational time is not provided in the literature. The parameters used for ABC is as follows: MCN=10,000; SN=10; limit=100.

In this section, the effectiveness of neighborhood structure on the performance of ABC-based UETP is experimentally studied using the Carter dataset. Seven convergence cases are run where each representing a version of ABC combined single, double or triple combinations of the neighborhood structures within the employee and onlooker bees operators as shown in Table 3. For example, case 1 is ABC version with a single move combined with employee and onlooker bees. Apparently, all possible combinations of the neighborhood structures are studied separately.

Each convergence case is ran ten times. The best result amongst the ten runs of each case is recorded in Table 4 for each Carter dataset. Numbers in Table 4 refer to the penalty value of the soft constraint violations. (lowest is best). The best solution achieved by any version of ABC is highlighted in bold.

Generally, the ABC combined with the three neighborhood structures (i.e., case 7) has a better performance than all other cases that combine single or double neighborhood structures. Furthermore, case 5 that combines MOVE and KEMPE is able to compete with case 7. This shows the efficiency of combining these neighborhood structures with ABC for UETP. Apparently, the performance of case 3 is better than case 1 and 2, in terms of the solution quality in almost all the instances and with little difference between the 2 cases (case 1, 2, 3 combined single neighborhood structure). However, with different combinations of these neighborhoods, the efficiency of ABC is clearly improved with further reduction in the proximity cost function as shown from cases 4 to 6. A plausible observation can be set as the combination of two or more neighborhood structures within ABC-based UETP enhances the search capability and therefore an impressive result is obtained. It is worth mentioning that each neighborhood structure is able to navigate the UETP search space in a way different from the others. As such, the selection of an efficient neighborhood structure is inevitably required to achieve superior results.

Table 5 and 6 showed the penalty value achieved by the proposed method in comparison with those provided by some state of the art techniques and best known results as given by (Qu et al, 2009b). This includes a total of 10 comparative methods comprising metaheuristic-based methods, Heuristic and Hyper-Heuristic Methods. The key for the comparative methods is as follows:

M1: Graph-Based Hyper-Heuristic (Burke et al, 2007).

M2: Graph-Based Hyper-Heuristic (Qu et al, 2009a).



**Table 3** Cases studied the effect of neighborhood structure on ABC-based UETP

CASE	MOVE	SWAP	KEMPE
Case 1	✓	X	X
Case 2	X	✓	X
Case 3	X	X	✓
Case 4	✓	✓	X
Case 5	✓	X	✓
Case 6	X	✓	✓
Case 7	✓	✓	✓

M3: Graph-Based Hyper-Heuristic (Qu and Burke, 2008).

M4: Graph-Based Hyper-Heuristic (Pillay and Banzhaf, 2009).

M5: Fuzzy Multiple Heuristic Orderings (Asmuni et al, 2009).

M6: Harmony Search Algorithm (Al-Betar et al, 2010b).

M7: Ant Algorithms (Eley, 2006).

M8: Multi-Objective Evolutionary Algorithm (Côté et al, 2005).

M9: An integrated hybrid approach by (Turabieh and Abdullah, 2011).

M10: A hybrid Variable Neighbourhood Search with Genetic Algorithm (Burke et al, 2010).

As shown in table 5 and 6, it can be seen that ABC algorithm produce comparable results. The best penalty values (lowest is best) are highlighted in bold, while '+' indicates that the method could not find a feasible timetable. In general, it is able to achieve very close to the best results.

**Table 4** Experimental Results

Dataset	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
CAR-S-91-I	6.79	6.50	6.86	6.48	5.70	5.99	<b>5.38</b>
CAR-F-92-I	5.71	5.43	5.64	5.39	4.75	4.66	<b>4.61</b>
EAR-F-83-I	46.37	47.60	42.79	44.79	38.83	39.18	<b>38.58</b>
HEC-S-92-I	13.54	14.99	11.81	14.02	11.34	15.98	<b>11.17</b>
KFU-S-93	18.49	16.88	17.14	17.35	15.04	15.00	<b>14.89</b>
LSE-F-91	14.88	14.55	15.38	13.94	12.19	12.17	<b>11.74</b>
RYE-S-93	12.18	11.97	13.56	12.12	10.11	9.91	<b>9.80</b>
STA-F-83-I	161.35	162.77	158.87	161.62	157.30	157.42	<b>157.21</b>
TRE-S-92	11.04	10.93	10.25	10.03	9.26	9.14	<b>8.96</b>
UTA-S-92-I	4.52	4.46	4.49	4.44	3.81	3.82	<b>3.65</b>
UTE-S-92	30.44	29.52	31.43	27.46	27.88	27.43	<b>26.89</b>
YOR-F-83-I	45.61	48.55	43.88	45.21	40.43	39.84	<b>39.34</b>

## 7 Conclusion

In order to tackle the university examination timetabling problems (UETP), Artificial Bee Colony has been presented using a *defacto* dataset established by Carter et al. (1996). Three main operators in ABC are able to guide the search toward the global optima: employee bee, onlooker bee, and scout. For

**Table 5** Comparison with previous Methods

Dataset	ABC	M1	M2	M3	M4	M5	Best Known Results
CAR-S-91-I	5.38	5.36	5.11	5.16	4.97	5.29	<b>4.5</b>
CAR-F-92-I	4.61	4.93	4.32	4.16	4.84	4.54	<b>3.9</b>
EAR-F-83-I	38.58	37.92	35.56	35.86	36.86	37.02	<b>29.3</b>
HEC-S-92-I	11.17	12.25	11.62	11.94	11.85	11.78	<b>9.2</b>
KFU-S-93	14.89	15.2	15.18	14.79	14.62	15.8	<b>13.0</b>
LSE-F-91	11.74	11.33	11.32	11.15	11.14	12.09	<b>9.6</b>
RYE-S-93	9.80	+	+	+	9.65	10.38	<b>6.8</b>
STA-F-83-I	157.21	158.19	158.88	159	158.33	160.42	<b>156.9</b>
TRE-S-92	8.96	8.92	8.52	8.6	8.48	8.67	<b>7.88</b>
UTA-S-92-I	3.65	3.88	3.21	3.59	3.4	3.57	<b>3.14</b>
UTE-S-92	26.89	28.01	28	28.3	28.88	28.07	<b>24.4</b>
YOR-F-83-I	39.34	41.37	40.71	41.81	40.74	39.8	<b>34.9</b>

**Table 6** Comparison with previous Methods

Dataset	ABC	M6	M7	M8	M9	M10	Best Known Results
CAR-S-91-I	5.38	4.99	5.4	5.2	4.80	4.6	<b>4.5</b>
CAR-F-92-I	4.61	4.29	4.2	4.3	4.10	<b>3.9</b>	<b>3.9</b>
EAR-F-83-I	38.58	34.42	34.2	36.8	34.92	32.8	<b>29.3</b>
HEC-S-92-I	11.17	10.40	10.4	11.1	10.73	10.0	<b>9.2</b>
KFU-S-93	14.89	13.5	14.3	14.5	<b>13.0</b>	<b>13.0</b>	<b>13.0</b>
LSE-F-91	11.74	10.48	11.3	11.3	10.01	10.0	<b>9.6</b>
RYE-S-93	9.80	8.79	8.8	9.8	9.65	+	<b>6.8</b>
STA-F-83-I	157.21	157.04	158.03	157.3	158.26	<b>156.9</b>	<b>156.9</b>
TRE-S-92	8.96	8.16	8.6	8.6	<b>7.88</b>	7.9	<b>7.88</b>
UTA-S-92-I	3.65	3.43	3.5	3.5	3.20	3.2	<b>3.14</b>
UTE-S-92	26.89	25.09	25.3	26.4	26.11	24.8	<b>24.4</b>
YOR-F-83-I	39.34	35.86	36.4	39.4	36.22	<b>34.9</b>	<b>34.9</b>

UETP, the employee bee and onlooker bee operators are redefined to hold three neighborhood structures: move, swap and Kempe chain. The influence of these neighborhood structures on the behaviour of ABC for UETP is studied and analyzed in this paper.

The experimental design is intentionally made with various convergence cases of different neighborhood structures. The result suggests that the ABC combined with the three neighborhood structures is an effective method for UETP. Comparative evaluation with previous methods is also provided. The results produced by the proposed method are competitive in comparison with the state of the art methods.

The main contribution of this study is to provide the examination timetabling community with an ABC template which combines both efficiency and flexibility for tackling UETP.

In view of the fact that, ABC-based UETP combined with various neighborhood structures has been proved to be very efficient, future work can improve the ABC-based UETP method by:

- Hybridizing ABC with other gradient descent methods to improve its exploitation.
- Studying the suitable parameters for ABC-based UETP.
- Investigating other efficient neighborhood structures.
- Combining different selection schemes in onlooker bee phase such as, linear rank, exponential rank, tournament selection and many others (Al-Betar et al, 2011a).
- Investigating the performance of the proposed ABC Technique using the third track dataset of the 2nd International Timetabling Competition (ITC-2007) presented by (McCollum et al, 2010)

### Acknowledgement

This research was supported by the Universiti Sains Malaysia under IPS-Graduate Fellowship Scheme, 2011 awarded to the first author and CS-USM Postdoctoral Research Fellowship awarded to the third author.

### References

- Abramson D, Abela J (1991) A parallel genetic algorithm for solving the school timetabling problem. In: in proceedings of the Fifth Australian Computer Science Conference (ACSC-15), Volume 14, pp 1–14
- Ahmadi S, Barone R, Cheng P, Cowling P, McCollum B (2003) Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. Proceedings of Multidisciplinary International Scheduling: Theory and Applications (MISTA 2003), Nottingham, August pp 13–16
- Al-Betar M, Khader A (2008) A harmony search algorithm for university course timetabling. *Annals of Operations Research* DOI: 101007/s10479-010-0769-z, pp 1–29
- Al-Betar M, Khader A, Nadi F (2010a) Selection mechanisms in memory consideration for examination timetabling with harmony search. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM, pp 1203–1210
- Al-Betar M, Khader A, Thomas JJ (2010b) A combination of metaheuristic components based on harmony search for the uncapacitated examination timetabling. *Practice and Theory of Automated Timetabling (PATAT 2010)* Belfast, North Ireland pp 57–80
- Al-Betar M, Doush I, Khader A, Awadallah M (2011a) Novel selection schemes for harmony search. *Applied Mathematics and Computation* 218((10):6095–6117
- Al-Betar MA, Khader AT, Zaman M (2011b) University Course Timetabling Using a Hybrid Harmony Search Metaheuristic Algorithm. *IEEE Transactions on Systems, Man, and Cybernetics — Part C: Applications and Reviews* DOI: 101109/TSMCC20112174356
- Asmuni H, Burke E, Garibaldi J, McCollum B, Parkes A (2009) An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers & Operations Research* 36(4):981–1001
- Bolaji A, Khader A, Al-Betar M, Awadallah M (2011) An improved artificial bee colony for course timetabling. In: *Bio-Inspired Computing: Theories and Applications (BIC-TA)*, 2011 Sixth International Conference on, IEEE, pp 9–14
- Burke E, Newall J (2003) Enhancing timetable solutions with local search methods. In Edmund Burke and Patrick De Causmaecker, editors, *Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 2740 Springer-Verlag, Berlin pp 195–206
- Burke E, Newall J, Weare R (1996) A memetic algorithm for university exam timetabling. In: *First International Conference on the Practice and Theory of Automated Timetabling*, Springer, pp 241–250
- Burke E, Bykov Y, Newall J, Petrovic S (2004) A time-predefined local search approach to exam timetabling problems. *IIE Transactions* 36(6):509–528

- Burke E, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research* 176(1):177–192
- Burke E, Eckersley A, McCollum B, Petrovic S, Qu R (2010) Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research* 206(1):46–53
- Carter M, Laporte G (1996) Recent developments in practical examination timetabling. In Edmund Burke and Patrick De Causmaecker, editors, *Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 1153 Springer-Verlag, Berlin pp 3–21
- Côté P, Wong T, Sabourin R (2005) A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. *Practice and Theory of Automated Timetabling V* pp 294–312
- Di Gaspero L, Schaerf A (2001) Tabu search techniques for examination timetabling. In Edmund Burke and Patrick De Causmaecker, editors, *Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 2079 Springer-Verlag, Berlin pp 104–117
- Dowland K, Thompson J (2004) Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society* 56(4):426–438
- Eley M (2006) Ant algorithms for the exam timetabling problem. In: *Proceedings of the 6th international conference on Practice and theory of automated timetabling VI*, Springer-Verlag, pp 364–382
- Fealko D, Adviser-Mukherjee S (2006) Evaluating particle swarm intelligence techniques for solving university examination timetabling problems. A Dissertation for the degree of Doctor of Philosophy, Graduate School of computer and Information Sciences, Nova Southeastern University.
- Garey M, Johnson D (1979) *Computers and intractability. A guide to the theory of NP-completeness*. A Series of Books in the Mathematical Sciences. WH Freeman and Company, San Francisco, Calif
- Glover F, Laguna M (1998) *Tabu search*, vol 1. Kluwer Academic Pub
- Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Techn Rep TR06, Erciyes Univ Press, Erciyes
- Karaboga D, Akay B (2009a) A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation* 214(1):108–132
- Karaboga D, Akay B (2009b) A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review* 31(1):61–85
- Kendall G, Hussin N (2005) An investigation of a tabu-search-based hyper-heuristic for examination timetabling. *Multidisciplinary Scheduling: Theory and Applications* pp 309–328
- Kostuch P (2005) The university course timetabling problem with a three-phase approach. In: Burke E, Trick M (eds) *Practice and Theory of Automated Timetabling (PATAT) V*, vol 3616 pp 109–125
- Lü Z, Hao J, Glover F (2011) Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics* 17(2):97–118
- McCollum B, McMullan P, Parkes A, Burke E, Abdullah S (2009) An extended great deluge approach to the examination timetabling problem. *Proceedings of the 4th Multidisciplinary International Scheduling: Theory and Applications 2009 (MISTA 2009)* pp 424–434
- McCollum B, Schaerf A, Paechter B, McMullan P, Lewis R, Parkes A, Gaspero L, Qu R, Burke E (2010) Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing* 22(1):120–130
- Merlot L, Boland N, Hughes B, Stuckey P (2003) A hybrid algorithm for the examination timetabling problem. In Edmund Burke and Patrick De Causmaecker, editors, *The Practice and Theory of Automated Timetabling Lecture Notes in Computer Science* 2740 Springer-Verlag, Berlin, pp 207–231
- Millar H, Kiragu M (1998) Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. *European journal of operational research* 104(3):582–592
- Paquete L, Fonseca C (2001) A study of examination timetabling with multiobjective evolutionary algorithms. In: *4th Metaheuristics International Conference (MIC 2001)*, pp 149–154

- Pillay N, Banzhaf W (2009) A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. *European Journal of Operational Research* 197(2):482–491
- Qu R, Burke E (2008) Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society* 60(9):1273–1285
- Qu R, Burke E, McCollum B (2009a) Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research* 198(2):392–404
- Qu R, Burke E, McCollum B, Merlot L, Lee S (2009b) A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling* 12(1):55–89
- Teodorović D, DellOrco M (2005) Bee colony optimization—a cooperative learning approach to complex transportation problems. In: *Advanced OR and AI Methods in Transportation*. Proceedings of the 10th Meeting of the EURO Working Group on Transportation, Poznan, Poland, Citeseer, pp 51–60
- Thompson J, Dowsland K (1996) Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research* 63(1):105–128
- Thompson J, Dowsland K (1998) A robust simulated annealing based examination timetabling system. *Computers & Operations Research* 25(7-8):637–648
- Turabieh H, Abdullah S (2011) An integrated hybrid approach to the examination timetabling problem. *Omega* 39(6):589–607
- White G, Xie B (2001) Examination timetables and tabu search with longer-term memory. In: Burke E and Erbens W, (eds) *The Practice and Theory of Automated Timetabling*. Lecture Notes in Computer Science, Vol 2079 Springer-Verlag, Berlin, pp 85–103
- White G, Xie B, Zonjic S (2004) Using tabu search with longer-term memory and relaxation to create examination timetables. *European Journal of Operational Research* 153(1):80–91

---

# A matheuristic approach to the shift minimisation personnel task scheduling problem

Pieter Smet · Greet Vanden Berghe

Received: date / Accepted: date

**Abstract** In the context of personnel rostering, several levels of granularity have been discussed. Typically, these levels range from very coarse grained (e.g. days-off scheduling) to more finely granulated (e.g. tour scheduling). However, in some cases a more detailed type of assignment is required. Not only shifts need to be assigned to personnel, but also the allocation of tasks is incorporated in the optimisation of the rosters. The present paper introduces a matheuristic approach based on local search for the subproblem of assigning tasks to a set of multi-skilled employees whose working times are already determined. Experimental results show that the presented algorithm is capable of finding new best solutions for the benchmark instances.

**Keywords** Integrated personnel rostering · Personnel task scheduling · Matheuristic · Local search · Constructive heuristic

## 1 Introduction

After several years of research, personnel rostering remains a relevant academic timetabling subject. As personnel costs have become the major part of operational expenses, it is ever so important to try and organise a given workforce as efficiently as possible in order to reduce the associated costs and to increase employee satisfaction.

In the personnel scheduling literature, assigning shifts to personnel is often the most fine-grained level at which the allocation is being discussed, even

---

Pieter Smet  
CODeS, KAHO Sint-Lieven  
Gebroeders De Smetstraat 1, 9000 Gent, Belgium  
E-mail: pieter.smet@kahosl.be

Greet Vanden Berghe  
CODeS, KAHO Sint-Lieven  
Department of Computer Science, KU Leuven  
E-mail: greet.vandenbergh@kahosl.be

though some authors do incorporate elements at a more detailed level (Ernst et al 2004). For example, Beer et al (2008) discuss the problem of assigning breaks in shifts. The shifts are already assigned, and breaks need to be planned such that various restrictions and requirements are met.

The assignment of particular tasks to employees during a shift is often not incorporated in the construction of rosters. In some cases, employees know automatically which tasks to perform during working hours. This is often the case in hospitals, where nurses know exactly what they are supposed to do in each shift (Burke et al 2004). However, in other cases tasks are assigned to employees in an ad hoc manner, often resulting in unnecessary usage of resources. Therefore it is recommendable to incorporate the assignment of tasks in the construction of rosters for employees, in order to reduce operational expenses while maintaining a high quality of service.

The practical relevance of this problem is apparent in various contexts. In the food production industry, due to the short batches being produced, employees have to perform tasks on more than one machine during one shift. In order to create efficient solutions, it is necessary to integrate the assignment of shifts and tasks resulting in an structured problem combining personnel rostering and task allocation.

The aforementioned problem has only been addressed by a few authors. Meisels and Schaerf (2003) discuss a general class of employee timetabling problems in which, during each shift, tasks need to be assigned to employees. No temporal details concerning the tasks are incorporated in the assignment, only the required number of employees for each task in each shift is given. Detienne et al (2009) present two cut generation based approaches for another employee timetabling problem. Their problem contains two decision stages. First the working times of employees are determined. Second, for each employee and for each working period, the used qualification of the employee is decided on. Guyon et al (2010) include scheduling decisions concerning specific activities. The integrated employee timetabling and production scheduling problem is solved using both exact and heuristic approaches including Benders decomposition and a cut generation approach based on the work of Detienne et al (2009). The system presented by Dowling et al (1997) is developed for rostering employees and assigning tasks to employees at an international airport. First, the personnel rostering problem is solved heuristically for a long scheduling period (35 days). Afterwards individual tasks are allocated to the available employees on a day-to-day basis.

Burke et al (2006) discuss a related personnel rostering problem in which the coverage requirements are not specified per shift type, but in terms of time intervals. The time interval coverage requirements are translated to shift type coverage constraints. A metaheuristic is then used to solve the resulting personnel rostering problem in an efficient way.

In the present paper we discuss a solution approach to the shift minimisation personnel task scheduling problem (SMPTSP), originally introduced by Krishnamoorthy and Ernst (2001). The problem considers assigning tasks to multi-skilled employees, while minimising the number of employees used.

Figure 1 shows an example of a solution for an SMPTSP instance, in which 60 tasks are assigned to 25 employees over a one day period. Krishnamoorthy et al (2011) present a Lagrangean relaxation based approach that combines two problem specific heuristics. They have found feasible solutions for 135 out of 137 instances. For a large number of these instances their algorithm is capable of finding the optimal solution. Furthermore, they discuss some interesting properties of the problem and present algorithms that can be used to solve particular subproblems of the SMPTSP.

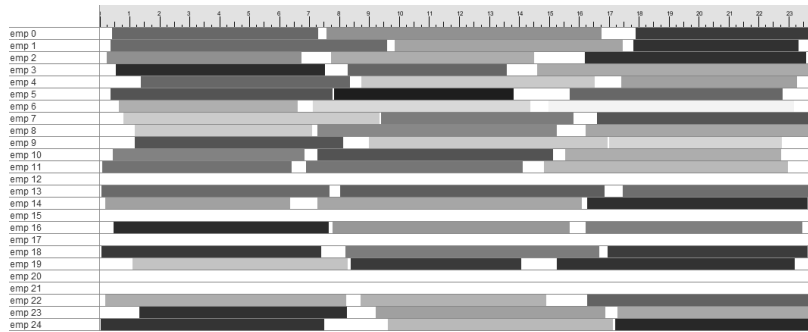


Fig. 1: Optimal solution for an SMPTSP instance.

The approach presented in this paper is a very large-scale neighbourhood search algorithm in which neighbouring solutions are reached by solving a heuristically selected subproblem to optimality. This solution approach is based on the principles of matheuristics (Maniezzo et al 2009), in which (meta)heuristics and exact approaches are combined to exploit the strengths of both solution techniques.

Della Croce and Salassa (2010) describe a matheuristic based on a variable neighbourhood search for a real world nurse rostering problem. Different neighbourhoods are searched by including additional constraints. These constraints fix particular variables which are selected heuristically. Computational results show that this matheuristic approach significantly outperforms exact commercial general purpose solvers. Matheuristic approaches have been applied in various other contexts such as vehicle routing (Doerner and Schmid 2010), permutation flow shop scheduling (Della Croce et al 2011) and the multidimensional knapsack problem (Hanafi et al 2010).

The rest of the paper is organised as follows. The problem definition is provided in Section 2. In Section 3 the solution approach is presented. Details are provided for both a constructive heuristic and the hybrid improvement heuristic. The experimental setup and results are discussed in Section 4. Section 5 concludes the paper and presents future work.



## 2 Problem definition

Let  $J = \{1, \dots, n\}$  be the set of tasks to be assigned to employees and  $W = \{1, \dots, m\}$  the set of employees. Each task  $j \in J$  has a duration  $d_j$ , a start time  $s_j$  and a finish time  $f_j = s_j + d_j$ . Each employee  $w \in W$  has a set of tasks  $T_w \subseteq J$  which he/she can perform. Similarly, there exists a set  $P_j \subseteq W$  for each task  $j \in J$  which contains all employees that can perform task  $j$ . Both  $T_w$  and  $P_j$  are defined based on required qualifications and time windows.

An interval graph  $G = (J, A)$  can be defined with  $J$  the set of nodes and  $A$  the set of arcs. Two nodes  $i$  and  $j$  are connected when their respective time intervals,  $[s_i, f_i]$  and  $[s_j, f_j]$ , overlap. The set of maximal cliques in the interval graph is defined as  $C$ . For interval graphs, this set can be found in polynomial time by first sorting the nodes based on start time and then applying a forward pass algorithm. The set  $C = \{K_1, \dots, K_t\}$  consists of sets  $K_t \subseteq J$  such that any pair of tasks in  $K_t$  overlaps in time and  $K_t$  is maximal. There are no tasks in  $J \setminus K_t$  which overlap with any of the tasks in  $K_t$ . In terms of the SMPTSP, it is obvious that overlapping tasks, represented by nodes in  $K_t$ , should be assigned to different employees. For each employee  $w \in W$ , the set of maximal cliques  $C^w = \{K_1^w, \dots, K_t^w\}$  is constructed in the same way as  $C$ , but for  $C^w$ , only the set of tasks for which the employee is qualified is considered. An employee  $w$  can only be assigned to one of the tasks from each set  $K_t^w \in C^w$ . This ensures that there are no overlapping assignments in a solution.

To solve the SMPTSP, a feasible solution has to be found in which all tasks in  $J$  are assigned to qualified employees from  $W$  in a non-preemptive manner, while minimising the number of workers used.

Two sets of decision variables are defined for the mathematical model:

$$x_{jw} = \begin{cases} 1 & \text{if task } j \in J \text{ is assigned to employee } w \in W \\ 0 & \text{otherwise} \end{cases}$$

$$y_w = \begin{cases} 1 & \text{if employee } w \in W \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

The SMPTSP can now be defined as follows (Krishnamoorthy et al 2011):

$$\min \sum_{w \in W} y_w \quad (1)$$

$$s.t. \sum_{w \in P_j} x_{jw} = 1 \quad \forall j \in J \quad (2)$$

$$\sum_{j \in K_t^w} x_{jw} \leq y_w \quad \forall w \in W, K_t^w \in C^w \quad (3)$$

$$0 \leq y_w \leq 1 \quad \forall w \in W \quad (4)$$

$$x_{jw} \in \{0, 1\} \quad \forall j \in J, w \in W \quad (5)$$

The objective function (1) states that the number of used employees should be minimised. Constraints (2) ensure that each task is only performed by one employee, and that no infeasible assignments in terms of qualifications are

made. Constraints (3) make sure that tasks are only assigned to employees who are active in the final solution. Furthermore, these constraints do not allow for overlapping tasks to be assigned to an employee. Finally, constraints (4) and (5) set bounds for the decision variables.

The SMPTSP can be seen as an application of list colouring on interval graphs, which is NP-complete. Colours correspond to employees and vertices correspond to tasks. The qualifications of the employees are represented by the list of feasible colours on each vertex. Other applications of list colouring on interval graphs include room allocation (Carter and Tovey 1992) and register assignment (Zeitlhofer and Wess 2003).

A class of problems similar to the SMPTSP are the interval scheduling problems (Kolen et al 2007). Here, a set of jobs with fixed start and end times are given as well as a set of machines that can process the jobs. The goal is to decide which jobs to assign and to which machines, while e.g. maximising the value of the assigned jobs. The difference between the basic interval scheduling problem and the SMPTSP lies in the fact that in the SMPTSP not all machines (employees) are qualified for all jobs, and that machines (employees) are not always available. Furthermore, all tasks should be assigned in a feasible solution for the SMPTSP.

### 3 A hybrid heuristic approach

We present a hybrid heuristic local search algorithm for the SMPTSP, based on the principles of matheuristics. The solution approach is hybrid in the sense that neighbouring solutions are reached by solving a randomly selected part of the problem to optimality using a general purpose solver. Details on the local search procedure and the explored neighbourhood are given in Section 3.2. To ensure feasibility of the final solution, the algorithm remains in the feasible search space during the length of the search. A constructive heuristic, described in Section 3.1, is designed to provide a feasible initial solution.

#### 3.1 Constructive heuristic

Krishnamoorthy et al (2011) state that when the qualification constraints are relaxed, i.e. when all employees are qualified to perform all tasks, the resulting problem can be solved in polynomial time. For this purpose, they describe a forward pass maximal clique algorithm on an interval graph (Gupta et al 1979). This algorithm assigns all tasks in order of increasing starting time, using, if possible, an employee who already has tasks assigned. This characteristic of the SMPTSP is incorporated in our constructive heuristic by sorting all tasks  $j \in J$  on start time  $s_j$  in ascending order. Ties are broken by taking into account the qualifications of employees. For this purpose, the tasks are additionally sorted based on the number of qualified personnel able to perform them, also in ascending order. This results in an ordering in which tasks with the smallest

number of feasible personnel are before others. These highly constrained tasks, which are the most difficult to assign, will then be assigned first.

An additional mechanism is introduced to ensure that the constructive heuristic finds feasible solutions in those cases where tasks can only be performed by a limited number of employees. Whenever there is a task  $j$  which cannot be assigned to a qualified employee due to other overlapping tasks, a qualified employee is randomly selected and his/her assigned tasks overlapping with  $j$  are removed. Task  $j$  is then assigned to this employee and the removed tasks are assigned to other employees.

Algorithm 1 shows pseudo code of the constructive heuristic.

---

**Algorithm 1** Constructive heuristic
 

---

$s_j$  := Start time of task  $j \in J$

$O_j$  := Jobs overlapping with task  $j$

$P_j$  := Employees qualified for task  $j$

$R_e$  := Tasks assigned to employee  $e$

Order all  $j \in J$  by  $(s_j + |P_j|)$  in ascending order

**while**  $J \neq \emptyset$  **do**

  Remove task  $j$  from the first position in  $J$

  Assign  $j$  to the first feasible employee

**if** Cannot feasibly assign  $j$  **then**

    Select random employee  $e \in P_j$

    Remove the conflicting tasks  $O_j$  from  $R_e$

    Assign  $j$  to employee  $e$

    Add the previously removed tasks  $O_j$  to the list of tasks to be assigned  $J$

**end if**

**end while**

---

Experiments performed on realistic problems from literature (Section 4.2) and problems based on real world data provided by an industrial partner <sup>1</sup>, show that Algorithm 1 is capable of finding feasible solutions for problems with realistic dimensions. Algorithm 1 will not terminate if an instance has no feasible solution. To resolve this issue, additional mechanisms could be added to the algorithm which e.g. do not assign all visits or include dummy employees in the set of available employees.

### 3.2 Matheuristic based local search

Typically, the initial solution can still be improved. For this purpose, an improvement procedure based on local search is used.

To ensure feasibility throughout the search trajectory of the algorithm only feasible neighbouring solutions are considered. These are reached by randomly selecting  $k$  employees and optimally solving the subproblem composed of them and their assigned jobs, using a general purpose solver. The initial solution is

---

<sup>1</sup> SAGA Consulting

feasible and therefore solutions in this neighbourhood are also always feasible, thus forcing the local search only to explore the feasible search space.

Figure 2 illustrates the move used to reach new solutions. A gray roster indicates employees that are not considered by the move, i.e. the assignments of these employees remain unchanged during the move. In the example, the subproblem is composed of the selected employees (2, 3, 4 and 5) and their assigned jobs (4, 5, 6, 7, 8, 9 and 10). All tasks can be performed by all employees, except for task 8 for which only employees 2, 3 and 4 are qualified. This subproblem is then solved to optimality by a general purpose solver. In the resulting neighbouring solution (Figure 2b), employee 4 is no longer required to perform any tasks. The objective value of the new solution is thus one lower than the current solution.  $k$  should be limited to ensure computational feasibility. Based on initial experimentation  $k$  was set to 40 employees. However, when  $k > |W|$ ,  $k$  was set to the total number of employees.

Employee 1		Task 1		Task 2		Task 3	
Employee 2		Task 4			Task 5		
Employee 3		Task 6		Task 7			
Employee 4					Task 8		
Employee 5		Task 9		Task 10			
Employee 6		Task 11			Task 12		

(a) Current solution

Employee 1		Task 1		Task 2		Task 3	
Employee 2		Task 4			Task 5		
Employee 3		Task 6		Task 10		Task 8	
Employee 4							
Employee 5		Task 9		Task 7			
Employee 6		Task 11			Task 12		

(b) New solution

Fig. 2: Illustration of a move with  $k = 4$ . Employees with grayed out rosters are not considered in the move.

The aforementioned neighbourhood is explored with a very large-scale neighbourhood search algorithm. In order to further guarantee computational feasibility of the solution approach, only one neighbouring solution is sampled at each iteration. In terms of iterations per minute, a trade-off exists between the number of sampled solutions in each iteration and the number of employees  $k$  selected for composing the subproblem. Smaller values for  $k$  result in faster solution times for the subproblem, making it possible to sample more neighbouring solutions in each iteration. Initial experimentation showed that better results were achieved by limiting the number of sampled solutions and increasing the size of the subproblems.

The pseudo code of the local search algorithm is shown in Algorithm 2.

**Algorithm 2** Local search algorithm

---

```

 $F(C)$  := Evaluation function
 $H$  := Move to reach a neighbouring solution
 $C_0$  := Initial solution
 $C \leftarrow C_0$ 
while Termination criterion not met do
   $C' \leftarrow H(C)$ 
  if  $F(C') \leq F(C)$  then
     $C \leftarrow C'$ 
  end if
end while

```

---

## 4 Experiments

### 4.1 Experimental setup

We evaluate the presented solution approach using instances from a benchmark dataset <sup>2</sup>. According to Krishnamoorthy et al (2011) these instances are based on their experience with real world problems. Information with regard to the number of employees and the number of tasks is shown in Figure 3. The dimensions of the instances range from small (23 employees and 40 tasks) to very large (245 employees and 2105 tasks).

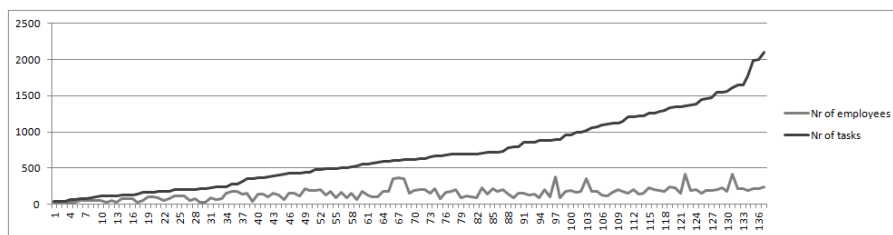


Fig. 3: Number of employees and jobs in the instances. The horizontal axis represents the different problem instances.

The experiments are carried out on an Intel Core 2 Duo at 3.16GHz with 4GB RAM operating on Windows 7. All algorithms are coded in Java and CPLEX 12.2 is used as general purpose solver. Experiments with the local search algorithm are each carried out five times. Results regarding the constructive heuristic are reported by one value since initial experimentation showed that, for the available benchmark instances, the same solution was obtained each time the algorithm was executed. The execution time for the local search procedure is limited to 1800 seconds per run.

<sup>2</sup> <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/ptaskinfo.html>

## 4.2 Experimental results

Tables 1, 2 and 3 show the results of the constructive heuristic ( $CH$ ), the average solution quality of the matheuristic local search ( $LS_{avg}$ ) and the best solution of five runs ( $LS_{best}$ ) compared with 1) a lower bound obtained by CPLEX ( $LB$ ) and 2) results from a Lagrangean-based heuristic as presented by Krishnamoorthy et al (2011) ( $LH$ ). Furthermore, the time required by the constructive heuristic and the local search are given by  $t_{CH}$  and  $t_{LS}$ , respectively.

Figure 4 shows the relative quality gap between solutions obtained with the constructive heuristic and the lower bound (gap to LB), and the constructive heuristic and the Lagrangean heuristic (gap to LH). Positive values indicate a relatively worse solution by the constructive heuristic. The average gap between the constructive heuristic and the lower bound is 4.22% whereas the average gap between the constructive heuristic and the Lagrangean heuristic is 0.08%. Based on Figure 4, two observations can be made. First, the performance of the constructive heuristic remains relatively stable for all problem sizes. This shows that the constructive heuristic is able to generate high quality solutions, even for very large instances. Second, from a certain problem size onward, the constructive heuristic outperforms the Lagrangean heuristic. This mostly indicates that the Lagrangean heuristic has difficulties with increasing problem size. For 7 out of 137 instances, the constructive heuristic finds the optimal solution, while for 100 out of 137 instances the solution is only 5% worse than the lower bound. Note that the constructive heuristic is capable of finding feasible solutions for all benchmark instances.

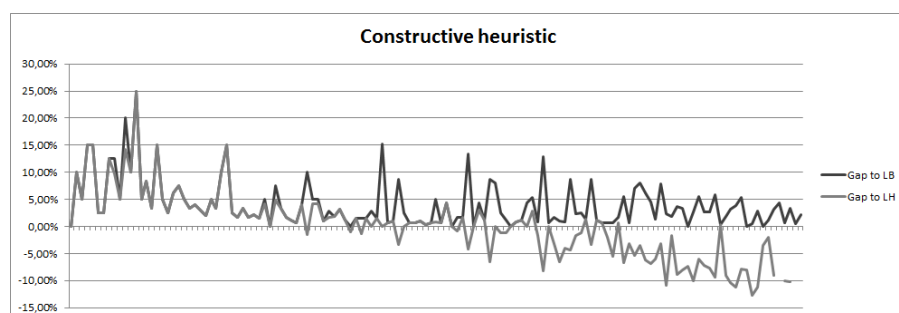


Fig. 4: Relative quality gap between the constructive heuristic and the lower bound (gap to LB) and the constructive heuristic and Lagrangean heuristic (gap to LH). The horizontal axis represents the different problem instances, from small to large.

Figure 5 shows the relative difference between the quality of solutions obtained by the local search and the lower bounds (gap to LB) and the local search and Lagrangean heuristic (gap to LH). The results clearly show that

the local search performs very well, with a maximum gap to the lower bound of 7.56% and an average gap of 0.67%. Furthermore, the presented matheuristic is capable of finding the optimal solution for 72 out of 137 instances, while for 135 out of 137 instances the local search finds solutions within 5% of the lower bound. Compared to the Lagrangean heuristic the hybrid local search also performs very well with an average improvement of 3.34%. Overall, 68 new best solutions are found by the presented matheuristic approach.



Fig. 5: Relative quality gap between the local search and the lower bound (gap to LB) and the local search and Lagrangean heuristic (gap to LH). The horizontal axis represents the different problem instances, from small to large.

When comparing the computation times in Tables 1, 2 and 3, it can be observed that, for all instances, the constructive heuristic requires less than one second of computation time to construct a solution. The computation time of the local search algorithm is plotted in Figure 6. This plot shows that if the matheuristic finds the optimal solution it finds it rather quickly. In these cases the average computation time is 74.58 seconds. However, once the dimensions of problems increase, the matheuristic does not find the optimum anymore within the time limit.

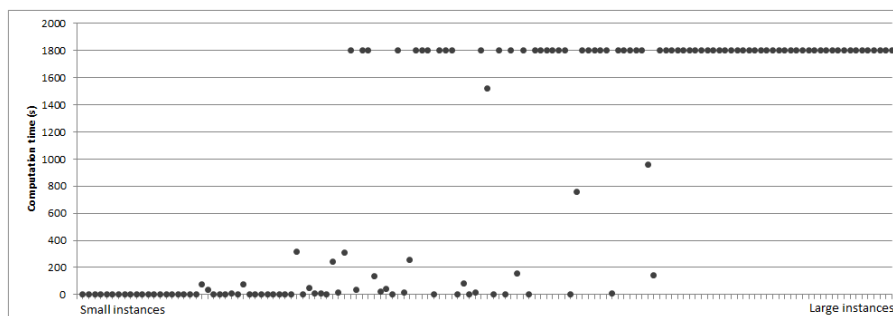


Fig. 6: Computation times of the local search algorithm (time limit set to 1800s). The horizontal axis represents the different problem instances.

The relative improvement obtained by the local search procedure over the constructive heuristic is shown in Figure 7. It can be seen that for the smaller instances, larger relative improvements are found than for the larger instances. These differences can possibly be explained by the value chosen for  $k$ . For the smaller instances,  $k$  is relatively large compared to the instance size, implying that each subproblem will consider a large part of the original problem. However, for the larger instances, the same value for  $k$  will consider a much smaller part of the problem as subproblem, and thus making it more difficult to optimise the solution as a whole. If the number of employees in the subproblem  $k$  would be chosen higher, larger improvements could be found in the local search phase. However, it is possible that other phenomena play an important role in the observations from Figure 7. Further investigation is required to determine other influencing factors.

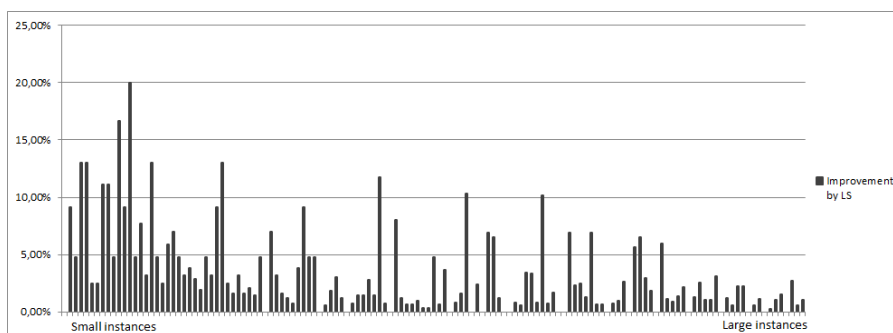


Fig. 7: Relative improvement obtained by the local search algorithm compared to the constructive heuristic. The horizontal axis represents the different problem instances.

## 5 Conclusions and future work

The paper is focussed on a hybrid heuristic approach to the SMPTSP. The constructive component of the heuristic is capable of generating feasible solutions in a very short computation time. Furthermore, the constructed solutions are of high quality, with an average gap to the lower bound of 4.22% on a set of benchmark instances from the literature.

A hybrid local search algorithm, based on the principles of matheuristics, is employed for further improving the initial solution. In this algorithm, neighbouring solutions are reached by randomly selecting a number of employees and solving the thus delineated subproblem to optimality. Due to computational feasibility issues, the number of sampled solutions in each neighbourhood at each iteration is limited, as well as the number of selected employees for the subproblem.



Experimental results showed that the hybrid heuristic performs very well, reaching solutions on average less than 1% worse than the lower bound. Analysis of the computation times showed that, if the algorithm is capable of reaching the optimum, it does so rather quickly with an average computation time of 74.58 seconds. Furthermore, the presented matheuristic found 68 new best solutions for the available benchmark instances.

In order to reach better solutions for larger instances, a high level strategy for the local search, i.e. a metaheuristic, can be used. Additional improvements to the existing algorithm can further increase the algorithmic performance. For example, another neighbourhood can be explored by probabilistically selecting employees in the subproblem instead of randomly selecting them.

Future work includes the incorporation of the presented solution approach for the SMPTSP in the larger problem of assigning tasks and shifts to employees in the same process. The speed of the constructive heuristic combined with the high quality solutions it generates present a particularly interesting opportunity in developing algorithms for the larger integrated problem.

**Acknowledgements** This research was carried out within the IWT project (IWT 110257).

## References

- Beer A, Gaertner J, Musliu N, Schafhauser W, Slany W (2008) Scheduling breaks in shift plans for call centers. In: Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montréal
- Burke E, De Causmaecker P, Vanden Berghe G, Van Landeghem H (2004) The state of the art of nurse rostering. *Journal of Scheduling* 7(6):441–499
- Burke EK, De Causmaecker P, Petrovic S, Vanden Berghe G (2006) Metaheuristics for handling time interval coverage constraints in nurse scheduling. *Applied Artificial Intelligence* 20(9):743–766
- Carter MW, Tovey CA (1992) When is the classroom assignment problem hard? *Operations Research* 40(S1):28–39
- Della Croce F, Salassa F (2010) A variable neighborhood search based matheuristic for nurse rostering problems. In: McCollum B, Burke E, White G (eds) Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010), Queen's University Belfast, pp 167–3175
- Della Croce F, Grosso A, Salassa F (2011) A matheuristic approach for the total completion time two-machines permutation flow shop problem. In: Merz P, Hao JK (eds) Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol 6622, Springer Berlin / Heidelberg, pp 38–47
- Detienne B, Peridy L, Pinson E, Rivreau D (2009) Cut generation for an employee timetabling problem. *European Journal of Operational Research* 193(3):1178–1184
- Doerner K, Schmid V (2010) Survey: matheuristics for rich vehicle routing problems. In: Blesa M, Blum C, Raidl G, Roli A, Sampels M (eds) Hybrid Metaheuristics, Lecture Notes in Computer Science, vol 6373, Springer Berlin / Heidelberg, pp 206–221
- Dowling D, Krishnamoorthy M, Mackenzie H, Sier D (1997) Staff rostering at a large international airport. *Annals of Operations Research* 72:125–147
- Ernst A, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153(1):3–27
- Gupta U, Lee D, Leung JT (1979) An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers* C-28(11):807–810

- Guyon O, Lemaire P, Pinson E, Rivreau D (2010) Cut generation for an integrated employee timetabling and production scheduling problem. *European Journal of Operational Research* 201(2):557–567
- Hanafi S, Lazic J, Mladenovic N, Wilbaut C, Crvits I (2010) New hybrid metaheuristics for solving the multidimensional knapsack problem. In: Blesa M, Blum C, Raidl G, Roli A, Sampels M (eds) *Hybrid Metaheuristics*, Lecture Notes in Computer Science, vol 6373, Springer Berlin / Heidelberg, pp 118–132
- Kolen A, Lenstra J, Papadimitriou C, Spieksma F (2007) Interval scheduling : a survey. *Naval Research Logistics* 54(5):530–543
- Krishnamoorthy M, Ernst A (2001) The personnel task scheduling problem. In: Yang X, Teo K, Caccetta L (eds) *Optimisation methods and application*, Kluwer, pp 434–368
- Krishnamoorthy M, Ernst A, Baatar D (2011) Algorithms for large scale shift minimisation personnel task scheduling problems. *European Journal of Operational Research*
- Maniezzo V, Stutzle T, Voss S (eds) (2009) *Metaheuristics: Hybridizing Metaheuristics and Mathematical Programming*, Annals of Information Systems, vol 10. Springer
- Meisels A, Schaerf A (2003) Modelling and solving employee timetabling problems. *Annals of Mathematics and Artificial Intelligence* 39:41–59
- Zeitlhofer T, Wess B (2003) List-coloring of interval graphs with application to register assignment for heterogeneous register-set architectures. *Signal Processing* 83(7):1411 – 1425

Instance	LB	LH	CH	$LS_{avg}$	$LS_{best}$	$t_{CH}$ (s)	$t_{LS}$ (s)
data_1_23_40_66	20.00	20.00	20.00	20.00	20.00	0.00	0.03
data_2_24_40_33	20.00	20.00	22.00	20.00	20.00	0.00	0.02
data_3_25_40_66	20.00	20.00	21.00	20.00	20.00	0.00	0.03
data_4_23_59_33	20.00	20.00	23.00	20.00	20.00	0.00	0.08
data_5_25_60_33	20.00	20.00	23.00	20.00	20.00	0.00	0.03
data_6_48_80_66	40.00	40.00	41.00	40.00	40.00	0.00	0.10
data_7_51_80_66	40.00	40.00	41.00	40.00	40.00	0.00	0.13
data_8_48_85_33	40.00	40.00	45.00	40.00	40.00	0.00	0.38
data_9_49_104_33	40.00	41.00	45.00	40.00	40.00	0.00	0.84
data_10_51_111_66	40.00	40.00	42.00	40.00	40.00	0.00	2.19
data_11_24_119_33	20.00	21.00	24.00	20.00	20.00	0.00	0.30
data_12_49_119_33	40.00	40.00	44.00	40.00	40.00	0.00	0.95
data_13_25_120_33	20.00	20.00	25.00	20.00	20.00	0.00	0.24
data_14_75_124_33	60.00	60.00	63.00	60.00	60.00	0.00	0.38
data_15_72_126_33	60.00	60.00	65.00	60.00	60.00	0.00	0.69
data_16_75_131_66	60.00	60.00	62.00	60.00	60.00	0.00	2.19
data_17_23_139_66	20.00	20.00	23.00	20.00	20.00	0.00	1.08
data_18_48_160_66	40.00	40.00	42.00	40.00	40.00	0.00	1.19
data_19_97_160_33	80.00	80.00	82.00	80.00	80.00	0.00	0.44
data_20_99_163_33	80.00	80.00	85.00	80.00	80.00	0.00	0.92
data_21_93_175_33	80.00	80.00	86.00	80.00	80.00	0.00	78.58
data_22_47_180_66	40.00	40.00	42.00	40.00	40.00	0.00	33.88
data_23_74_180_66	60.00	60.00	62.00	60.00	60.00	0.00	1.48
data_24_110_200_33	100.00	100.00	104.00	100.00	100.00	0.00	0.75
data_25_120_200_33	100.00	100.00	103.00	100.00	100.00	0.00	1.05
data_26_116_203_66	100.00	100.00	102.00	100.00	100.00	0.00	11.81
data_27_49_204_66	40.00	40.00	42.00	40.00	40.00	0.00	3.73
data_28_75_208_66	60.00	60.00	62.00	60.00	60.00	0.00	73.82
data_29_22_219_66	20.00	20.00	22.00	20.00	20.00	0.00	2.73
data_30_25_219_66	20.00	20.00	23.00	20.00	20.00	0.00	3.31
data_31_90_230_66	80.00	80.00	82.00	80.00	80.00	0.00	0.88
data_32_70_236_66	60.00	60.00	61.00	60.00	60.00	0.00	0.81
data_33_76_240_66	60.00	60.00	62.00	60.00	60.00	0.00	2.08
data_34_152_240_33	120.00	120.00	122.00	120.00	120.00	0.00	1.06
data_35_171_280_33	140.00	140.00	143.00	140.00	140.00	0.02	0.30
data_36_175_280_33	140.00	140.00	142.00	140.00	140.00	0.00	2.57
data_37_145_321_33	120.00	121.00	126.00	120.67	120.00	0.00	316.42
data_38_147_347_66	120.00	120.00	120.00	120.00	120.00	0.02	0.37
data_39_45_351_66	40.00	41.00	43.00	40.00	40.00	0.00	49.97
data_40_138_360_33	120.00	120.00	124.00	120.00	120.00	0.00	9.48
data_41_144_360_66	120.00	120.00	122.00	120.00	120.00	0.00	9.13
data_42_101_380_66	80.00	80.00	81.00	80.00	80.00	0.00	0.70
data_43_156_387_66	140.00	140.00	141.00	140.00	140.00	0.00	240.18
data_44_121_400_33	100.00	100.00	104.00	100.00	100.00	0.00	16.32
data_45_67_420_33	60.00	67.00	66.00	60.00	60.00	0.02	308.35
data_46_147_423_33	120.00	121.00	126.00	120.67	120.00	0.02	1800.00
data_47_150_430_33	120.00	121.00	126.00	120.00	120.00	0.02	34.63
data_48_120_434_66	100.00	100.00	101.00	101.00	101.00	0.02	1800.00
data_49_211_446_66	180.00	182.00	185.00	184.33	184.00	0.02	1800.00
data_50_187_447_66	160.00	160.00	163.00	160.00	160.00	0.02	136.13

Table 1: Detailed computational results for SMPTSP benchmark instances

Instance	LB	LH	CH	$LS_{avg}$	$LS_{best}$	$t_{CH}$ (s)	$t_{LS}$ (s)
data_51_196_480_33	160.00	160.00	165.00	160.00	160.00	0.02	23.31
data_52_205_480_66	160.00	160.00	162.00	160.00	160.00	0.02	45.65
data_53_127_487_66	100.00	101.00	100.00	100.00	100.00	0.02	1.97
data_54_175_492_66	140.00	140.00	142.00	141.00	141.00	0.02	1800.00
data_55_85_493_66	70.00	72.00	71.00	70.33	70.00	0.00	17.10
data_56_163_500_66	140.00	140.00	142.00	140.33	140.00	0.02	255.40
data_57_88_508_66	70.00	72.00	72.00	70.67	70.00	0.02	1800.00
data_58_158_517_66	140.00	140.00	142.00	140.67	140.00	0.02	1800.68
data_59_70_525_33	59.00	68.00	68.00	60.00	60.00	0.02	1800.00
data_60_181_549_66	139.00	139.00	140.00	139.33	139.00	0.02	2.87
data_61_121_557_66	100.00	100.00	101.00	101.00	101.00	0.02	1800.00
data_62_101_571_33	80.00	90.00	87.00	80.67	80.00	0.02	1800.00
data_63_97_577_66	80.00	82.00	82.00	81.00	81.00	0.02	1800.00
data_64_176_595_66	160.00	160.00	161.00	160.00	160.00	0.03	1.34
data_65_179_596_66	159.00	159.00	160.00	159.00	159.00	0.03	85.10
data_66_348_600_33	300.00	300.00	303.00	300.00	300.00	0.03	5.62
data_67_371_600_66	300.00	300.00	301.00	300.00	300.00	0.05	16.72
data_68_359_613_66	300.00	300.00	302.00	301.00	301.00	0.06	1800.00
data_69_148_614_33	120.00	125.00	126.00	120.33	120.00	0.02	1518.01
data_70_192_623_66	160.00	160.00	161.00	160.00	160.00	0.03	3.74
data_71_197_624_33	158.00	158.00	165.00	159.00	159.00	0.02	1800.00
data_72_205_624_66	160.00	160.00	160.00	160.00	160.00	0.03	0.70
data_73_155_661_66	120.00	123.00	122.00	121.67	121.00	0.03	1800.00
data_74_209_664_33	180.00	180.00	183.00	180.00	180.00	0.02	159.15
data_75_72_665_33	60.00	71.00	68.00	61.00	61.00	0.02	1800.00
data_76_162_683_66	140.00	140.00	140.00	140.00	140.00	0.03	1.83
data_77_180_688_33	160.00	162.00	167.00	163.00	163.00	0.02	1800.00
data_78_199_688_66	160.00	160.00	162.00	162.00	162.00	0.03	1800.00
data_79_94_689_33	80.00	93.00	87.00	81.00	81.00	0.00	1800.00
data_80_112_691_33	99.00	107.00	107.00	100.00	100.00	0.02	1800.00
data_81_97_692_66	80.00	83.00	82.00	81.00	81.00	0.02	1800.00
data_82_89_697_66	80.00	82.00	81.00	81.00	81.00	0.03	1800.00
data_83_222_700_66	180.00	180.00	180.00	180.00	180.00	0.05	0.87
data_84_136_718_66	120.00	120.00	121.00	120.67	120.00	0.05	761.00
data_85_217_720_66	180.00	180.00	182.00	181.00	181.00	0.05	1800.00
data_86_178_721_33	140.00	146.00	146.00	141.33	141.00	0.03	1800.00
data_87_203_735_33	170.00	174.00	179.00	173.00	173.00	0.03	1800.00
data_88_137_777_66	120.00	123.00	121.00	120.67	120.00	0.03	1800.00
data_89_88_788_33	70.00	86.00	79.00	71.33	71.00	0.02	1800.00
data_90_157_791_66	139.00	140.00	140.00	139.67	139.00	0.05	8.35
data_91_147_851_66	118.00	124.00	120.00	119.33	118.00	0.05	1800.00
data_92_126_856_66	98.00	106.00	99.00	99.00	99.00	0.03	1800.00
data_93_141_856_66	119.00	125.00	120.00	120.00	120.00	0.05	1800.00
data_94_93_881_33	80.00	91.00	87.00	81.00	81.00	0.02	1800.00
data_95_204_882_33	170.00	177.00	174.00	171.33	170.00	0.03	1800.00
data_96_98_886_66	80.00	83.00	82.00	80.33	80.00	0.05	962.12
data_97_383_895_33	300.00	300.00	304.00	300.00	300.00	0.06	146.39
data_98_91_896_33	80.00	90.00	87.00	81.33	81.00	0.03	1800.00
data_99_176_956_66	160.00	160.00	162.00	161.33	161.00	0.08	1800.00
data_100_194_956_66	160.00	160.00	161.00	160.00	160.00	0.08	1800.00

Table 2: Detailed computational results for SMPTSP benchmark instances

Instance	LB	LH	CH	$LS_{avg}$	$LS_{best}$	$t_{CH}$ (s)	$t_{LS}$ (s)
data_101_166_997_66	140.00	144.00	141.00	141.00	141.00	0.03	1800.00
data_102_179_997_66	138.00	147.00	139.00	138.33	138.00	0.03	1800.00
data_103_348_1024_33	300.00	303.00	305.00	302.67	302.00	0.05	1800.00
data_104_181_1057_33	146.00	165.00	154.00	150.67	150.00	0.02	1800.00
data_105_173_1075_66	150.00	156.00	151.00	151.00	151.00	0.05	1800.00
data_106_121_1096_33	100.00	113.00	107.00	101.67	101.00	0.02	1800.00
data_107_114_1112_33	100.00	112.00	108.00	101.33	101.00	0.02	1800.00
data_108_162_1115_33	128.00	145.00	136.00	132.00	132.00	0.02	1800.00
data_109_205_1115_33	157.00	176.00	164.00	161.33	161.00	0.03	1800.00
data_110_183_1143_66	155.00	167.00	157.00	157.00	157.00	0.05	1800.00
data_111_155_1211_33	139.00	155.00	150.00	141.67	141.00	0.03	1800.00
data_112_200_1213_33	169.00	194.00	173.00	171.00	171.00	0.03	1800.00
data_113_141_1221_66	110.00	114.00	112.00	111.33	111.00	0.05	1800.00
data_114_157_1227_33	138.00	157.00	143.00	141.33	141.00	0.03	1800.00
data_115_228_1257_33	177.00	199.00	183.00	180.33	179.00	0.03	1800.00
data_116_205_1262_66	176.00	190.00	176.00	176.00	176.00	0.08	1800.00
data_117_192_1285_33	149.00	170.00	153.00	152.00	151.00	0.03	1800.00
data_118_180_1302_33	147.00	165.00	155.00	151.67	151.00	0.03	1800.00
data_119_236_1335_33	188.00	208.00	193.00	191.67	191.00	0.05	1800.00
data_120_228_1341_33	187.00	208.00	192.00	190.67	190.00	0.05	1800.00
data_121_147_1345_33	120.00	140.00	127.00	123.33	123.00	0.03	1800.00
data_122_422_1358_66	324.48	348.00	349.00	349.00	349.00	0.33	1800.00
data_123_187_1376_33	159.00	178.00	162.00	161.00	160.00	0.08	1800.00
data_124_198_1383_33	158.00	182.00	163.00	162.33	162.00	0.08	1800.00
data_125_157_1448_33	130.00	152.00	135.00	132.33	132.00	0.08	1800.00
data_126_193_1462_33	167.00	191.00	176.00	172.33	172.00	0.09	1800.00
data_127_192_1472_66	167.83	185.00	170.00	170.00	170.00	0.19	1800.00
data_128_207_1542_66	175.29	205.00	179.00	178.67	178.00	0.22	1800.00
data_129_233_1546_33	178.00	206.00	183.00	182.00	181.00	0.11	1800.00
data_130_176_1562_66	138.35	145.00	140.00	140.00	140.00	0.17	1800.00
data_131_415_1610_33	344.07	359.00	352.00	351.00	351.00	0.34	1800.00
data_132_216_1645_33	186.00	211.00	192.00	190.67	190.00	0.13	1800.00
data_133_211_1647_33	185.00	193.00	190.33	190.33	190.00	0.14	1800.00
data_134_184_1776_66	157.56	179.00	161.00	161.00	161.00	0.27	1800.00
data_135_213_1988_33	179.00	206.00	185.00	181.67	180.00	0.19	1800.00
data_136_216_2000_66	180.00	180.00	180.00	179.67	179.00	0.36	1800.00
data_137_245_2105_33	190.00	223.00	194.00	193.33	192.00	0.22	1800.00

Table 3: Detailed computational results for SMPTSP benchmark instances

---

## A stepping horizon view on nurse rostering

Fabio Salassa · Greet Vanden Berghe

Received: date / Accepted: date

**Abstract** The present paper introduces a *stepping horizon* approach to optimisation problems whereby data can be considered static within a limited time horizon. This implies that problem instances need to be solved at certain moments in time, while imposing constraints on the subsequent period's instance. Nurse rostering can be identified as an optimisation problem for which a stepping horizon approach is recommended, whereas a static approach is suitable for academic algorithm development objectives.

In order to support this claim, the paper focuses on the *sprint* instances from the 2010 Nurse Rostering Competition. These instances represent a sufficiently realistic set of constraints while still being solvable to optimality with a general purpose solver. Two different sets of experiments are presented. First, it is shown that a static horizon approach runs the risk of generating unbalanced rosters regarding some so called *counter* constraints. A second set of experiments points at the benefits of a stepping horizon approach with respect to constraints of the so called *series* type. These two are general constraint types used as clarifying examples supporting the need for a *stepping horizon* approach. In both experimental setups, lower bounds are computed for rosters spanning more than one time horizon. The stepping horizon approach yields rosters that violate fewer constraints than those obtained in a static setting.

**Keywords** Nurse Rostering · Series Constraints · Counters Constraints · Stepping Horizon

---

F. Salassa

DAI - Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

Tel.: +39 0110907195

E-mail: fabio.salassa@polito.it

G. Vanden Berghe

CODeS - KAHO Sint-Lieven, Gebroeders De Smetstraat 1, 9000 Gent, Belgium

KU Leuven-Kulak, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium

Tel.: +32 92658610

E-mail: greet.vandenbergh@kahosl.be

## 1 Introduction

Nurse Rostering Problems (NRPs) are encountered in almost every hospital around the world. Despite possible differences between countries due to contractual and operational regulations, the core problem is always the same: assign a working shift or a day off to each nurse of a ward on each day of a defined planning horizon (often one month) taking into account a set of constraints. The assignment problem's objective function, assessing the quality of generated rosters, is usually based on constraint violations. According to [3], nurse rostering constraints can be divided into counters and series. The counters denote all the constraints that can be evaluated by counting the appearance of certain assignments in a roster. The series constraints correspond to restrictions on successive assignments, e.g. successive working weekends, successive morning shifts, etc. The literature presents a large number of different approaches devoted to NRPs covering many aspects [8]. Some work has focused on generic approaches providing a sufficient quality level over a class of instances [3,6]. Very fast and accurate heuristics [1] and recently also hyperheuristics [2] have been developed. In addition, exact methods are available [15], some of which exploit intrinsic peculiarities or specific knowledge about the problem [10], while others combine a metaheuristic framework and a Mixed Integer Linear Programming (MIP) solver [9]. The winning approach of the Nurse Rostering Competition (NRC) 2010 [22] is an excellent example of a hybrid mathematical approach. [14] obtained very good results for the competition instances with an adaptive neighborhood search thereby borrowing some ideas from SAT solvers.

We noticed, in the nurse rostering literature, that almost all the effort was spent on solving problems with a single compounded time horizon rather than on improving the perceived quality of rosters over a long period. It is a natural approach in academia to consider a restricted time horizon within which the information is complete. However, real hospital applications are strongly influenced by the inertia of previous periods. The working history has been modelled in [24], where a balance is made between the quality of the nurses' previous rosters and their preferences. [7,12] model constraints induced by the previous planning period. A few roster schemes stretching out over more than one planning horizon are visualized in both papers. In addition, some data considering future timeslots (e.g. requests for days off) may have an impact on the attainable roster quality within the present planning horizon.

It appears that the rostering horizon is a crucial element to be considered when designing NRP approaches. Suppose that an optimal approach can generate a solution in limited time. It is not unlikely that this solution suffers from an imbalance in workforce assignment. Workload balancing has only rarely been identified as an objective of nurse rostering problems [19]. In case the planning period is isolated, this workload imbalance will probably be repeated when addressing future planning periods. Such results obviously prevent automated nurse rostering approaches from being acceptable in practice. A similar example is represented by the impact of shifts assigned at the end

of the previous planning period w.r.t. the first days of the current planning period. Constraints on consecutive assignments are strongly affected by the presence of assignments in the period preceding the current one. Regardless of the quality of the applied algorithm, approaches based on these restricted models do not correspond to a hospital's requirements. Nevertheless, they are common in academic environments [5, 11, 23]. This consideration may, to some extent, explain the gap between academic and applied approaches to nurse rostering [13].

Some practical aspects should be included in an NRP model, whether clearly stated or not, in order to generate a repeatedly applicable automatic timetabling procedure. Considering simplified models for nurse rostering, and in particular models with an isolated planning horizon, we demonstrate some risks and show how they can be overcome without modifying the algorithm.

## 2 Stepping horizon

The present paper introduces the keyword *stepping horizon*, identifying the class of problems with a static time horizon, yet subject to inertia from previous periods and characterized by data concerning future timeslots that will be disclosed only as time proceeds. *Stepping horizon* approaches differ from *rolling horizon* methods [20] in that they consider a fixed time horizon and fixed data. *Rolling horizon* approaches are characterized by data uncertainty. As soon as data become available (partial) rescheduling of the current solution is performed thereby adapting the time horizon, if necessary, and the solutions to the newly available information. The ideas of a *static* versus a *rolling* and *stepping horizon* are depicted in Figures 1, 2 and 3.

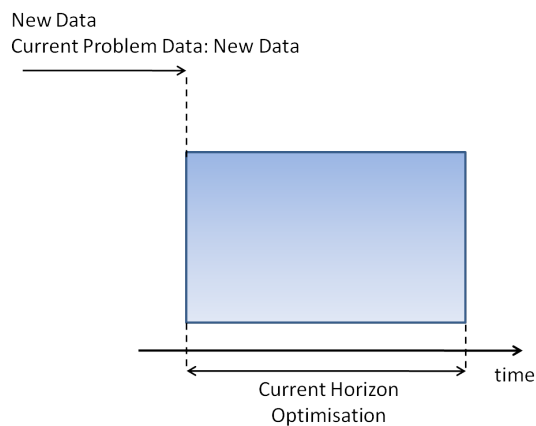
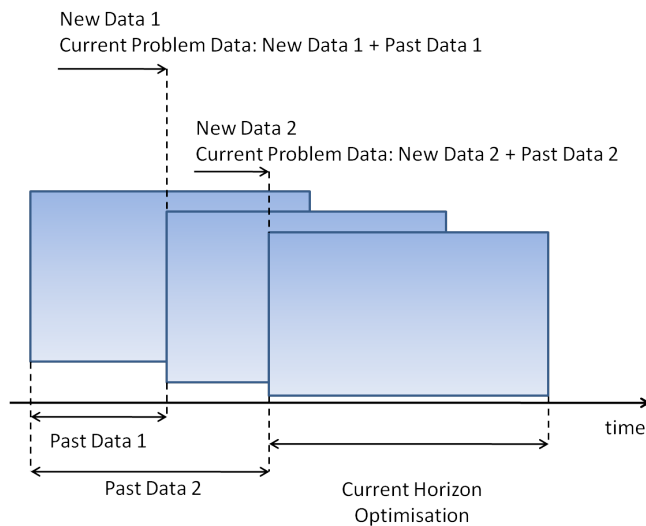


Fig. 1 Static horizon approach example



Fig. 1 presents the simple situation of a problem that is completely determined by the data corresponding to its planning horizon. The new data available are the only information used to optimise the problem. It corresponds to most of the nurse rostering instances that are publicly available for research purposes.

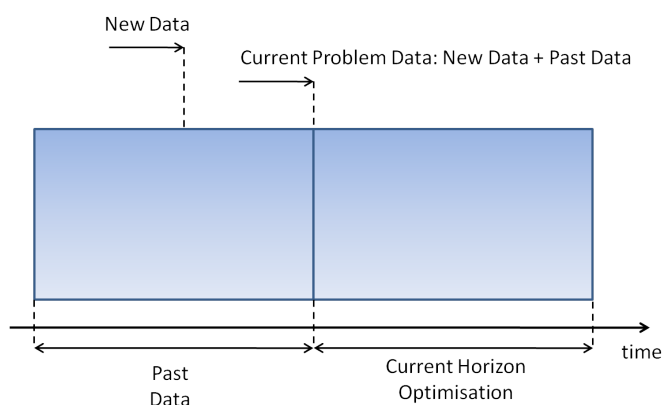


**Fig. 2** Rolling horizon approach example

Fig. 2 shows that a new static problem, denoted by a rectangle, is delineated each time new information becomes available. The time horizons are overlapping. Hence, subsets of the problem's variables will take part in a number of consecutive instances to be solved. This situation is not likely to occur in nurse rostering environments but it is very common for production scheduling [20].

Fig. 3 denotes a different approach in which the information does not change as rapidly as in rolling horizon situations. The problem presented by a rectangle can be treated as a static problem. The approach should provide some mechanisms for improving the computed solution if it appears no longer valid due to data modifications. More importantly, the solution obtained for the first planning horizon, corresponding to the leftmost rectangle, imposes restrictions on the second one [7].

The problem addressed in this paper compares to Fig. 3 in that it will not adapt the time horizon when new information becomes available. Rather, it considers whichever information from the previous planning period in order to generate a roster that violates as little constraints as possible. Obviously, in case of any data disruption, e.g. an unexpected absence, the rostering algorithm should be called for sorting out the problem, without extending the original rostering horizon. As a matter of fact, reducing the planning horizon is



**Fig. 3** Stepping horizon approach example

a more common way of dealing with disruptions [4]. Moz and Pato labelled the requirement that computed rosters sometimes need a revision due to external circumstances as the nurse rostering problem. They modelled it as a multicommodity flow problem [17] while their later work focuses on evolutionary approaches [18]. The same authors continued developing new algorithms to the nurse rostering problem, e.g. a bi-objective approach in [21]. Also Maenhout and Vanhoucke [16] present a genetic approach for rostering nurse schedules. All these recent nurse rostering approaches would fit well in a stepping horizon model.

In what follows, the importance of the stepping horizon approach is illustrated with some clear examples. The first set of experiments indicates the danger of imbalanced solutions in the long term. This is illustrated by focusing on a counter constraint and showing the potential long term effect of a small imbalance in a static roster. While balancing constraints may not be explicitly part of the problem, the results show that only limited effort is required for considering a better workload balance. In the second set of experiments, we point at the issue that series constraints can be evaluated consistently across time horizon boundaries. The results of static rosters are misleading because they appear to be better than the results of the stepping horizon approach, whereas the long term effect is again not acceptable.

### 3 Problem description

The problem considered here is a classical Nurse Rostering Problem where a working shift or a free day should be assigned to each nurse on each day of the planning horizon according to several contractual and operational requirements. Please note that free days are modelled as a special shift and hence shifts are of five kinds:

Late	14:30 – 22:30
Day	08:30 – 16:30
Early	06:30 – 14:30
Night	22:30 – 06:30
(Off)	not explicitly requested but needed to model the problem.

The instances are based on a given number of nurses, i.e. 10 for the *sprint* instances addressed in this paper. All the *sprint* instances have their planning horizon set equal to 28 days, roughly referring to a period of one month.

A set of hard constraints must be satisfied, otherwise solutions would be infeasible. The hard constraints include

- demand cover: all the shifts demanded on a day of the planning period must be assigned to the exact number of nurses
- exactly one shift (working or free) must be assigned to each nurse on each day.

The instances considered incorporate a large number of soft constraints that, when violated, contribute to the objective function value by weighted penalties. The problem's objective function should be minimised. The soft constraints of the problem belong to either the counter or the series category [3]. A limited selection of the soft constraints is presented below.

#### Counters

- maximum and minimum number of shifts that can be assigned to nurses
- maximum and minimum number of free days
- day off or shift off requests

#### Series

- maximum and minimum number of consecutive working days
- unwanted patterns (such as a Night shift followed by an Early shift).

For the complete problem definition and a detailed description of the constraints, refer to [11]<sup>1</sup>. The computational results and instance files are available at [www.kuleuven-kulak.be/nrpcompetition](http://www.kuleuven-kulak.be/nrpcompetition).

The problem considered at the competition can be modelled as an Integer Linear Problem. Indeed, with  $n$  nurses,  $m$  days in a planning horizon and  $s$  different shifts, it is sufficient to introduce a set of 0/1 variables  $x_{i,j,k}$  ( $i = 1..n$ ,  $j = 1..m$ ,  $k = 1..s$ ) indicating if nurse  $i$  is assigned to shift  $k$  on day  $j$  of the roster horizon. Correspondingly, sets of integer variables represent the different penalties that can be associated with soft constraint violations. As an example two different constraints of the problem model are discussed in detail. First, the constraint related to the maximum number of assignments is an example of how other counter type constraints are modelled. Second, the constraint determining the maximum number of allowed consecutive working

<sup>1</sup> [www.kuleuven-kulak.be/~u0041139/nrpcompetition/nrpcompetition\\_description.pdf](http://www.kuleuven-kulak.be/~u0041139/nrpcompetition/nrpcompetition_description.pdf)

days represents all other constraints of the series type. Let *MaxAssignments* be the maximum number of assignments for a nurse in the considered period and *PenaltyMA<sub>i</sub>* the integer (variable) penalty caused by the total number of exceeding assignments over the horizon for a nurse *i*.

Let *W* be the set of working shifts and *D* the set of days in the considered time horizon. Let *lim* be the limit on the number of consecutive working days defined by the problem instance and let *PenaltyMW<sub>i,j</sub>* be the binary (variable) penalty caused by a working day on day *j* exceeding the limit for a nurse *i*. This leads to the following inequalities:

$$\sum_j^D \sum_k^W x_{i,j,k} \leq \text{MaxAssignments} + \text{PenaltyMA}_i \quad \forall i = 1, \dots, n$$

$$\sum_k^W \sum_{t=0}^{\text{lim}} x_{i,j+t,k} \leq \text{lim} + \text{PenaltyMW}_{i,j} \quad \forall i = 1, \dots, n, \quad j = 1, \dots, m - \text{lim}$$

The first constraint determines a maximum number of assigned working shifts. When single time horizons are considered, constraints like the one described above can be tricky because nurses can have excess assignments over successive planning periods. The second linear inequality conditions the maximum number of consecutive working days. A penalty proportional to the excess value is issued whenever that value is exceeded. Clearly, these constraints will not be violated at the beginning of a time horizon if no data about previous periods is considered, while with the stepping horizon approach past assignments do have an effect on the current time horizon.

The objective function to be minimised eventually is the weighted sum of penalties over the entire set of nurses and constraints. All instances have been implemented with the XPRESS MOSEL modelling language. XPRESS (v. 21.01.06) has been used to solve problem instances on an Intel Core2 Duo CPU @ 2.13 GHz with 4 GB of RAM memory.

The experimental setup serves the purpose of indicating the potential drawbacks of static nurse rostering approaches, which are very common in the academic literature, compared to the stepping horizon approaches we advocate. In order to provide a clear example of the presented issues, the ideas were tested on a few instances from the Nurse Rostering Competition [11]. This choice is motivated by the fact that 1) these instances have become benchmarks for nurse rostering research and 2) some of the instances are fairly easy in that they are solvable, in less than 120 seconds, with a MIP solver. The optimality of these instances can thus be certified by the solver and validated by the evaluation algorithm provided by the competition organizers.

The stepping horizon idea is simulated by solving each instance and making sequences of the roster solutions obtained for one time horizon, here corresponding with 28 days, into a multi-period roster. In real hospital environments, the availability of the nurses and the personnel demand cannot be considered constant over the entire period. Nevertheless, the results of these simple experiments are convincing and support the stepping horizon approach.

## 4 Numerical example

The present section provides numerical results of the *stepping horizon* approach, applied to nurse rostering instances. The basic idea is to test the impact of both series and counters constraints on the solutions' quality when more than one time horizon is considered. This is simulated by considering preceding timeslots as key inputs to the current optimization problem. In other terms, results of past optimisations are added trying to adhere more to real-life applications.

### 4.1 Counter constraints, balanced workload

The focus of the first set of experiments is on workload balancing for which evaluations of counter constraints provide sufficient information. Table 1 presents results for the 10 sprint instances. The result obtained for one instance was copied into a large roster 12 times the original time horizon's size. The imbalance of working shifts between nurses is denoted by a very simple quality indicator, namely the largest difference of the total number of assigned working shifts, measured among all the nurses. Assume for example that the optimal solution assigns  $k$  working shifts to nurse  $i$  and  $l$  working shifts to nurse  $j$ , then the difference between these two nurses' assignments equals  $|k - l|$ . Although it is not always explicitly requested, we assume that a balanced number of working shifts among nurses is desirable and contributes to a balanced overall workload. Table 1 shows that the optimal solution for *sprint01* is a roster in which one or more nurses have 96 assignments over a year, while at least one other nurse has 216 assignments. The term *maximum imbalance* is introduced. It refers to the maximum difference between the work assignments of nurses over a given period. We would like to underline that we are here considering the total number of working shifts assigned to nurses, not yet taking into account the understandable preferences between the different shifts for nurses. The maximum imbalance of a solution to the *sprint01* instance is 10 shifts for a monthly roster, which produces an imbalance of 120 shifts when replicating the solution over 12 consecutive months.

In the second set of experiments, an additional hard constraint was added to the problems so that the maximum imbalance between the working shifts of any two nurses within one month is at most 3. This means that the nurse with the heaviest workload has to work at most 3 shifts more than the least active nurse, considering one rostering period. As a consequence, the instances are no longer the same as the original ones. Nevertheless, the solutions are evaluated with the same objective function. Alternatively, the maximum imbalance constraint could have been modelled as a soft constraint. Without understanding how the other constraints' weights were set, it would be hard to set an appropriate value to the new imbalance constraint's weight. The authors opted to avoid search space distortion by modelling the new

INSTANCE	MIN (working shift)	MAX (working shift)	Objective over one year	Maximum imbalance
<i>sprint01</i>	96	216	672	120
<i>sprint02</i>	96	228	696	132
<i>sprint03</i>	108	240	612	132
<i>sprint04</i>	144	204	708	60
<i>sprint05</i>	120	252	696	132
<i>sprint06</i>	144	204	648	60
<i>sprint07</i>	108	240	672	132
<i>sprint08</i>	144	240	672	96
<i>sprint09</i>	108	228	660	120
<i>sprint10</i>	132	228	624	96

**Table 1** Optimal solutions for the sprint instances (12 replicated monthly rosters) with indications of workload imbalance.

constraint as hard. In future work, the impact of a soft versus a hard balance constraint is an interesting subject to investigate.

Table 2 shows the computational results for the same instances. As was expected, the imbalance is reduced considerably within one roster. The result obtained for *sprint01* reveals that people assigned to the least number of working shifts perform 13 shifts per 28 days, whereas the people working most perform 16 shifts per 28 days. When replicated 12 times, the overall imbalance equals 36 which is much better than the imbalance of 120 resulting from the experiments in Table 1. The introduction of the balance constraint has a limited negative effect on the overall roster quality. The value of the yearly objective function increased from 672 to 744 for *sprint01*. From a computational point of view, adding a hard constraint such as the one we have introduced, makes these *sprint* instances more difficult to solve to optimality. However, the computation time never exceeds 60 seconds.

INSTANCE	MIN (working shift)	MAX (working shift)	Objective over one year	Maximum imbalance
<i>sprint01</i>	156	192	744	36
<i>sprint02</i>	156	192	780	36
<i>sprint03</i>	156	192	624	36
<i>sprint04</i>	156	192	708	36
<i>sprint05</i>	168	204	708	36
<i>sprint06</i>	168	192	648	24
<i>sprint07</i>	156	192	672	36
<i>sprint08</i>	156	192	672	36
<i>sprint09</i>	156	192	684	36
<i>sprint10</i>	168	204	636	36

**Table 2** Optimal solutions for the sprint instances subject to an additional constraint restricting the maximum difference between people's shift assignments to 3 working shifts per month (12 replicated monthly rosters).

The experiments reported in Table 3 go beyond the previous ones in that the balance constraint is much stricter. The overall results of this previous set of experiments cannot be considered completely satisfactory. It is, in fact,

clear that in the worst case a nurse can work 36 shifts more than the “luckiest” one, which roughly corresponds to a difference of almost two full time months of work. This is definitely unwanted. It seems not to be sufficient to introduce a simple working imbalance constraint because the imbalance can still be very large over a period of one year. The new constraint is formulated such that the maximum shift assignment imbalance between personal rosters is 3 within a roster horizon as well as over the entire period of 12 repetitive roster horizons. Again, the results are obtained by solving the problem once for one roster period and repeating it 12 times, which is computationally a small effort. Considering the illustrative example of *sprint01*, it can be noticed that the yearly shift assignment imbalance between people is at most 3, which is an excellent result. The drawback is that the overall roster quality over a 12 month period is 840, which is worse than the result of the experiments conducted with a monthly imbalance constraint only. Clearly, other constraint violations compensate for a better balance of the number of working shifts.

INSTANCE	MIN (working shift)	MAX (working shift)	Objective over one year	Maximum imbalance
<i>sprint01</i>	181	184	840	3
<i>sprint02</i>	181	184	875	3
<i>sprint03</i>	181	184	684	3
<i>sprint04</i>	181	184	757	3
<i>sprint05</i>	181	184	766	3
<i>sprint06</i>	181	184	683	3
<i>sprint07</i>	181	184	722	3
<i>sprint08</i>	181	184	706	3
<i>sprint09</i>	181	184	733	3
<i>sprint10</i>	181	184	716	3

**Table 3** Optimal solutions subject to an additional constraint restricting the maximum difference within a single roster horizon and over all 12 replicated roster horizons to be at most 3 working shifts

#### 4.2 Series constraints

Series constraint restrict the number of consecutive working days, free days, working weekends, etc. Similar to the experiments reported in Section 4.1, solutions are obtained with the static problem definition of the NRC instances as well as with the stepping horizon approach. The latter incorporates data from previous and future roster horizons into the problem to be solved. Little effort was spent on developing an efficient MIP model. Some of the constraints incorporated in the competition’s instances were hard to model and to verify. The number of variables appears huge for problems considering multiple planning horizons at once. It is definitely worth concentrating on improving the model in future research. Given the straightforward MIP model from Section 3, instances of limited size can be loaded by the XPRESS solver. They

correspond to period stretches of five months, which are all solvable within 10 seconds.

Preliminary experiments with the MIP solver generated a memory exception for planning horizons exceeding five months. We therefore restricted the horizon to five consecutive periods, without losing generality. Table 4 reports the results covering these five consecutive periods, from now on referred to as months. Particularly, column 2 in the table shows objective values of solving a planning period of five months as a whole. In other words, a planning period of five months has been considered instead of a single month with the objective function calculated over the complete horizon. Column 3 shows the results achieved in terms of objective function values, replicating five times the optimal value of a single month. These solutions were generated without taking into account constraints overlapping different months, such as the series constraints. Therefore the overall objective value is worse than in the previous case even for solutions that are optimal for a single month. The last column provides the achievements when previous periods are considered fixed. These solutions have been generated by optimally solving one month but considering the inertia of past periods as follows. First a problem with a planning horizon of one month is solved to optimality. Then the second month is again solved to optimality but the time horizon considered is now two months, of which the first one is represented by the solution achieved in the past step. In practice a model considering two months is generated and the variables related to the first one are fixed to the values obtained in the previous step. This procedure is repeated up to five months each time considering all the previous months. The depicted values reveal that the best would be to solve a large horizon in one go. This is almost impossible because data are available only as timeslots pass. Even if suboptimal, a more interesting procedure than solving only single time horizons, is the stepping horizon approach. A fairly good solution can be generated when also considering constraints overlapping months. That is obtained by solving single time horizons while basing the current solution on what has happened in the previous periods. In the authors' opinion this procedure should always be conducted when optimising nurse rostering instances.

INSTANCE	Obj. Fun. one go	Month by month Obj over 5 months	Stepping horizon
<i>sprint01</i>	276	332	287
<i>sprint02</i>	286	306	297
<i>sprint03</i>	251	287	262
<i>sprint04</i>	284	315	292
<i>sprint05</i>	290	310	297
<i>sprint06</i>	266	314	272
<i>sprint07</i>	280	312	287
<i>sprint08</i>	276	296	280
<i>sprint09</i>	271	307	281
<i>sprint10</i>	264	304	271

**Table 4** Optimal solutions subject to series constraints



## 5 Conclusion and future work

Academic problems tend to concentrate on static instances representing an isolated planning period. While a static approach is very common in nurse rostering research, the present paper focused on drawbacks over a long term work stretch. The idea of a stepping horizon was introduced in order to model problems in such a way that they compare better to real practice in hospitals. A stepping horizon incorporates characteristics from a static as well as from a rolling horizon.

A set of simple experiments was set up so as to indicate the strength of a stepping horizon approach. Instances have been taken from the first International Nurse Rostering Competition. The smallest *sprint* instances of that competition were solvable to optimality with a straightforward MIP approach and these optimal solutions allowed to make strong quality claims.

The experiments concentrated on two sets of roster qualities. Static horizons often proved to induce significant imbalance between individual nurses' assignments. This set of experiments concentrated on counter type constraints. The introduction of an additional balance constraint showed not to be sufficient to cope with the intrinsic imbalance of splitting a long term problem into isolated small problems. The stepping horizon approach provides an alternative in that its long term effect on balanced workload is advantageous, at the expense of potentially reducing the quality within the present planning horizon.

Besides counter constraints, a second set of experiments demonstrated that series constraints can also have a strong impact on the roster quality of subsequent rosters, when optimising static rosters only. In practical applications of nurse rostering it is inevitable that series constraints will overlap the monthly planning horizons. Hence, static horizon approaches are inadequate while a stepping horizon approach offers a manner to cope with series constraints across planning horizon borders.

Both sets of experiments produced somewhat poorer results within a single time horizon, whereas the long term effect was significantly better.

The tests were conducted on the smallest NRC instances only for computational reasons. Exactly the same experiments can be translated to the larger instances or to complex real experiments, for which dedicated algorithms are more appropriate than MIP solvers. The positive effect of the stepping horizon approach is expected to be stronger in case of a large set of complex constraints.

Future work will be dedicated to investigating the impact on other counter constraints than the total number of assigned shifts. The implications of the stepping horizon approach on the overall objective including all the counter and series constraints will be investigated too. In addition, the design of appropriate objective functions will be investigated so that results of a stepping horizon approach generate the best possible long term effect. These examinations will preferably be conducted using the mathematical solver, if the model

can be improved sufficiently. Otherwise, heuristics for nurse rostering are a reasonable alternative.

Another interesting aspect to be studied is the benchmark of the stepping horizon approach compared to other ways of dealing with long term horizons, such as shift rotation schedules.

Sets of real rostering problems with a given working history will be collected to support future research.

## References

1. J.F. Bard and H.W. Purnomo. Real-time scheduling for nurses in response to demand fluctuations and personnel shortages. In E.K. Burke and M. Trick, editors, *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, PATAT, pages 67–87, Pittsburgh, August 2004.
2. B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, and G. Vanden Berghe. One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics*, 18(3):401–434, 2012.
3. B. Bilgin, P. De Causmaecker, B. Rossie, and G. Vanden Berghe. Local search neighbourhoods to deal with a novel nurse rostering model. *Annals of Operations Research*, 194(1):33–57, 2012.
4. E.K. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence, Special issue on Simulated Evolution and Learning*, 15:199–214, 2001.
5. E.K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A scatter search approach to the nurse rostering problem. *Journal of the Operational Research Society*, 61:1667–1679, 2010.
6. E.K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A time pre-defined variable depth search for nurse rostering. *INFORMS Journal on Computing*, to appear.
7. E.K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Fitness evaluation for nurse scheduling problems. In *Proceedings of the Congress on Evolutionary Computation (CEC2001)*, pages 1139–1146, Seoul, Korea, May 27–30 2001. IEEE Press.
8. E.K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
9. F. Della Croce and F. Salassa. A variable neighborhood search based matheuristic for nurse rostering problems. Technical report, Politecnico di Torino – <http://dl.dropbox.com/u/24916303/TR-01-02-2012.pdf>, 2012.
10. C.A. Glass and R.A. Knight. The nurse rostering problem: A critical appraisal of the problem structure. *European Journal of Operational Research*, 202:379–389, 2009.
11. S. Haspeslagh, P. De Causmaecker, M. Stolevik, and A. Schaefer. The first international nurse rostering competition 2010. *Annals of Operations Research*, 194(1):59–70, 2012.
12. A. Ikegami and A. Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.
13. D.L. Kellogg and S. Walczak. Nurse scheduling: From academia to implementation or not? *Interfaces*, 37(4):355–369, 2007.
14. Z. Lu and J.K. Hao. Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, 218(3):865 – 876, 2012.
15. B. Maenhout and M. Vanhoucke. Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, 13:77–93, 2010.
16. B. Maenhout and M. Vanhoucke. An evolutionary approach for the nurse rerostering problem. *COMPUTERS & OPERATIONS RESEARCH*, 38:1400–1411, 2011.
17. M. Moz and M. Pato. An integer multicommodity flow model applied to the rerostering of nurse schedules. *Annals of Operations Research*, 119:285–301, 2003.
18. M. Moz and M. Pato. A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research*, 34:667–691, 2007.

19. D. Ouelhadj, S. Martin, P. Smet, E. Özcan, and G. Vanden Berghe. Fairness in nurse rostering. Technical report, University of Portsmouth, 2012.
20. D. Ouelhadj and S. Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12:417–431, 2009.
21. M. Pato and M. Moz. Solving a bi-objective nurse rostering problem by using a utopic pareto genetic heuristic. *Journal of Heuristics*, 14:359–374.
22. C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, and E. Housos. A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, 219(2):425 – 433, 2012.
23. M. Vanhoucke and B. Maenhout. NSPLib - a nurse scheduling problem library: A tool to evaluate (meta-)heuristic procedures. *Operational research for health policy: Making better decisions*, pages 151–165, 2007.
24. M. Warner. Nurse staffing, scheduling, and reallocation in the hospital. *Hospital & Health Services Administration*, pages 77–90, 1976.

---

# The Patrol Scheduling Problem

Hoong Chuin Lau • Aldy Gunawan

**Abstract** This paper presents the problem of scheduling security teams to patrol a mass rapid transit rail network of a large urban city. The main objective of patrol scheduling is to deploy security teams to stations at varying time periods of the network subject to rostering as well as security-related constraints. We present a mathematical programming model for this problem. We then discuss the aspect of injecting randomness by varying the start times, the break times for each team as well as the number of visits required for each station according to their reported vulnerability. Finally, we present results for the case of Singapore mass rapid transit rail network and synthetic instances.

*Keywords:* patrol scheduling problem, preferences, mass rapid transit rail network, mathematical programming.

## 1 Introduction

Personnel scheduling and rostering is concerned with the process of constructing optimized work timetables for staff in order to satisfy the demand for the organization. Ernst et al. (2004) provide a recent review of staff scheduling and rostering in specific applications areas. Some are concerned with rostering within a physical premise such as hospitals, and examples of such problems include nurse rostering (e.g. Petrovic and Berghe, 2008) and physician scheduling (e.g. Gunawan and Lau, 2010). A more challenging problem involves rostering of personnel that require them to move from one geographical location to another as they discharge their duties, such as airline crew scheduling (e.g. Maenhout and Vanhoucke, 2010) and train crew scheduling (e.g. Chu and Chan, 1998).

In this paper, we are concerned with the planning problem of assigning security teams to patrol a public transportation network (such as subways) of a large urban city. This is termed the **Patrol Scheduling Problem**. This problem is motivated by increasing need for protecting major public facilities (such as urban transport systems) in response to global threats. In order to enforce security, security personnel or teams are deployed to patrol the stations throughout the day. Unlike

---

Hoong Chuin Lau  
School of Information Systems, Singapore Management University  
80 Stamford Road, Singapore 178902  
E-mail: [hclau@smu.edu.sg](mailto:hclau@smu.edu.sg)

Aldy Gunawan  
School of Informatics & IT, Temasek Polytechnic  
21 Tampines Avenue 1  
E-mail: [agunawan@tp.edu.sg](mailto:agunawan@tp.edu.sg)

standard employee rostering which follows prescribed patterns, patrol activities should ideally exhibit randomness so as to hedge against adversarial observations. It goes without saying that in view of limited manpower resources, it is necessary to maximize the impact of patrolling duties through solving the problem optimally.

The main objective of this paper is to develop an exact model to deploy security teams to stations in varying time periods of the network while ensuring rostering and other security-related constraints. We also consider aspects of randomness to hedge against adversarial observations. To our knowledge, this study is one of very few attempts to solve the patrol scheduling problem on a mass rapid transit rail network.

The remaining part of the paper is organized as follows. We first provide a brief literature review. We then give a detailed description of our patrol scheduling problem in Section 3. We provide a deterministic mathematical programming model that solves the problem, followed by a randomized strategy which allows the planner to generate solutions based on randomized start times, break times for each team as well as the number of visits required for each station. The next section is dedicated to the computational analysis of the model on the Singapore MRT Rail System, as well as on randomly generated problem instances. Finally, we provide some concluding perspectives and directions for future research.

## 2 Literature Review

Crew rostering in public transport systems is an active area of research. An example of a rail transport scheduling problem is Chu and Chan (1998), who studied the problem of crew scheduling for the Hong Kong Light Rail Transit. The complex schedule construction is decomposed into separate solution stages by network and heuristic algorithms. They reported that the entire crew schedule can be constructed iteratively in less than an hour, which is better than the manual allocation. Although optimality cannot be claimed, the feasibility of the solution was ensured, which can still be further improved manually.

A more recent work of Elizondo et al. (2010), which considers the problem of conductors duty generation in the Santiago Metro System. With regard to operational and labor conditions, the goal is to use the lowest possible number of conductors and minimize total idle time between trips. They solved the problem using a constructive hybrid approach which takes advantage of the benefits offered by evolutionary methods. Their hybrid method produced solutions with the minimum number of duties in six of the ten problems solved.

On patrol scheduling, the major purpose is to ensure the safety of the commuters and to discourage those who might commit crimes (Rosenshine, 1970). The patrol scheduling method developed there is based on the assumption of randomness. The arrival patterns of the security patrol to a particular station could not be predicted. On the other hand, the irregularity of patrol schedules would increase the awareness of the commuters that patrol is taking place. The arc flows were determined by solving a linear programming problem while the random arrival patterns on each arc were generated by choosing exponential inter-dispatch times along the generated routes.

Stern and Teomi (1986) studied and proposed two algorithms for scheduling security guards in a large organization in Israel. The problem was formulated as a multi-objective problem and solved by a simpler heuristic intuitive algorithm. Taylor and Huxley (1989) considered the problem of assigning police officer shifts so that under cover is minimized. The optimization-based decision support system was developed and implemented in the Police Patrol Scheduling System at the San Francisco Police Department. Sharma and Ghosh (2007) proposed an optimal deployment of police patrol cars for the department of traffic police on the metropolitan city, Delhi (Central). A goal programming model was designed to determine the number of patrol cars to have on duty per shift and road segment.

The application of game theory to patrol scheduling took center stage in recent research. Tsai et al. (2009) for example modeled the strategic security allocation problem as a Stackelberg game and developed the *Intelligent Randomization In Scheduling* (IRIS) system – a tool for strategic security allocation in transportation networks. The algorithmic advances in multi-agent systems research are being used to solve the class of massive security games with complex constraints, the Federal Air Marshals (FAMs) that provide law enforcement aboard U.S. commercial flights.

Ordóñez et al. (2012) described the recent development of game-theoretic models to assist security forces in randomizing their patrols and their deployment by assuming intelligent adversary responses to security measures. They proposed fast algorithms for solving large instances of Bayesian Stackelberg games to two real-world security applications: 1) the police at the Los Angeles International Airport and 2) the Federal Air Marshal Service. Stackelberg games are a bilevel model that account for the ability of an adversary to gather information about the defense strategy before planning an attack (Basar and Olsder, 1995). The generic mathematical formulation is described as the set covering model where the set of schedules of security forces are pre-determined.

Jiang et al. (2012) presented an approach to generate fare-inspection strategies in urban transit systems using a Stackelberg game. The problem is to deploy security personnel randomly to inspect passenger tickets. The real problem from the Los Angeles Metro Rail System was formulated and solved as an LP relaxation with a maximum-revenue patrol strategy. The solutions obtained seem to effectively deter fare evasion and ensure high levels of revenue.

### 3 Problem Definition

This paper focuses on a patrol scheduling the mass rapid transit rail network. Figure 1 shows the subway systems of London, Beijing, Paris and Singapore respectively. A common feature of these networks is that each network consists of many stations linked by hub (interchange) stations.

We define the Patrol Scheduling Problem as follows. We are given a number of security teams responsible for the patrolling task. We assume the time horizon to be a single work day divided into time periods. A shift is defined as a consecutive set of time periods, and in this paper, we assume each period to be one hour, and there are two 8-hourly shifts (7am – 3pm and 3pm – 11pm respectively). Each team is rostered to a single shift duty during which it is responsible for

patrolling/visiting a subset of stations in the network. We assume each station patrol/visit takes one period and each shift is made up of exactly six visits plus two breaks.

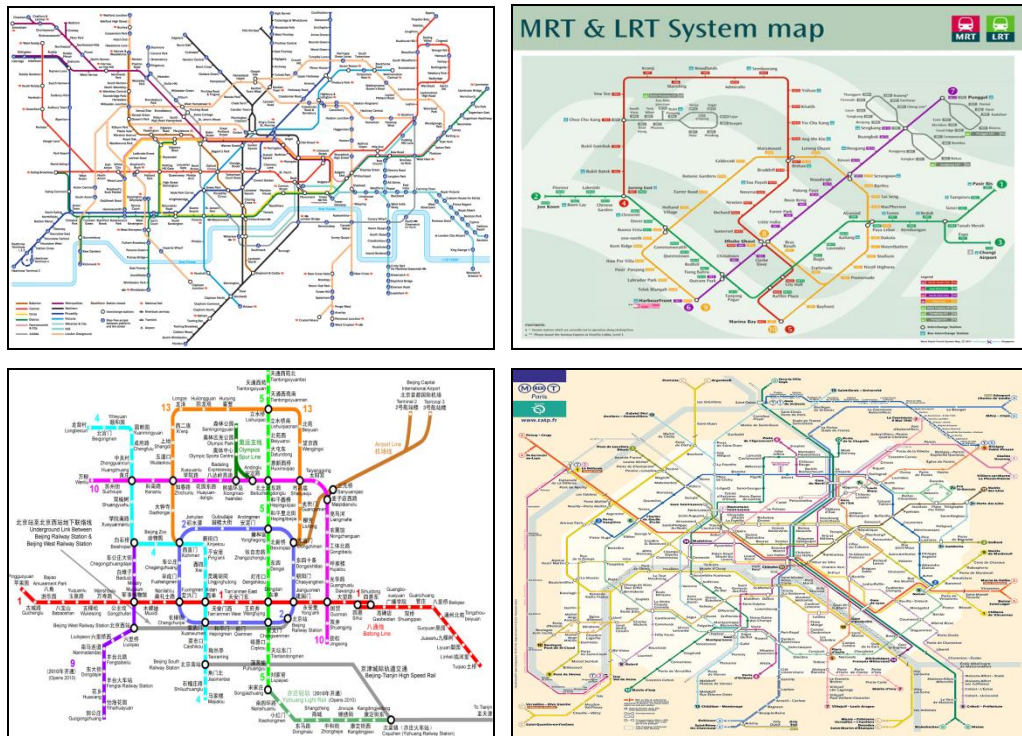


Figure 1. Examples of the mass rapid transit rail network in some cities

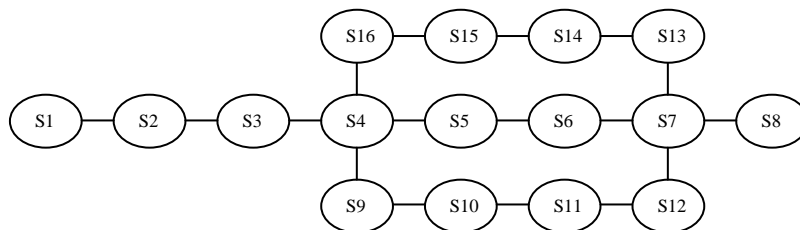


Figure 2. Example of the mass rapid transit rail network

Time Period	1	2	3	4	5	6	7	8
Team1	S1	S3	Break	S5	S6	Break	S7	S13
Team2	S10	S12	Break	S14	S16	Break	S4	S2

Figure 3. Example of Patrol Scheduling Problem

As shown in Figure 2 as illustration, the mass rapid transit rail network consists of two different lines. There are 16 stations in total where Station 4 (S4) and Station 7 (S7) are interchange stations. Assuming there are two teams in the first shift, Figure 3 represents one possible patrol scheduling for both teams. Team1 has to visit S1, S3, S5, S6, S7 and S13 consecutively while Team2 has to visit S10, S12, S14, S16, S4 and S2 consecutively.

Our goal is to minimize the total distance travelled. We use this objective in order to generate solutions that minimize unnecessary movements (where teams move in a certain path, rather than haphazardly or in loops). The distance travelled between two stations is computed as the smallest number of stations passed (since there may be more than one path from one station to another). For example, the distance between S1 and S4 is 3 stations. Furthermore, we also impose an additional penalty for distance between two stations using *different* lines. From S5 to S16, the distance travelled is 2 stations +  $\Delta$ , where  $\Delta$  is the penalty value. In our study, we set  $\Delta$  to an arbitrarily number, e.g. 10. This can be set to any large number with the purpose to minimize unnecessary movements.

The following summarizes the requirements/constraints treated in this paper:

- The number of visits for each team should meet the requirement.
- At most one team can visit a particular station at a particular time period.
- Each station has a minimum and maximum number of visits per day.
- Each team has its own start and finish times. In this paper, we treat the start and finish times as input (i.e. assume they have been determined by the planner).
- Each team may only visit a particular station at most once during its duty.
- Each team visits at most one station at a particular time period.
- Consecutiveness constraints: describes whether a pair of stations can be visited consecutively (i.e. one after another).
- Break constraints: each team is given two breaks, and breaks cannot occur consecutively.

As discussed in the Introduction, the element of randomness is important to patrol scheduling to hedge against adversarial observations. To this end, game theory has been applied recently (see Literature Review above) which utilizes reports from Intelligence sources to compute mixed strategies. Since our focus in this paper is on the patrol scheduling problem defined above, we treat the computation of these mixed strategies as a pre-processing step, which is computed whenever a new roster needs to be generated (e.g. daily).

It is conceivably that Intelligence sources will provide different data on the vulnerability of stations from time to time, which in our context can be translated to the input required by our problem (more precisely, the randomized frequency of visits of each station, as well as start times and break times of each patrol team). Our purpose in this paper is to demonstrate that by appropriately randomizing the frequency of visits, we can effectively deter crimes compared to fixed frequency.

## 4 Mathematical Programming Model

In this section, we first present a deterministic mathematical programming model to solve the patrol scheduling problem. We then consider a simple strategy for randomizing inputs for feeding into our model.



## 4.1 Deterministic Model

The problem can be presented as a mathematical programming model, using the following sets, input parameters and decision variables:

### Parameters

$I$	= Set of patrol teams, $i \in \{1, 2, \dots,  I \}$
$J$	= Set of stations, $j \in \{1, 2, \dots,  J \}$
$K$	= Set of time periods, $k \in \{1, 2, \dots,  K \}$
$Req_i$	= number of visit required for patrol team $i$ per day ( $i \in I$ )
$Max\_Visit_j$	= maximum number of visit required for station $j$ per day ( $j \in J$ )
$Min\_Visit_j$	= minimum number of visit required for station $j$ per day ( $j \in J$ )
$Start_i$	= start time for team $i$ ( $i \in I$ )
$Finish_i$	= finish time for team $i$ ( $i \in I$ ) (i.e., $Finish_i = Start_i + 7$ )
$Break_i^1$	= team $i$ 's first break (i.e., the first break is at period $(Start_i + Break_i^1)$ )
$Break_i^2$	= team $i$ 's second break ( $Break_i^2 > Break_i^1 + 1$ )
$Dist_{j_1 j_2}$	= the distance between stations $j_1$ and $j_2$ ( $j_1, j_2 \in J$ )
$Cons_{j_1 j_2}$	= 1 if a patrol team can visit $j_2$ consecutively (i.e. at the next time period) after visiting station $j_1$ , and 0 otherwise

### Decision variables

$X_{ijk}$  = 1 if patrol team  $i$  visits station  $j$  at time period  $k$  ( $i \in I, j \in J, k \in K$ ), 0 otherwise

The formulation for the Patrol Scheduling problem is then given by

$$\begin{aligned}
 \text{Minimize } Z = & \sum_{i \in I} \sum_{\substack{j_1 \in J, j_2 \in J \\ (j_1 \neq j_2)}} \sum_{\substack{k \in \{0, \dots, (Finish_i - Start_i - 1) \\ k \notin \{Break_i^1, Break_i^1 - 1, Break_i^2, Break_i^2 - 1\}}} Dist_{j_1 j_2} \times X_{ij_1(Start_i + k)} \times X_{ij_2(Start_i + k + 1)} \\
 & + \sum_{i \in I} \sum_{\substack{j_1 \in J, j_2 \in J \\ (j_1 \neq j_2)}} \sum_{k \in \{Break_i^1 - 1, Break_i^2 - 1\}} Dist_{j_1 j_2} \times X_{ij_1(Start_i + k)} \times X_{ij_2(Start_i + k + 2)} \quad (1)
 \end{aligned}$$

subject to:

$$\sum_{j \in J} \sum_{k \in K} X_{ijk} = Req_i \quad i \in I \quad (2)$$

$$\sum_{i \in I} X_{ijk} \leq 1 \quad j \in J, k \in K \quad (3)$$

$$\sum_{i \in I} \sum_{k \in K} X_{ijk} \geq Min\_Visit_j \quad j \in J \quad (4)$$

$$\sum_{i \in I} \sum_{k \in K} X_{ijk} \leq Max\_Visit_j \quad j \in J \quad (5)$$

$$\sum_{k \in K} X_{ijk} \leq 1 \quad i \in I, j \in J \quad (6)$$

$$\sum_{j \in J} X_{ijk} \leq 1 \quad i \in I, k \in K \quad (7)$$

$$\sum_{j \in J} \sum_{\substack{k \in K \\ k < Start_i}} X_{ijk} = 0 \quad i \in I \quad (8)$$

$$\sum_{j \in J} \sum_{\substack{k \in K \\ k > Finish_i}} X_{ijk} = 0 \quad i \in I \quad (9)$$

$$\sum_{j \in J} X_{ij(Start_i + Break_i^1)} = 0 \quad i \in I \quad (10)$$

$$\sum_{j \in J} X_{ij(Start_i + Break_i^2)} = 0 \quad i \in I \quad (11)$$

$$X_{ij_2(Start_i + k + 1)} \leq \sum_{\substack{j_1 \in J \\ j_1 \neq j_2}} Cons_{j_1 j_2} \times X_{ij_1(Start_i + k)} \quad \begin{array}{l} i \in I, j_2 \in J, k \in \{0, \dots, Finish_i - Start_i - 1\} \\ k \notin \{Break_i^1, Break_i^2, Break_i^1 - 1, Break_i^2 - 1\} \end{array} \quad (12)$$

$$X_{ij_2(Start_i + k + 2)} \leq \sum_{\substack{j_1 \in J \\ j_1 \neq j_2}} Cons_{j_1 j_2} \times X_{ij_1(Start_i + k)} \quad i \in I, j_2 \in J, k \in \{Break_i^1 - 1, Break_i^2 - 1\} \quad (13)$$

$$X_{ijk} \in \{0, 1\} \quad i \in I, j \in J, k \in K \quad (14)$$

Equation (1) shows that the objective function consists of two terms. The first term of the objective function refers to the distance travelled between two consecutive time periods. Due to the break constraints, we introduce the second term in the objective function. Suppose a team has the start time at Period 1 and the breaks at Periods 3 and 6. Then the first term calculates the distance travelled between Periods 1 and 2, Periods 4 and 5 and Periods 7 and 8, while the second term computes the distance travelled between one period before and after the break (i.e. Periods 2 and 4 and Periods 5 and 7).

Constraint (2) ensures that all teams have to visit a certain number of stations during their duty. Constraint (3) restricts that only one team can visit a particular station at a particular time period. Constraints (4) and (5) represent the number of visits allowed for each station per day. In our problem, the interchange stations are visited more often than those of non-interchange stations. Each team can only visit a particular station at most once per day and each team can only patrol at most one station at a particular time period. These requirements are represented by Constraints (6) and (7).

Note that the start (and therefore finish) times of each team is an input to the model. Constraints (8) and (9) ensure that all teams can only perform their patrolling task during their shift. The break constraints are defined in (10) and (11). The consecutiveness constraint is represented by constraints (12) and (13). Constraint (12) defines the consecutiveness requirement between two consecutive periods of duty. Since the breaks occur at periods  $Start_i + Break_i^1$  and  $Start_i + Break_i^2$ , we introduce constraint (13) to ensure consecutiveness between the stations visited the one period before and after a particular break.

Notice that the objective function of the model is not linear and our experiment shows that the model given above cannot be solved within reasonable time. Furthermore, standard linearization technique (see Hammer and Rudeanu, 1968) also yields unsatisfactory performance. In the following, we propose a linearization of the problem by introducing an additional set of binary variables  $Y_{ij,kj_2m} = X_{ij_1k} \times X_{ij_2m}$ . The objective function (1) is replaced by the following equation:

$$\begin{aligned} \text{Minimize } Z = & \sum_{i \in I} \sum_{j_1 \in J, j_2 \in J} \sum_{\substack{k \in \{0, \dots, (Finish_i - Start_i - 1)\} \\ (j_1 \neq j_2) k \notin \{Break_i^1, Break_i^1 - 1, Break_i^2, Break_i^2 - 1\}}} \sum_{k \in \{0, \dots, (Finish_i - Start_i - 1)\}} Dist_{j_1 j_2} \times Y_{ij_1(Start_i+k)j_2(Start_i+k+1)} \\ & + \sum_{i \in I} \sum_{j_1 \in J, j_2 \in J} \sum_{\substack{k \in \{Break_i^1 - 1, Break_i^2 - 1\} \\ (j_1 \neq j_2)}} \sum_{k \in \{Break_i^1 - 1, Break_i^2 - 1\}} Dist_{j_1 j_2} \times Y_{ij_1(Start_i+k)j_2(Start_i+k+2)} \end{aligned} \quad (15)$$

To achieve this, the following constraints need to be added:

$$\sum_{\substack{j_2 \in J \\ j_2 \neq j_1}} Y_{ij_1(Start_i+k)j_2(Start_i+k+1)} = X_{ij_1(Start_i+k)} \quad \begin{array}{l} i \in I, j_1 \in J, k \in \{0, \dots, Finish_i - Start_i - 1\} \\ k \notin \{Break_i^1, Break_i^2, Break_i^1 - 1, Break_i^2 - 1\} \end{array} \quad (16)$$

$$\sum_{\substack{j_1 \in J \\ j_1 \neq j_2}} Y_{ij_1(Start_i+k)j_2(Start_i+k+1)} = X_{ij_2(Start_i+k+1)} \quad \begin{array}{l} i \in I, j_2 \in J, k \in \{0, \dots, Finish_i - Start_i - 1\} \\ k \notin \{Break_i^1, Break_i^2, Break_i^1 - 1, Break_i^2 - 1\} \end{array} \quad (17)$$

$$\sum_{\substack{j_2 \in J \\ j_2 \neq j_1}} Y_{ij_1(Start_i+k)j_2(Start_i+k+2)} = X_{ij_1(Start_i+k)} \quad \begin{array}{l} i \in I, j_1 \in J, k \in \{0, \dots, Finish_i - Start_i - 1\} \\ k \notin \{Break_i^1 - 1, Break_i^2 - 1\} \end{array} \quad (18)$$

$$\sum_{\substack{j_1 \in J \\ j_1 \neq j_2}} Y_{ij_1(Start_i+k)j_2(Start_i+k+2)} = X_{ij_2(Start_i+k+2)} \quad \begin{array}{l} i \in I, j_2 \in J, k \in \{0, \dots, Finish_i - Start_i - 1\} \\ k \notin \{Break_i^1 - 1, Break_i^2 - 1\} \end{array} \quad (19)$$

$$Y_{ij_1kj_2m} = \{0, 1\} \quad i \in I, j_1 \& j_2 \in J, k \& m \in K \quad (20)$$

## 4.2 Randomized Strategy

The above model will work well in commercial rosters where the emphasis is on regularity. In security patrol scheduling context however, the element of randomness is important. In this section, we consider the problem of randomizing the start times, the break times for each team as well as the number of visit required for each station.

For start times and break times, we generate them randomly for each team based on a uniform distribution, which may be replaced with any other probability distributions. We would like to observe the impact of computational performance in solving the underlying deterministic mathematical model.

The frequency of visits is not purely random, but is dependent on the level of threats (vulnerability) of each station. In this paper, we assume the existence of Intelligence sources that provide information about the likelihood that crimes are going to occur at each particular station, from which we can calculate the randomized frequency distribution of visits.

More precisely, suppose  $X$  is a discrete random variable that represents the adversary's probability distribution of committing a crime, given as follows:

$$X = \begin{cases} Pr(\text{a crime occurs at Station1}) = p_1 \\ Pr(\text{a crime occurs at Station2}) = p_2 \\ \vdots \\ Pr(\text{a crime occurs at Station } |J|) = p_{|J|} \end{cases} \quad (21)$$

where  $p_1 + p_2 + \dots + p_{|J|} = 1$ . Assume that there are  $|I|$  teams where each team has to visit  $Req_i$  stations, so the total number of visits per day is  $\sum_{i \in I} Req_i$ . In this paper, we use the inverse transform method (Ross, 2009) to generate the randomized strategy (i.e. the distribution of the number of visits for all stations) as follows. We generate  $\sum_{i \in I} Req_i$  random numbers drawn from the uniform distribution  $U(0,1)$ , and for each number  $U$ , we increment the number of visits by one to Station  $j$  if  $\sum_{i=1}^{j-1} p_i < U \leq \sum_{i=1}^j p_i$ . The result is a randomized vector which represents the frequency of visits for all stations.

To simulate the occurrence of crime, we apply the same method, namely, generate a random number  $U$  from the uniform distribution  $U(0,1)$ ; a crime occurs at Station  $j$  if  $\sum_{i=1}^{j-1} p_i < U \leq \sum_{i=1}^j p_i$ . To test the effectiveness of our proposed randomized strategy, we perform a simulation of a number of replications. For each replication, we simulate the occurrence of crime at a particular station as described above, and determine whether the roster generated from the randomized visit frequency is able to counteract/deter this crime. This is benchmarked against a fixed frequency of visits described above. In the following section, we report results on the effectiveness of our random strategy against the fixed strategy.

## 5 Computational Results

In this section, we present the computation results together with our evaluation based on of the proposed mathematical programming model. All experiments that we report on this section were run on a 3.07 GHz Intel (R) Xeon (R) CPU with 128GB of RAM under the Microsoft Windows XP Operating System. The mathematical programming model was solved by CPLEX 10.0 solver engine. We first describe the experimental setup, followed by experimental results.

### 5.1 Experimental Setup

In order to demonstrate the capabilities of our proposed model, the Singapore rail network was chosen as a case study (Figure 4). In addition, two different random instances (Figures 5 and 6) were also generated with varying values of the following parameters – the number of teams, the number of stations, the number of interchange stations (Table 1).

Table 1. Characteristics of Problem Instances

Problem Set	Number of Teams	Number of stations	Number of interchange stations	Number of time periods per day	Number of stations visited per team
Random1	4	20	1	16	6
Random2	5	24	2	16	6
Case Study	24	90	10	16	6

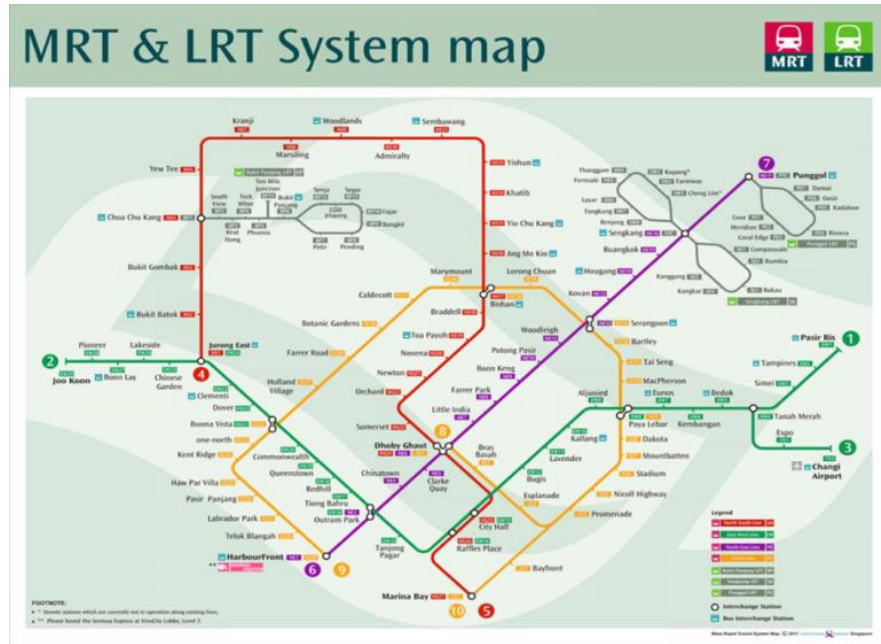


Figure 4. Singapore MRT map (source: [http://www.smrt.com.sg/trains/network\\_map.asp](http://www.smrt.com.sg/trains/network_map.asp))

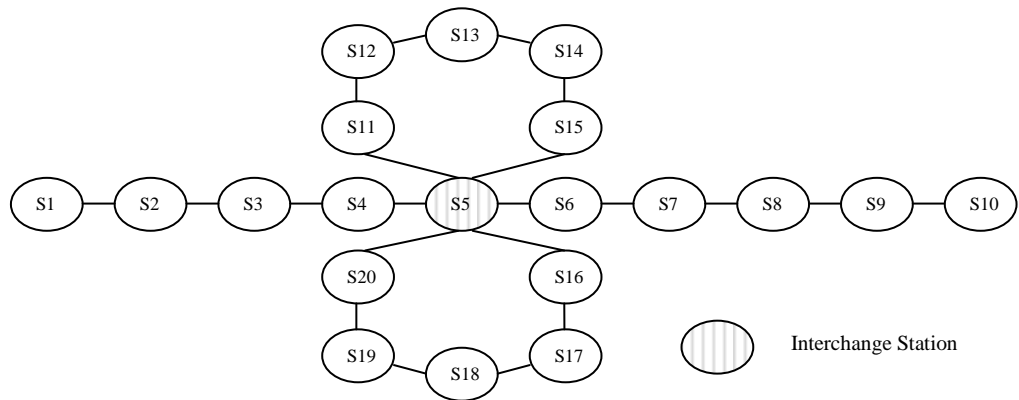


Figure 5. Random1 station map

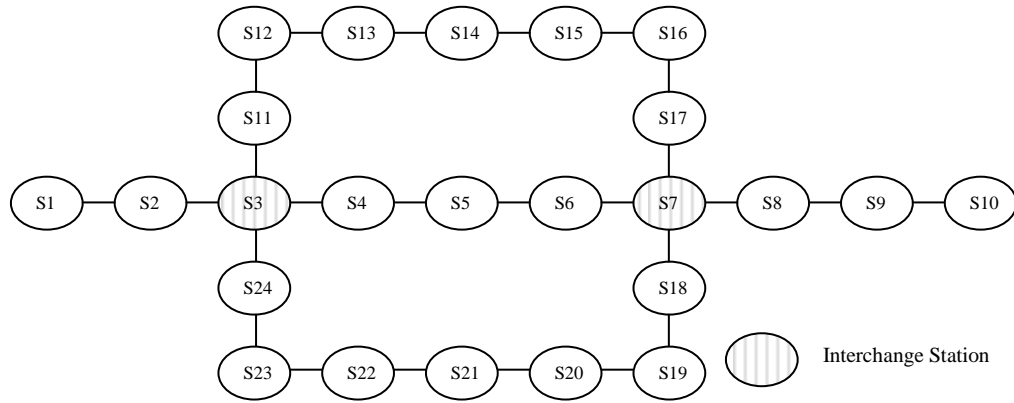


Figure 6. Random2 station map

In order to ensure the feasibility of random instances, the number of teams is set as:

$$|I| = \left\lceil \left( \sum_{j \in J} Min\_Visit_j \right) / 6 \right\rceil \tag{22}$$

For simplicity in our experiments, we assume that the break periods for all teams occur at periods  $Start_t + 2$  and  $Start_t + 5$ , meaning that the break periods are at the third and sixth periods of an eight-period shift. In the following, we report a suite of computational results and analysis obtained from our mathematical model described above. We also conduct some additional experiments by varying the values of some parameters that would be described in Section 5.3.

## 5.2 Results of the Deterministic Model

### 5.2.1 Results of Random1 and Random2 Instances

Both Random1 and Random2 can be optimally solved by the CPLEX 10.0 solver engine. The following tables summarize the schedules of all teams. It is observed that all teams are not required to change to another line for both instances. This situation provides us the minimum total distance travelled for all teams. In Random1, three stations, S1, S5 and S10, are required to be visited twice a day. Here, we found that this requirement is satisfied. Similar observation can be obtained for Random 2 where S1, S3, S7 and S10 have to be visited twice as well. The total runtimes for both random instances are 111 and 155 seconds, respectively.

Teams	Time Periods															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	S1	S2	Break	S3	S7	Break	S8	S10								
2	S17	S18	Break	S19	S20	Break	S5	S16								
3									S5	S15	Break	S14	S13	Break	S12	S11
4									S10	S9	Break	S6	S4	Break	S3	S1

Figure 7. Result of Random1 instance

Teams	Time Periods															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	S6	S5	Break	S4	S3	Break	S2	S1								
2	S10	S8	Break	S7	S17	Break	S16	S15								
3									S14	S13	Break	S12	S11	Break	S3	S1
4									S19	S20	Break	S21	S22	Break	S23	S24
5									S10	S9	Break	S8	S7	Break	S18	S19

Figure 8. Result of Random2 instance

### 5.2.2 Results of Case Study

This is a large-scale problem with 90 stations, which could not be solved in CPLEX after 24 hours. We decompose the problem into four sub-problems where each sub-problem represents a single line. In our case study, there are four different lines, namely the East-West Line, North-South Line, North-East Line and Circle Line (Table 2). The number of teams allocated to each line is defined by equation (22). The details of lines and station names can be found in <http://www.smrt.com.sg>. In this network, some stations are interchange stations (such as Jurong East, Dhoby Ghaut, Buona Vista stations and so on) which serve more than one lines. By solving each line separately, there is a possibility that these interchange stations could be visited by more than one teams at the same time. This situation is acceptable since an interchange station is generally a large station with multiple platforms. For instance, the Buona Vista station has two different platforms for trains serve East West and Circle Lines. On the other hand, we ensure that other stations (non-interchange stations) may only be visited by at most one team at a particular time period.

Figure 9 summarizes the detailed route taken by each team for the entire network. In general, stations visited by each team are close to each other (in accordance with the minimum total distance objective we define for our model). We divide the number of teams for each line into two different groups, Groups I and II. The teams in Group I would start their duty at time period 1 while others in Group II would be at time period 9 (represent two different shifts). The runtimes for each line are as follows: 780 seconds (East West Line), 200 seconds (North South Line), 33 seconds (North East Line), and 8,322 seconds (Circle Line). Solving the mathematical model for the North East Line takes significantly less runtime than solving for the other lines since it has less number of teams and stations.

Table 2. Characteristics of lines in Case Study

Lines	Number of teams	Number of stations	Number of interchange stations	Number of time periods per day	Number of stations visited per team
East West Line	7	31	7	16	6
North South Line	6	25	7	16	6
North East Line	4	16	6	16	6
Circle Line	7	30	7	16	6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
East West Line																
1	Paya Lebar	Kallang	Break	Lavender	Bugis	Break	City Hall	Raffles Place								
2	Changi Airport	Expo	Break	Tanah Merah	Simei	Break	Tampines	Pasir Ris								
3	Tanah Merah	Bedok	Break	Kembangan	Eunos	Break	Paya Lebar	Aljuneid								
4	Chinese Garden	Jurong East	Break	Clementi	Dover	Break	Buona Vista	Commonwealth								
5									Jurong East	Chinese Garden	Break	Lakeside	Boon Lay	Break	Pioneer	Joo Kon
6									Redhill	Tiong Bahru	Break	Outram Park	Tanjong Pagar	Break	Raffles Place	City Hall
7									Buona Vista	Commonwealth	Break	Queenstown	Redill	Break	Tiong Bahru	Outram Park
North South Line																
1	Marina Bay	Raffles Place	Break	City Hall	Dhoby Ghaut	Break	Somerset	Orchard								
2	Kranji	Yew Tee	Break	Choa Chu Kang	Bukit Gombak	Break	Bukit Batok	Jurong East								
3	Bishan	Ang Mo Kio	Break	Yio Chu Kang	Khatib	Break	Yishun	Sembawang								
4									Newton	Novena	Break	Toa Payoh	Braddell	Break	Bishan	Ang Mo Kio
5									Admiralty	Woodlands	Break	Marsiling	Choa Chu Kang	Break	Bukit Batok	Jurong East
6									Orchard	Somerset	Break	Dhoby Ghaut	City Hall	Break	Raffles Place	Marina Bay
North East Line																
1	Harbour Front	Outram Park	Break	Chinatown	Clarke Quay	Break	Dhoby Ghout	Little India								
2	Potong Pasir	Woodleigh	Break	Serangoon	Buangkok	Break	Sengkang	Punggol								
3									Serangoon	Kovan	Break	Hougang	Buangkok	Break	Sengkang	Punggol
4									Boon Keng	Farrer Park	Break	Dhoby Ghaut	Clarke Quay	Break	Outram Park	Harbour Front
Circle Line																
1	Holland Village	Buona Vista	Break	One North	Kent Ridge	Break	Labrador Park	Harbour Front								
2	Harbour Front	Telok Blangah	Break	Labrador Park	Pasir Panjang	Break	Haw Par Villa	Buona Vista								
3	Dhoby Ghaut	Bras Basah	Break	Esplanade	Promenade	Break	Bay Front	Marina Bay								
4	Bishan	Lorong Chuan	Break	Serangoon	Bartley	Break	Tai Seng	Paya Lebar								
5									Farrer Road	Botanic Garden	Break	Caldecott	Marymount	Break	Bishan	Serangoon
6									Dhoby Ghaut	Bras Basah	Break	Esplanade	Promenade	Break	Bay Front	Marina Bay
7									Nicoll Highway	Stadium	Break	Mountbatten	Dakota	Break	Paya Lebar	Mac Pherson

Figure 9. Result of Case Study



### 5.3 Results of the Randomized Strategy

First, we generate different instances by randomizing the start and break times (Table 3). Note that the number of teams allocated has to be adjusted according to equation (22) in order to ensure feasibility. Scenario 1 is the base problem that has been solved and shown in Figure 9. Scenario 2 is generated by varying the start time for each team. The same start time for all teams is set for Scenario 3. Finally, we increase the number of visits for some stations in Scenario 4. It turns out that the number of teams has to be increased by one team. We present and discuss the results of the East West line of the Singapore network.

Table 3. Randomized start and finish times

East West Line	Number of teams	Start and finish times for each team (team no [start – finish])
Scenario 1	7	1[1-8]*, 2[1-8], 3[1-8], 4[1-8], 5[9-16], 6[9-16], 7[9-16]
Scenario 2	7	1[1-8], 2[1-8], 3[3-10], 4[3-10], 5[7-14], 6[9-16], 7[9-16]
Scenario 3	7	1[1-8], 2[1-8], 3[1-8], 4[1-8], 5[1-8], 6[1-8], 7[1-8]
Scenario 4	8	1[1-8], 2[1-8], 3[3-10], 4[4-11], 5[5-12], 6[6-13], 7[8-15], 8[9-16]

\*1[1-8]: Team 1 would start at time period 1 and finish at time period 8

As mentioned earlier, Scenario 1 with only two values of start time periods: time periods 1 and 9, could be solved within 780 seconds. When the start time for each team is randomly set to several time periods (Scenario 2), it turns out that the problem could be solved faster (within 629 seconds). On the other hand, if all teams have to patrol at the same time (Scenario 3), the runtime significantly increases to 6,400 seconds. When we increase the number of visits for some stations and change the start time for each team randomly (Scenario 4), the runtime is up to 17,078 seconds. Increasing the number of visits seems to make the problem harder. Similar observations can be obtained for other lines.

The next set of experiments is related to randomizing the break times. Initially, we assume that the break periods for all teams occur at periods  $Start_t + 2$  and  $Start_t + 5$ , where the values of  $Break_t^1$  and  $Break_t^2$  are 2 and 5, respectively. Table 4 summarizes different values of the break times. In Scenario 5, each team might have different values of  $Break_t^1$  and  $Break_t^2$  with a constant gap between both values ( $Break_t^2 - Break_t^1 = 3$ ), while in Scenario 6, the gap is not constant.

Table 4. Randomized  $Break_t^1$  and  $Break_t^2$

East West Line	Number of teams	$Break_t^1$ and $Break_t^2$ for each team (team no [ $Break_t^1$ – $Break_t^2$ ])
Scenario 5	7	1[2&5]*, 2[2&5], 3[3&6], 4[3&6], 5[2&5], 6[2&5], 7[3&6]
Scenario 6	7	1[1&3], 2[2&5], 3[3&5], 4[2&5], 5[1&4], 6[2&5], 7[3&6]

\*1[2&5]: Team 1 would have two breaks at  $(Start_t+2)$  and  $(Start_t+5)$

The runtime for both scenarios, Scenario 5 and Scenario 6, are 33,240 and 28,017 seconds respectively. It seems that both scenarios are more difficult to solve compared with the previous scenarios in Table 3. When the break times for each team are randomly set to different values (Scenario 6), the runtime is less than that of Scenario 5.

Finally, we report results on randomizing the visit frequencies. As described in Section 4.2, assuming that the probability distribution of a crime is known, our goal is to simulate a number of realizations of crime occurrences based on this distribution, and evaluate the effectiveness of our solutions in crime deterrence against those generated by a fixed visit frequency. For this purpose, we choose the North East Line (with a total of 16 stations). The probability distribution  $P(\text{crime occurs at Station})$  is presented in Table 5 column 3. We generate 100 realizations, and for each realization, the randomized vector for the minimum number of visits required for each station ( $Min\_Visit$ ) is as shown in Table 5.

Table 5. Simulating crime and visits at different stations

Station	Station Name	Pr(crime occurs at Station)	$Min\_Visit$ (1)	$Min\_Visit$ (2)	...	$Min\_Visit$ (100)
1	Harbour Front	0.08	2	3	...	0
2	Outram Park	0.07	1	2	...	0
3	China Town	0.02	1	1	...	1
4	Clarke Quay	0.09	1	1	...	3
5	Dhoby Ghaut	0.13	3	2	...	3
6	Little India	0.06	1	1	...	1
7	Farrer Park	0.06	1	1	...	1
8	Boon Keng	0.04	1	1	...	0
9	Potong Pasir	0.04	1	1	...	2
10	Woodleigh	0.07	2	1	...	4
11	Serangoon	0.10	1	4	...	1
12	Kovan	0.08	4	1	...	3
13	Hougang	0.04	0	1	...	2
14	Buangkok	0.04	1	1	...	1
15	Sengkang	0.06	1	1	...	2
16	Punggol	0.04	3	2	...	0
Total		1	24	24	...	24

For each realization, we randomly generate the station where the crime occurs (according to the probability distribution). The randomized strategy is said to effectively deter the crime occurring that station if the  $Min\_Visit$  value for that station exceeds that of the fixed strategy, and vice versa; otherwise, we have a tie. For convenience, we set the  $Min\_Visit$  vector for the fixed strategy to be the  $Min\_Visit$  vector of the first realization (i.e. Table 5 column 4). The entire simulation results are summarized in Table 6. In this table, we set the value of 1 for a particular replicate if the randomized model is more effective; otherwise 0. If a tie exists, we use the word “tie”.

We observe that our randomized strategy can perform better than the fixed strategy which is based on the fixed strategy. Of the 100 replicates, the randomized strategy provides 53% successful deterrence versus 25% for the fixed strategy. Both are tied at 22% of the runs.

Table 6. Simulation results for North East Line

Run	Crime at station	<i>Min_Visit</i>		Result	Run	Crime at station	<i>Min_Visit</i>		Result
		Randomized Strategy	Fixed Strategy				Randomized Strategy	Fixed Strategy	
1	5	3	3	tie	51	5	5	3	1
2	11	4	1	1	52	15	2	1	1
3	12	5	4	1	53	4	1	1	tie
4	5	5	3	1	54	11	3	1	1
5	11	3	1	1	55	1	2	2	tie
6	15	1	1	tie	56	5	3	3	tie
7	1	3	2	1	57	15	0	1	0
8	10	3	2	1	58	10	2	2	tie
9	15	3	1	1	59	4	3	1	1
10	9	0	1	0	60	6	1	1	tie
11	5	3	3	tie	61	4	3	1	1
12	4	3	1	1	62	3	1	1	tie
13	4	3	1	1	63	12	4	4	tie
14	1	1	2	0	64	5	6	3	1
15	4	4	1	1	65	15	0	1	0
16	1	3	2	1	66	15	4	1	1
17	13	1	0	1	67	15	1	1	tie
18	5	5	3	1	68	5	2	3	0
19	13	0	0	tie	69	11	2	1	1
20	11	5	1	1	70	14	1	1	tie
21	5	8	3	1	71	13	1	0	1
22	16	1	3	0	72	1	3	2	1
23	5	4	3	1	73	14	2	1	1
24	16	4	3	1	74	2	3	1	1
25	2	0	1	0	75	15	4	1	1
26	5	4	3	1	76	9	3	1	1
27	10	2	2	tie	77	12	3	4	0
28	7	3	1	1	78	12	3	4	0
29	5	1	3	0	79	14	4	1	1
30	12	1	4	0	80	14	2	1	1
31	12	1	4	0	81	10	3	2	1
32	4	4	1	1	82	5	2	3	0
33	10	1	2	0	83	12	3	4	0
34	11	3	1	1	84	7	1	1	tie
35	5	2	3	0	85	4	0	1	0
36	12	0	4	0	86	12	4	4	tie
37	5	5	3	1	87	14	1	1	tie
38	13	2	0	1	88	15	1	1	tie
39	11	4	1	1	89	5	2	3	0
40	2	1	1	tie	90	12	0	4	0
41	1	2	2	tie	91	4	3	1	1
42	13	1	0	1	92	5	4	3	1
43	5	5	3	1	93	4	2	1	1
44	16	2	3	0	94	4	4	1	1
45	12	2	4	0	95	5	6	3	1
46	7	1	1	tie	96	11	3	1	1
47	10	1	2	0	97	5	4	3	1
48	4	3	1	1	98	1	3	2	1
49	5	5	3	1	99	11	0	1	0
50	11	0	1	0	100	14	1	1	tie

## 6 Conclusion

In this paper, we presented initial results from a research on generating patrol scheduling in mass rapid transit systems. We proposed a mathematical programming model to formulate the problem. Security is a major importance issue in the patrol scheduling problem. Deterministic schedules are undesirable due to predictable vulnerabilities. Strategic randomization is one aspect that has to be considered in this problem. In this paper, we proposed a simple randomized strategy by randomizing the start (and therefore finish times), break times for each team and the frequency of visits for each station. We reported the efficiency and effectiveness of our proposed approach

under different circumstances. We believe that our model does not require major customizations for use in other mass rapid transit systems with similar constraints and requirements.

There are many possible extensions to our work. For the purpose of reducing the computing time needed to solve the proposed model, we can consider approaches, such as strengthening its LP relaxation by adding valid inequalities or reducing the number of variables by using pricing procedures. The random start time for each team can also be obtained by sampling from marginal probability of a certain distribution (Rosenshine, 1970). This paper merely considers a simple randomization strategy for the operator, but do not take the strategic behaviour of adversaries into account. Extending our proposed model to cover adversarial aspects is a very interesting area. One approach is to consider Stackelberg game models which have been applied in a variety of security domains (Ordóñez et al., 2012, Tsai et al., 2009).

### References

1. Basar, T., & Olsder, G.J. (1995). *Dynamic Noncooperative Game Theory*. Academic Press, San Diego, CA, 2<sup>nd</sup> edition.
2. Chu, S.C.K., & Chan, E.C.H. (1998). Crew scheduling of light rail transit in Hong Kong: from modeling to implementation. *Computers and Operations Research*, 25 (11), 887-894.
3. Elizondo R., Parada V., Pradenas L., & Artigues C. (2010). An evolutionary and constructive approach to a crew scheduling problem in underground passenger transport. *Journal of Heuristics* 16(4), 575 – 591.
4. Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153, 3-27.
5. Gunawan, A., & Lau, H.C. (2012). Master physician scheduling problem. *Journal of the Operational Research Society*. To appear.
6. Hammer, P.L., & Rudeanu, S. (1968). *Boolean Methods in Operations Research and Related Areas*. Springer, Berlin.
7. Jiang, A.X., Yin, Z., Johnson, M.P, Tambe, M., Kiekintveld, C. Leyton-Brown, K., & Sandholm, T. (2012). Towards optimal patrol strategies for fare inspection in transit systems. In AAAI Spring Symposium on Game Theory for Security, Sustainability and Health, Stanford, California, 26-28 March 2012.
8. Maenhout, B., & Vanhoucke, M. (2010). A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *European Journal of Operational Research*, 206, 155-167.
9. Ordóñez, F., Tambe, M., Jara, J.F., Jain, M., Kiekintveld, C., & Tsai, J. (2012). Deployed security games for patrol planning. In *Handbook of Operations Research for Homeland Security* (Book Chapter), ed. Herrmann, J.W., Springer. To appear.
10. Petrovic, S., & Berghe, G.V. (2008). Comparison of algorithms for nurse rostering problems. In *Proceedings of the 7<sup>th</sup> International Conference of Practice and Theory of Automated Timetabling 2008*, Montreal, Canada, 18 – 22 August 2008.

11. Rosenshine, M. (1970). Contributions to a theory of patrol scheduling. *Operational Research Quarterly (1970-1977)*, 21(1), 99-106.
12. Ross, S. (2009). *A First Course in Probability*. Prentice-Hall, Upper Saddle River, NJ, 8<sup>th</sup> edition.
13. Sharma, D.K., Ghosh, D., & Gaur, A. (2007). Lexicographic goal programming model for police patrol cars deployment in metropolitan cities. *Information and Management Sciences*, 18(2), 173-188.
14. Stern, Z.S., & Teomi, Y. (1986). Multi-objective scheduling plans for security guards. *The Journal of the Operational Research Society*, 37(1), 67-77.
15. Taylor, P.E., & Huxley, S.J. (1989). A break from tradition for the San Francisco police: patrol officer scheduling using an optimization-based decision support system. *Interfaces*, 19(1), 4-24.
16. Tsai, J., Rathi, S., Kiekintveld, C., Ordóñez, F., & Tambe, M. (2009). IRIS – A tool for strategic security allocation in transportation networks. In *Proceedings of the 8<sup>th</sup> International Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, 10-15 May 2009, 37-44.

---

## Patient-to-room assignment planning in a dynamic context

Wim Vancroonenburg · Patrick  
De Causmaecker · Greet Vanden Berghe

Received: date / Accepted: date

**Abstract** The present contribution proposes an extension to the patient assignment (PA) planning problem in a dynamic context. Two ILP-models have been developed for optimizing this day- to- day planning problem. The first considers finding the optimal assignment for newly arrived patients, whereas the second also considers future, but planned, arrivals. The performance of both models is compared to each other on a set of benchmark instances. The relative performance with respect to a known lower bound is also presented. Furthermore, the effect of uncertainty on the patients' length of stay is studied, as well as the effect of the percentage of emergency patients. The results show that the second model provides better results under all conditions, while still being computationally tractable.

**Keywords** Patient assignment problem · Dynamic planning · Integer Linear Programming

### 1 Introduction

Rooms and beds belong to the critical assets of just any hospital. They account for a considerable part of a hospital's infrastructure, and a large amount of financial resources are invested in equipping them with medical apparatus to facilitate patient care. Furthermore, they also represent the place where most patients will spend a large part of their stay, as they recover from surgery, wait for examinations to take place, etc. In order to improve their comfort, patients are offered a choice between single bed rooms, luxury rooms with

---

Wim Vancroonenburg  
CODeS research group, Computer Science, KAHO Sint-Lieven  
Gebroeders De Smetstraat 1, 9000 Ghent, Belgium  
Tel.: +32 9 265 87 04  
Fax: +32 9 225 62 69  
E-mail: Wim.Vancroonenburg@kahosl.be

private showers, and other amenities. As a result, a large variety of hospital rooms exists in terms of capacity, which are equipped with different medical apparatus and amenities. Assigning patients to such a variety of hospital rooms can therefore be challenging, necessitating an efficient plan for making such an assignment.

Bed managers aim at finding an assignment of patients to rooms that strikes a balance between patients' preferences and comfort on the one hand, and patients' clinical conditions and the resulting required room facilities on the other. However, both the availability of rooms and equipment, and hospital policies and standards need to be considered, making it difficult to generate a balanced patient-to-room assignment. A lack of overview on occupied beds and the uncertainty on how long patients will stay in the hospital, further complicate the matter.

Demeester et al (2010) defined and studied the patient assignment (PA) problem in the context just described. They consider a set of patients that arrive at a hospital over a certain period of time. The hospital comprises a set of rooms, each with given capacity and characteristics. The problem is to find an effective assignment of patients to rooms, satisfying room capacity restrictions. Moreover, a perceived cost is associated with each patient to room assignment relating to the *appropriateness* of that assignment. The objective is to minimize the total cost of these assignments. The present contribution focuses on this problem.

## 1.1 Related work

As pointed out by Rais and Viana (2011) in their survey on operations research in healthcare, a great deal of the considered literature has focussed on scheduling of patients and hospital resources. Notably, nurse rostering and operating theatre (OT) planning and scheduling have received a considerable amount of attention (see e.g. Burke et al 2004; Cardoen et al 2010), which is evident given that personnel and the OT are among the most expensive resources for any hospital.

The PA problem considered in this paper comprises an assignment problem that occurs at the operational level of hospital admission offices. It assumes that patients have already been attributed an admission date, a decision that is made as part of either an intervention scheduling<sup>1</sup> process (see e.g. Riise and Burke 2010) during operational surgery scheduling, or an appointment scheduling process when no surgery is required. The type of patients and the arrival pattern of patients with different pathologies is often also largely influenced by the Master Surgery Schedule (MSS), a timetable that allocates operating rooms and operating time to different medical disciplines. For example, Beliën and Demeulemeester (2007) show how a cyclic MSS can be constructed that results in an expected levelled bed occupancy.

---

<sup>1</sup> Also known as *advance scheduling*.

Demeester et al (2010) introduced the PA problem to the academic community as a challenging combinatorial optimization problem. In a follow up paper, Bilgin et al (2012) presented a new hyper-heuristic algorithm to the PA problem. They provide new benchmark instances and report test results. Vancroonenburg et al (2011) show that the PA problem is NP-hard.

The problem was also studied by Ceschia and Schaerf (2011), who developed a Simulated Annealing algorithm that improves on the best known results for the benchmark instances. Lower bounds for these instances are provided as well. Furthermore, they argue that the problem definition only assumes patients that are planned in advance (elective patients), and that it does not capture the dynamics of uncertainty on patient arrivals and departures. An extension to the problem definition is proposed where patient admission and discharge dates are revealed a few days before they occur (denoted as the *forecast level*). To this end, Ceschia and Schaerf developed a dynamic version of their algorithm that can be used for day- to- day scheduling. The performance of this algorithm is analysed under an increasingly larger forecast level.

The PA problem where patient transfers are not allowed, is related to the interval scheduling problem: patients can be represented by fixed length intervals/jobs with fixed start and end time, that need to be assigned to a machine (a room) for ‘processing’. The PA problem comprises required jobs and non-identical machines with different capacities, the goal being to find a minimum-cost schedule subject to side-constraints. In the dynamic context, it constitutes an *online* interval scheduling problem with uncertainty on the interval lengths. We refer to Kolen et al (2007) for a review on the subject of (online) interval scheduling problems. Ouelhadj and Petrovic (2009) give a survey of dynamic scheduling problems in manufacturing in general.

## 1.2 Present contribution

Similarly to Ceschia and Schaerf (2011), we define a new extension to the PA problem in a dynamic context. To this end, registration dates for each patient are added to the problem definition to denote when a patient’s (possibly future) arrival time is revealed. Contrary to Ceschia and Schaerf (2011) however, an estimate of the *length of stay* (LOS) for each patient is also assumed to be available, which in practice often is the case. Special care is taken to accommodate the decision process when patients outstay their estimated length of stay.

This dynamic version of the problem is modelled and solved using Integer Linear Programming (ILP). We discuss the performance of this approach and study the effect of the percentage of emergency cases and the accuracy of the LOS estimate.



## 2 Problem formulation

### 2.1 PA in a static context

The PA problem considers a set of patients  $P$  that each need to be assigned to one of a set of hospital rooms  $R$  over a certain time horizon  $H = \{1, \dots, T\}$ . Each room  $r \in R$  has a given capacity, denoted by  $c(r)$ . Each patient  $p \in P$  is attributed an arrival time  $a(p)$  and a departure time  $dd(p)$ , with the time interval  $[a(p), dd(p))$  representing the patient's stay in the hospital. The length of the patient's stay,  $dd(p) - a(p)$ , is denoted as  $los(p)$ .

The problem is to find an assignment  $\sigma : P \mapsto R$  of patients to rooms that minimizes a certain cost  $w(\sigma)$  related to these assignments. This cost  $w(\sigma)$  consists of two parts:

- $w_1(\sigma) = \sum_{p \in P} los(p) \cdot c(p, \sigma(p))$  : each patient/room combination is attributed a cost  $c(p, r)$ , that relates to the *appropriateness* of assigning patient  $p$  to room  $r$  for one time interval (the lower  $c(p, r)$ , the better). The goal is to minimize the sum of these assignment costs.
- $w_2(\sigma) = \sum_{r \in R} \sum_{t=1}^T Conflict_{\sigma, r, t}$  : the sum of all *gender conflicts* in all rooms  $r$  over the entire planning horizon  $H$ . The goal is to avoid that male and female patients are assigned to the same room at the same time. These conflicts are calculated as follows:

$$Conflict_{\sigma, r, t} = \min( |p \in P_{\sigma, r, t} : p \text{ is male}|, |p \in P_{\sigma, r, t} : p \text{ is female}|) \quad (1)$$

with

$$P_{\sigma, r, t} = \{p \in P : a(p) \leq t < dd(p), \sigma(p) = r\} \quad (2)$$

denoting the set of patients assigned to room  $r$  at time  $t$ . Furthermore, this assignment should respect the room capacities at all times, i.e. :

$$\forall t = 1, \dots, T, r \in R : |P_{\sigma, r, t}| \leq c(r) \quad (3)$$

The definition proposed by Demeester et al (2010) allows for patients to be transferred from one room to another during their stay. The present contribution considers the slightly simpler version of the problem, which does not allow for transfers.

### 2.2 PA in a dynamic context

In practice, the arrivals and departures of patients are gradually revealed over the planning horizon. The problem definition is therefore extended to account for these dynamics. Each patient  $p$  is attributed a *registration* date  $r(p)$ , at which point the patient becomes known to the system, and an *expected* departure date  $ed(p)$ , which is an estimate of the patient's departure date. The departure date of the patient  $dd(p)$  however, remains hidden until the patients' departure date has passed.

At each point  $t' \in H$  of the planning horizon, two sets are revealed:

- $P_{t'}$  : the set of patients with  $r(p) = t'$ , i.e. the patients that are registered at time  $t'$ . At this point, for each patient  $p \in P_{t'}$  only  $a(p)$  and  $ed(p)$  are known,  $dd(p)$  remains hidden.
- $DP_{t'}$  : the set of patients with  $dd(p) = t'$ , i.e. the patients that leave the hospital at time  $t'$ .

Let  $A_{t'}$  denote the set of patients that arrived at  $t'$ , i.e. :

$$A_{t'} = \{p \in P : a(p) = t'\} \quad (4)$$

The goal of the problem is to find at each time  $t'$  an assignment

$$\sigma_{t'} : \bigcup_{i=1}^{t'} A_i \mapsto R \quad (5)$$

that maps each *arrived* patient  $p$  (i.e. all  $p$  for which  $t' \geq a(p)$ ) to a hospital room  $r$ . Furthermore, patients who arrived before  $t'$  should not be moved, i.e. :

$$\forall p \in \bigcup_{i=1}^{t'-1} A_i : \sigma_{t'}(p) = \sigma_{t'-1}(p) \quad (6)$$

The assignment  $\sigma_T$  denotes the solution at the end of the planning horizon. It contains all the patients' assignments within that period. The solution quality can be assessed by computing  $w(\sigma_T)$ . It is interesting to compare this value with the quality obtained for the static variant of the problem, which supposes that each patient's departure date is fixed in advance. Any lower bound for (or the optimal solution to) the static version is a lower bound for the dynamic problem.

### 3 Optimization models

Two models were developed for the dynamic PA planning problem that correspond to the situation at each decision step  $t'$ . They extend the previous assignment  $\sigma_{t'-1}$  to include available information on newly arrived patients  $p \in A_{t'}$ .

The first approach is modelled after current practice, namely the assignment decision is made shortly before patient arrival and only current room availability is considered. The model tries to find the optimal assignment for the patients who arrived at the current decision step. Moreover, it uses the estimate of the newly arrived patients' LOS. The second model builds on the previous model by also considering all registered patients at each decision step, therefore anticipating future occupancy and room demand.

Both models are implemented as ILP-models. They are described in Sections 3.1 and 3.2. To simplify the description, the following notation will be used:

- $\mathbf{P}_{t'} = \bigcup_{i=1}^{t'} P_i$ , the set of all registered patients up till  $t'$ ,

- $\mathbf{A}_{t'}$  =  $\bigcup_{i=1}^{t'} A_i$ , the set of all patients who arrived up till  $t'$ ,
- $P^M, P^F \subseteq P$ , restricts a set of patients to either *males* or *females* respectively,
- $elos(p) = \max(ed(p) - a(p), t' - a(p))$ , the expected length of stay of patient  $p$  as it is known at the decision time  $t'$ . If the patient's stay has exceeded his or her expected departure date  $ed(p)$ , he or she is expected to stay at least one day longer.
- $\mathbf{AP}_{tt'}$  =  $\{p \in \mathbf{A}_{t'} : a(p) \leq t < a(p) + elos(p)\}$ , the set of *arrived* patients that are *expected* to be present at time  $t$ ,
- $\mathbf{PP}_{tt'}$  =  $\{p \in \mathbf{P}_{t'} : a(p) \leq t < a(p) + elos(p)\}$ , the set of *registered* patients that are *expected* to be present at time  $t$ .

### 3.1 Model 1

The decision variables are defined as follows:

$$x_{p,r} = \begin{cases} 1 & \text{if patient } p \text{ is assigned to room } r, \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

$$v_{r,t} = \text{the number of } \textit{gender} \text{ conflicts in room } r \text{ at time } t \quad (8)$$

$$y_{r,t} = \begin{cases} 1 & \text{if the number of } \textit{male} \text{ patients assigned to room } r \\ & \text{at time } t \text{ is larger than or equal to the number of } \textit{female} \text{ patients,} \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

The optimization problem is then modelled as follows:

$$\text{Min } \sum_{p \in \mathbf{A}_{t'}} \sum_{r \in R} elos(p) \cdot c(p, r) \cdot x_{p,r} + \sum_{r \in R} \sum_{t=1}^T w_G \cdot v_{r,t} \quad (10)$$

s.t.

$$\sum_{r \in R} x_{p,r} = 1 \quad \forall p \in \mathbf{A}_{t'} \quad (11)$$

$$\sum_{p \in \mathbf{AP}_{tt'}} x_{p,r} \leq c(r) \quad \forall r \in R, t = 1, \dots, T \quad (12)$$

$$\sum_{p \in \mathbf{AP}_{tt}^M} x_{p,r} \geq v_{r,t} \quad \forall r \in R, t = 1, \dots, T \quad (13)$$

$$\sum_{p \in \mathbf{AP}_{tt}^F} x_{p,r} \geq v_{r,t} \quad \forall r \in R, t = 1, \dots, T \quad (14)$$

$$\sum_{p \in \mathbf{AP}_{tt}^M} x_{p,r} \leq v_{r,t} + c(r) \cdot y_{r,t} \quad \forall r \in R, t = 1, \dots, T \quad (15)$$

$$\sum_{p \in \mathbf{AP}_{tt}^F} x_{p,r} \leq v_{r,t} + c(r) \cdot (1 - y_{r,t}) \quad \forall r \in R, t = 1, \dots, T \quad (16)$$

$$\begin{aligned}
x_{p,r} &= 1 \quad \forall p \in \mathbf{A}_{t'-1}, r = \sigma_{t'-1}(p) & (17) \\
x_{p,r} &\in \{0, 1\} \quad \forall p \in \mathbf{A}_{t'}, r \in R \\
v_{r,t} &\geq 0 \quad \forall r \in R, t = 1, \dots, T \\
y_{r,t} &\in \{0, 1\} \quad \forall r \in R, t = 1, \dots, T
\end{aligned}$$

The model describes an assignment that minimizes the expected cost of the newly arrived patients (Expression (10)). Constraint (11) specifies that each arrived patient has to be assigned to a room, while constraint (12) expresses that room capacity should be respected at all times. Constraints (13), (14), (15), and (16) relate the variables  $v_{r,t}$  and  $y_{r,t}$ , forcing  $v_{r,t}$  to take on the expected value of the minimum number of either males or females in room  $r$  at time  $t$ . Finally, constraint (17) ensures that the new assignment respects the assignments of previously arrived patients.

### 3.2 Model 2

The second model defines the same decision variables as Model 1, however it differs in the set of patients for which they are defined. Whereas the  $x_{p,r}$  variables are defined for all arrived patients  $\mathbf{A}_{t'}$  in the first model, in this model they are defined for all *registered* patients  $\mathbf{P}_{t'}$ .

Another difference is that patients can be assigned to a *dummy* room, denoted as  $\perp$ . Only registered patients who have not arrived are allowed in this dummy room, to ensure feasibility of the model under an expected, future, undercapacity. These assignments are attributed a high cost  $c(p, \perp)$  in such a way that the model gives priority to a real assignment for each future arrival.

The model is defined as follows:

$$\text{Min} \sum_{p \in \mathbf{P}_{t'}} \sum_{r \in R \cup \perp} \text{elos}(p) \cdot c(p, r) \cdot x_{p,r} + \sum_{r \in R} \sum_{t=1}^T w_G \cdot v_{r,t} \quad (18)$$

s.t.

$$\sum_{r \in R} x_{p,r} = 1 \quad \forall p \in \mathbf{A}_{t'} \quad (19)$$

$$\sum_{r \in R \cup \perp} x_{p,r} = 1 \quad \forall p \in \mathbf{P}_{t'} \setminus \mathbf{A}_{t'} \quad (20)$$

$$\sum_{p \in \mathbf{PP}_{tt'}} x_{p,r} \leq c(r) \quad \forall r \in R, t = 1, \dots, T \quad (21)$$

$$\sum_{p \in \mathbf{PP}_{tt'}^M} x_{p,r} \geq v_{r,t} \quad \forall r \in R, t = 1, \dots, T \quad (22)$$

$$\sum_{p \in \mathbf{PP}_{tt'}^F} x_{p,r} \geq v_{r,t} \quad \forall r \in R, t = 1, \dots, T \quad (23)$$

$$\sum_{p \in \mathbf{PP}_{tt'}^M} x_{p,r} \leq v_{r,t} + c(r) \cdot y_{r,t} \quad \forall r \in R, t = 1, \dots, T \quad (24)$$

instance	$ P $	$ R $	$\sum_{r \in R} c(r)$	avg. occupancy (%)	planning horizon
1	652	98	286	59.69	14
5	587	102	325	49.32	14
8	895	148	441	43.90	21
10	1575	104	308	47.76	56

Table 1: Problem characteristics of the instances.

$$\sum_{p \in \mathbf{PP}_{\mathbf{tt}'}} x_{p,r} \leq v_{r,t} + c(r) \cdot (1 - y_{r,t}) \quad \forall r \in R, t = 1, \dots, T \quad (25)$$

$$x_{p,r} = 1 \quad \forall p \in \mathbf{A}_{\mathbf{tt}'-1}, r = \sigma_{\mathbf{tt}'-1}(p) \quad (26)$$

$$x_{p,\perp} = 0 \quad \forall p \in \mathbf{A}_{\mathbf{tt}'} \quad (27)$$

$$x_{p,r} \in \{0, 1\} \quad \forall p \in \mathbf{P}_{\mathbf{tt}'}, r \in R \cup \perp$$

$$v_{r,t} \geq 0 \quad \forall r \in R, t = 1, \dots, T$$

$$y_{r,t} \in \{0, 1\} \quad \forall r \in R, t = 1, \dots, T$$

The objective of the model, expression (18), is again to minimize the total assignment cost, including minimizing any possible dummy assignments. Constraints (19) and (20) specify that each arrived and registered patient should be assigned to one room, allowing for dummy assignments for future arrivals. Constraints (21) - (26) are similar to their counterparts in Model 1, this time also considering future arrivals. Constraint (27) ensures that arrived patients are not assigned to dummy rooms.

#### 4 Experimental setup

We tested the sensitivity of the two models to the following problem characteristics:

- Occupancy
- Accuracy of the length of stay estimate (see further)
- Emergency versus planned cases

For this purpose, we used a subset<sup>2</sup> of the benchmark instances for the static PA problem available from the patient admission scheduling website (De-meester 2012). The instances were extended to the dynamic problem by adding a random registration date  $rd(p)$  and an expected departure date  $ed(p)$  for each patient  $p$  over the planning horizon. See Table 1 for the characteristics of these instances. The procedure for enriching the instances is as follows:

- $ed(p)$  is selected uniformly from the interval  $[dd(p) - acc, dd(p) + acc]$  for each patient individually. If  $ed(p) \leq a(p)$ , then it is set to  $ed(p) = a(p) + 1$ . We investigate the effect of  $acc$ , i.e. the effect of the accuracy of the expected departure date estimate.

<sup>2</sup> The subset consists of instances 1,5, 8 and 10.

- $rd(p)$  is either selected uniformly from the interval  $[a(p) - T, a(p) - 1]$  for planned patients, or is set to  $a(p)$  for emergency patients. We investigate the effect of the percentage (denoted  $em$ ) emergency versus planned cases.

To test the effect of increasing occupancy, we randomly remove (uniformly selected) beds from the instances to increase the projected average occupancy. This procedure is similar to what Ceschia and Schaerf (2011) did. To maintain feasibility, we limit this increase such that the peak occupancy is never above<sup>3</sup> 100%.

A lower bound for each instance was obtained by calculating the linear relaxation of a straightforward MIP model (not included in this paper) on the static version of the problem (i.e. where everything is known *a priori*). For studying the effect of increasing occupancy, we have calculated the lower bound for every occupancy setting since it increases as beds are removed from the instance. In the figures discussed in the following section, this lower bound is denoted as LB.

The ILP models have been implemented using Gurobi 4.5.2 with a free academic license and were solved with a time limit of 300 seconds per decision step. Thus, for example, an instance with a planning horizon of 14 days (14 decision steps) is solved in at most  $14 \times 300 = 4200$  seconds.

All experiments were performed on a computer equipped with a 3.0 GHz Core2Quad processor, and 4 GB of ram memory, running Windows XP Professional (Service Pack 3). The solver was configured to use only one processing thread. The supporting code was implemented in Java 1.6.

## 5 Discussion

### 5.1 Emergency versus planned cases, and the effect of the LOS estimate

Both models were tested on all combinations of the factors  $acc$  (LOS estimate) and  $em$  (percentage emergency cases), with  $acc$  ranging from 0 time units (perfect estimate) to 5 time units (a poor estimate) and  $em \in \{0, 0.25, 0.50, 0.75, 1.0\}$ . All tests were performed on 10 randomized instances for each specified combination of the mentioned factors. The subsequent figures and tables report on the *averages* over these 10 runs.

Figure 1 shows the effect of an increasing percentage of emergencies, under a perfect LOS estimate (left column) and a poor LOS estimate (right column), for instances 1 and 5 (top and bottom row). The results show that Model 2 consistently outperforms Model 1. However, in the limit for increasing percentage of emergencies, the result of Model 2 converges to Model 1. Obviously, in the case for 100% emergency cases, no future arrivals can be planned and the model is reduced to Model 1.

---

<sup>3</sup> Note however, that hospitals do face a 100% occupancy (and higher, using unlisted beds) from time to time.

For a perfect LOS estimate, Model 1 is not sensitive to the percentage of emergency cases, as it only considers arrived patients, whether they are emergency or planned.

Under a poor LOS estimate, both models show a more erratic behaviour. This appears unrelated to the percentage of emergencies. The reason for this change is that a decision (both for Model 1 and Model 2) may turn out good or bad when patients depart earlier or later than estimated. However, Model 2 still outperforms Model 1 for both good and bad estimates.

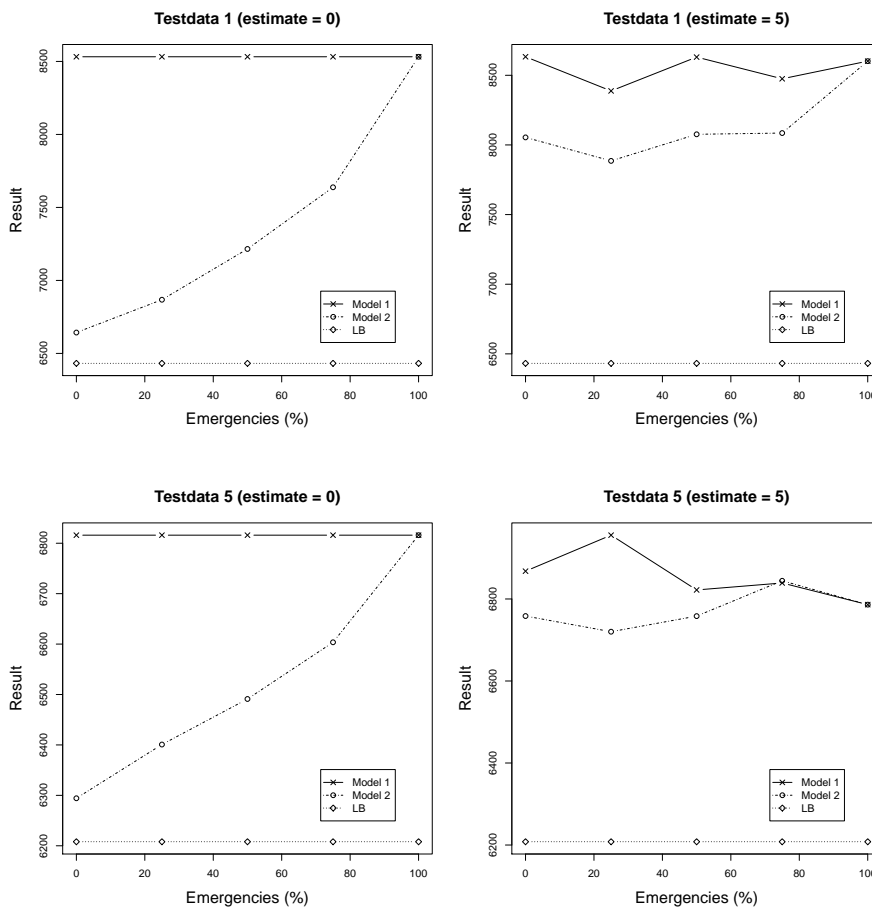


Fig. 1: Model performance for increasing percentage of emergencies, under a perfect LOS estimate (left) and a poor LOS estimate (right). Results shown for instance 1 and 5.

The above conclusions are again confirmed in Figure 2 that shows the effect of the model performance under an increasingly poorer LOS estimate, for a low to high percentage of emergencies. From the results, it follows that the performance of both models deteriorates for an increasingly poorer LOS estimate, while Model 2 always outperforms Model 1. This result was expected, as an increasing inaccuracy of the LOS estimate causes an inaccurate weighing of the patient assignments and thus suboptimal solutions. Furthermore, more patients will (significantly) go over their planned LOS, requiring future arrivals to be replanned. Again, in the limit for 100 % emergency cases, Model 2 converges to Model 1. For an overview of the results, please refer to Table 2.

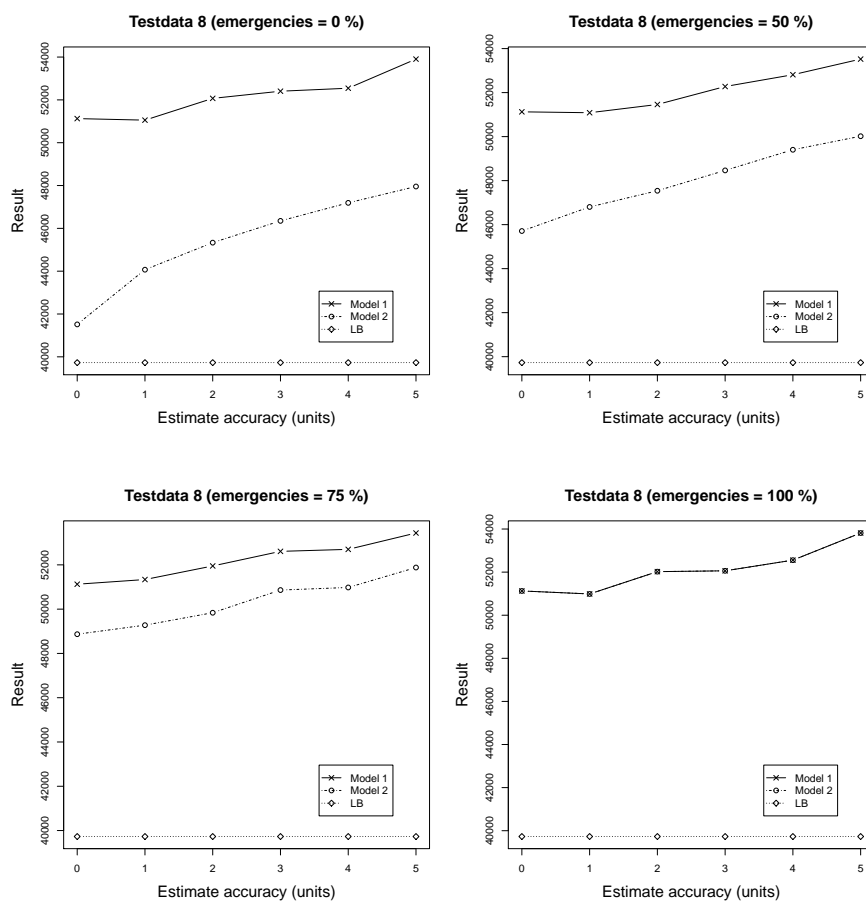


Fig. 2: Model performance for an increasing error on the LOS estimate, under a low percentage of emergencies towards a high percentage of emergencies. Results shown for instance 8.



<i>em</i> (%)	Model	Instance 1, increasing <i>acc</i>					Instance 5, increasing <i>acc</i>						
		0	1	2	3	4	5	0	1	2	3	4	5
0	1	853.2	815.6	811.1	818.3	851.0	863.3	681.6	673.9	676.9	678.9	685.6	686.8
	2	664.3	705.9	714.2	735.7	760.1	805.4	629.4	649.8	660.4	663.9	669.2	675.8
25	1	853.2	814.2	802.6	816.5	838.4	838.8	681.6	676.2	673.5	685.8	685.5	695.6
	2	686.8	708.3	721.4	729.6	755.6	788.6	640.1	659.8	662.6	668.3	676.2	672.0
50	1	853.2	822.4	820.0	821.4	840.9	863.1	681.6	677.0	676.1	676.4	684.1	682.2
	2	721.5	740.3	744.8	758.9	777.5	807.6	649.1	657.4	667.5	664.5	681.0	675.8
75	1	853.2	817.7	828.8	831.7	847.8	847.5	681.6	679.8	677.6	681.8	690.2	683.9
	2	763.8	766.8	810.0	799.1	821.7	808.5	660.4	666.0	677.0	676.8	677.4	684.4
100	1	853.2	821.0	818.1	810.0	845.9	860.2	681.6	675.6	675.9	677.2	694.2	678.6
	2	853.2	821.0	818.1	810.0	845.9	860.2	681.6	675.6	675.9	677.2	694.2	678.6

<i>em</i> (%)	Model	Instance 8, increasing <i>acc</i>					Instance 10, increasing <i>acc</i>						
		0	1	2	3	4	5	0	1	2	3	4	5
0	1	5112.6	5105.4	5207.1	5240.5	5254.5	5390.5	10347.0	10329.9	10408.1	10508.8	10688.5	10786.4
	2	4151.5	4407.0	4533.3	4634.9	4719.1	4795.5	8135.1	8705.6	9047.1	9403.4	9688.7	9805.4
25	1	5112.6	5090.4	5207.3	5200.0	5249.7	5390.7	10347.0	10333.0	10443.7	10561.9	10689.0	10782.9
	2	4361.4	4526.9	4610.4	4704.8	4790.6	4898.0	8592.7	8999.7	9299.0	9468.7	9674.5	9893.0
50	1	5112.6	5108.5	5146.0	5227.6	5281.2	5352.2	10347.0	10389.4	10383.9	10548.5	10693.6	10853.0
	2	4571.0	4680.4	4754.0	4846.5	4940.3	5001.8	9386.8	9578.4	9717.7	10003.6	10125.7	10235.2
75	1	5112.6	5133.7	5195.1	5261.0	5270.0	5343.9	10347.0	10347.7	10448.4	10552.5	10626.8	10806.0
	2	4886.9	4927.6	4983.8	5086.2	5097.9	5188.0	9959.4	10093.1	10199.9	10295.8	10442.9	10604.2
100	1	5112.6	5098.7	5202.2	5206.1	5255.6	5381.5	10347.0	10345.0	10427.4	10525.3	10644.7	10743.7
	2	5112.6	5098.7	5202.2	5206.1	5255.6	5381.5	10347.0	10345.0	10427.4	10525.3	10644.7	10743.7

Table 2: Performance comparison of Model 1 and Model 2 under different percentages of emergencies and a worsening LOS estimate. Results shown for instances 1, 5, 8 and 10.

## 5.2 Effect of increasing occupancy

The effect of an increasing occupancy was tested by artificially forcing a higher, average, occupancy in instances 1 and 5, ranging from 59% to 77% for instance 1 and from 49% to 67% for instance 5. Both instances reach a peak occupancy of 100%. Again, all combinations of factors were tested 10 times to reduce random effects. The following results report on the averages of those 10 runs.

Figure 3 shows the effect of an increasing occupancy on the performance of both models, under an increasing percentage of emergencies (from left-to-right, top-to-bottom) for instance 1. It is clear that both models perform worse under an increasing occupancy. However, the lower bounds of the instances also increase as beds are removed from the instances. Thus, the relative performance of the models compared to the lower bound does not change, indicating that occupancy does not have an effect on what the models can achieve. Figure 4 shows the same effect for instance 5.

## 6 Conclusion

In the present contribution, a dynamic version of the patient assignment problem that models a day- to- day planning process at hospital admission offices was proposed. The problem definition extended the previous, static, definition to account for the dynamics of *online* patient arrivals, including emergency patients, and explicitly models the LOS of patient as an *estimate*.

Two ILP models were developed: one that is modelled after current practice, namely assigning patients to rooms as they arrive, and one that also accounts for future planned arrivals. The first model improves on current practice by also considering the expected LOS of patients, therefore enabling a proper weighing of patient assignments. The second model also accounts for future, planned, arrivals in order to weigh patient assignments even better.

Experimental results showed that the second model can still be solved efficiently using a commercial MIP solver (under 5 minutes per scheduling step), outperforming the first model as it considers more available information on future arrivals. Furthermore, experimentation with the percentage of emergency patients, poorer LOS estimates and an increasing hospital occupancy indicate that this behaviour does not change under these conditions, advocating the use of model two over model one.

**Acknowledgements** Research funded by a Ph.D. grant of the Agency for Innovation by Science and Technology (IWT).

## References

- Beliën J, Demeulemeester E (2007) Building cyclic master surgery schedules with leveled resulting bed occupancy. *European Journal of Operational Research* 176(2):1185–1204, DOI 10.1016/j.ejor.2005.06.063

- Bilgin B, Demeester P, Misir M, Vancroonenburg W, Vanden Berghe G (2012) One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics* pp 1–34, DOI 10.1007/s10732-011-9192-0
- Burke EK, De Causmaecker P, Vanden Berghe G, Van Landeghem H (2004) The state of the art of nurse rostering. *Journal of Scheduling* 7:441–499, DOI 10.1023/B:JOSH.0000046076.75950.0b
- Cardoen B, Demeulemeester E, Beliën J (2010) Operating room planning and scheduling: A literature review. *European Journal of Operational Research* 201(3):921–932, DOI 10.1016/j.ejor.2009.04.011
- Ceschia S, Schaerf A (2011) Local search and lower bounds for the patient admission scheduling problem. *Computers & Operations Research* 38(10):1452–1463, DOI 10.1016/j.cor.2011.01.007
- Demeester P (2012) Patient admission scheduling problem website. Online, URL <http://allserv.kahosl.be/~peter/pas/>
- Demeester P, Souffriau W, De Causmaecker P, Vanden Berghe G (2010) A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine* 48(1):61–70, DOI 10.1016/j.artmed.2009.09.001
- Kolen AWJ, Lenstra JK, Papadimitriou CH, Spieksma FCR (2007) Interval scheduling: A survey. *Naval Research Logistics (NRL)* 54(5):530–543, DOI 10.1002/nav.20231
- Ouelhadj D, Petrovic S (2009) A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* 12:417–431, DOI 10.1007/s10951-008-0090-8
- Rais A, Viana A (2011) Operations research in healthcare: a survey. *International Transactions in Operational Research* 18(1):1–31, DOI 10.1111/j.1475-3995.2010.00767.x
- Riise A, Burke EK (2010) Local search for the surgery admission planning problem. *Journal of Heuristics* 17(4):389–414, DOI 10.1007/s10732-010-9139-x
- Vancroonenburg W, Goossens D, Spieksma FCR (2011) On the complexity of the patient assignment problem. Tech. rep., KAHO Sint-Lieven, Gebroeders De Smetstraat 1, Gent, Belgium, URL <http://allserv.kahosl.be/~wimvc/pas-complexity-techreport.pdf>

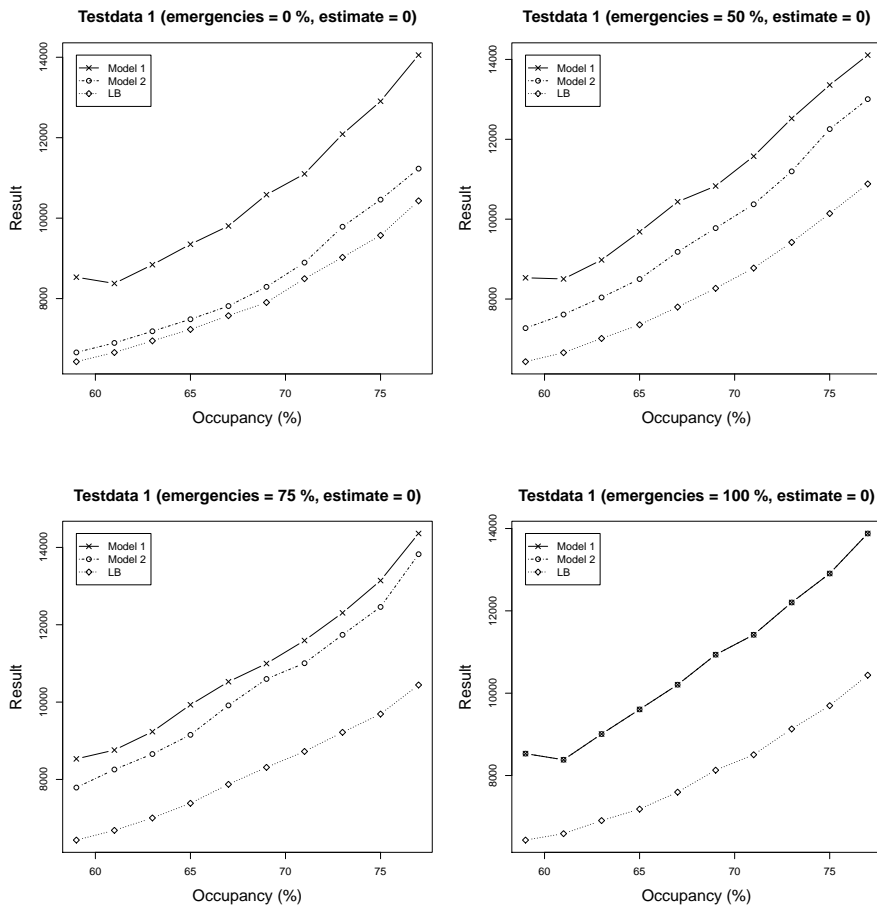


Fig. 3: Model performance for an increasingly higher occupancy rate, under different levels of emergency vs planned patients. Results shown for instance 1 with a perfect estimate.

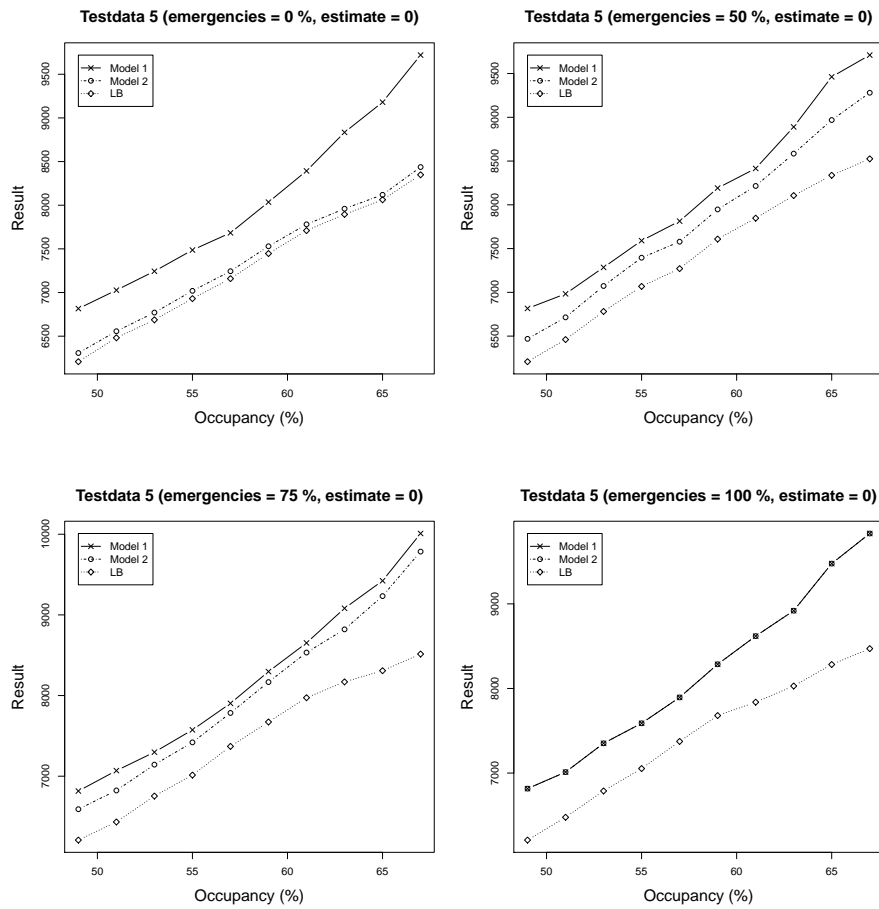


Fig. 4: Model performance for an increasingly higher occupancy rate, under different levels of emergency vs planned patients. Results shown for instance 5 with a perfect estimate.

---

# Decomposing the High School Timetable Problem

Christos Valouxis<sup>1,4</sup>, Christos Gogos<sup>2,4</sup>, Panayiotis Alefragis<sup>3,4</sup> and Efthymios Housos<sup>4</sup>

<sup>1</sup>*Secondary Education Office, Patras Branch, Greece*

<sup>2</sup>*Department of Finance and Auditing, Technological Educational Institute of Epirus, Psathaki, Preveza, Greece*

<sup>3</sup>*Department of Telecommunication Systems and Networks, Technological Educational Institute of Mesolonghi, Varia, Nafpaktos, Greece*

<sup>4</sup>*Department of Electrical and Computer Engineering, University Of Patras, Rio Patras, Greece*

**Abstract:** The process of timetable construction is a common and repetitive task for High Schools worldwide. In this paper a generic approach is presented for Greek High Schools organized around the idea of solving a significant number of tractable Integer Programming problems. Variables of the underlying mathematical model correspond to daily teacher schedules while a number of hard and soft constraints are included so as for the model to handle practical aspects that manifest themselves in Greek High Schools. By selecting better teacher schedules that exist in sub-problems the quality of the overall solution gradually improves. The collected results which are obtained within reasonable time are most promising. The strength of the approach is supported by the fact that it managed to find the best known results for two public instance problems included in the Benchmarking Project for High School Timetabling (XHSTT-2012<sup>1</sup>).

*Keywords:* high school timetabling, integer programming

## 1. Introduction

Timetabling problems manifest themselves across various domains of practice and research and can be described as the task of allocating resources to available time slots so as a set of constraints are satisfied. Additionally, a set of quality features renders alternative timetables superior or inferior with respect to each other. Timetabling problems in their general form belong to the class of NP-complete problems giving little hope of finding an algorithm that produces an optimal solution in polynomial bounded time. Nevertheless, specific timetabling problems with great practical interest can be solved satisfactorily. Therefore, a wealth of solution approaches has been studied originating mainly from Mathematical Programming, Computational Intelligence and Metaheuristics.

---

<sup>1</sup> <http://www.utwente.nl/ctit/hstt/archives/XHSTT-2012/>

Educational Timetabling problems are a specialization of timetabling problems. They have been studied in detail and even before the new millennium a plethora of approaches have been proposed as can be consulted in (Schaerf, 1999). The volume of papers published since then about the subject has been increased steadily. Educational Timetabling problems can be broadly classified in three main types:

- High School Timetabling problems: Sets of students forming classes have to be scheduled in teaching hours involving specific availability and specialization of teachers so as to generate a feasible and balanced timetable. The underlying model of High School Timetabling has been formulated as an edge coloring problem on a bipartite graph. Under this formulation the abstract problem can be polynomially solvable. Nevertheless, the addition of real life constraints makes it NP-complete. The edge colouring formulation of the problem can be traced in (Csiman, 1971) and (Bondy and Murty, 1976) with the latter having references to (de Werra, 1970) and (Dempster, 1971).
- University Course Timetabling problems: This problem can be seen as a specialization of the previous type with the difference of students that can belong to more than one classes. The main objective is to minimize lecture overlaps involving the same students.
- Examination Timetabling problems: This problem involves exams undertaken by sets of students and the goal is to reduce instances of students having to take part in more than one examinations simultaneously and evenly spread the exams over the whole exam period.

A significant number of papers about the High School problem in general and the Greek variant of it in particular have been published. Some representative papers are (Valouxis and Housos, 2003), (Burke et al, 2007), (Beligiannis et al, 2008), (Birbas et al, 2009) and (Liu et al, 2009). In this contribution we address the Greek High School Problem through a two phase process that incrementally solves parts of the problem. A feasible solution is progressively formed using solutions obtained through mathematical programming. Then, better solutions are located by keeping parts of the schedule fixed and changing the remaining schedule. The process gradually converges to better schedules. A similar methodology has been applied by our team for the nurse rostering problem with exceptionally good results. The interested reader is forwarded to the related paper (Valouxis et al, 2012).

The paper is structured as follows. Section 2 presents the problem as it occurs in Greek High schools. Constraints are categorized in hard and soft ones and a case study of a specific High school is documented. Section 3 gives the details of our approach by presenting the mathematical model used in the two phases. Then both phases are analyzed. Section 4 discusses implementation issues of our approach. Section 5 analyzes the experimental results obtained. Finally, section 6 provides conclusions drawn from our approach.

## 2. Problem Description

In Greek High schools, students have to attend a three year curriculum and in each year students are grouped in class sections alphabetically. Lectures are given from Monday to Friday and the teaching daily hours must be six or seven without idle hours in between for all students. The lectures that must be given to every class are predefined and common across Greece. Most of the lectures are given to students belonging to the original class sections by a single teacher for each subject. However, class sections can be split so as each group to attend a different course or to be merged with groups from other class sections in order to form temporary class sections attending certain courses. This usually occurs in courses like foreign languages where class sections are partitioned subject to the knowledge level of the students or subject to each student preference amongst the foreign language options provided by their school (usually English and French but also other languages in some schools). Class splitting also occurs for lessons that have to be attended in rooms with smaller capacity than that of the full number of a class section. Such courses are chemistry, technology and informatics which take place in laboratories. In order to have a full day schedule for all students of a class section that is split in two groups for a lesson like chemistry the first group might attend chemistry while the second group might attend different lesson like gymnastics or career guidance. Then at another time slot courses taught to each group appear to be swapped. Every teacher is qualified to teach a subset of the courses while the teaching load is different across teachers depending on seniority, motherhood and other conditions. It is possible for a teacher not to teach exclusively in a specific school and to have to complement his weekly work duty by teaching to more than one school. If that is the case then he is available for teaching to each school for certain days only.

In the model of the problem examined every class has its own predefined room but certain rooms or other resources like video projectors might be shared by more than one classes. Teachers are assigned for certain hours to each class section before the creation of the timetable. As, usually is the case in combinatorial optimization problems two categories of constraints can be identified: hard constraints and soft constraints. Hard constraints must hold for a solution to be considered valid while soft constraints violations degrade the quality of the proposed solution. Hard constraints of the problem are the following:

- HC1: Each teacher should teach a specified number of hours to each class.
- HC2: Each teacher should teach only at days that he is present at school.
- HC3: Each teacher should teach only to one class per time period.
- HC4: Each teacher should teach at least one hour for the days that he is present at school.
- HC5: Each class should have no more than one lesson per time period.
- HC6: Lectures of the same lessons should not be positioned in the same day.
- HC7: Empty time periods for each class section should only be positioned at the last period of each day.



- HC8: A resource should not be consumed more times than its availability in each time period.

On the other hand soft constraints are:

- SC1: Each teacher should have the smallest possible number of idle periods in his schedule. An idle period for a teacher is defined as a not busy period with a busy period earlier in the day and another busy period later in the day.
- SC2: Each teacher should have a balanced daily work schedule.
- SC3: Teacher preferences for teaching in early or in late hours should be respected to the highest possible degree. This constraint can also serve as an indirect way of giving preference to certain courses to be positioned in early or late time periods (e.g. Mathematics in the early hours).

Throughout the paper a specific problem instance depicted in Tables 1 and 2 will be presented and analyzed so as to better grasp the underlying concepts of the problem. This problem comes from the “3rd Gymnasium” High School in the city of Patras, Greece for the period from September 2010 to June 2011. It is about a medium size by Greek standards school with 9 classes, 29 teachers giving 340 lessons either to entire classes or to classes formed from student groups coming from different classes. Each class has to attend 7 teaching hours per day. For those teachers that have preferences for teaching early or late hours in some days this is marked in Table 1 with the letter E or L respectively. The letter X implies that the teacher is not available in this day. When a teacher prefers early teaching in a day this means that the solution will be penalized if he has to teach in hours 5 to 7 of this day. Likewise when a preference is late teaching assignments in hours 1 to 3 gets penalized. Furthermore, each teacher should have a balanced scheduled across the days that he is present at school. So lower and upper bounds, shown under column “Low” and column “High” in Table 1, are imposed. Columns A1A2, A3A4, B1B2, C1C2 and C2C3 denote class formations generated by joining groups of students between classes. For example, teachers T15 and T16 teach to mutually exclusive groups of students from classes A1 and A2 subject “English Language” for 3 hours each week.

Teacher	Week					Hours per day		Classes													Total		
	M	T	W	T	F	Low	High	A1	A2	A3	A4	A1A2	A3A4	B1	B2	B1B2	C1	C2	C3	C1C2		C2C3	
T0	E	E	X		E	1	1		2		2												4
T1	L	L	L	E		1	2										2	2	2				6
T2	E	L	E	E	X	2	4	2	2	2	2			2	2								12
T3	L	E		L	E	2	4		3	10	2												15
T4		E	L	L	L	2	4										3	12					15
T5	E		E	E	L	2	4	2							5		9						16
T6		L	L	L	E	2	4		2						4				9				15
T7	E			L		2	4		5		3			9									17
T8		E	E	E	E	2	4	8			5								3				16
T9	X				L	1	1								4								4
T10	L	L	L		L	2	4							4			4	4	4				16
T11	L	L	L	L		2	4	4	4	4	4												16
T12	E	L		L	E	2	4	2		2				3	3		1	1	1				13
T13	E	E	E		L	1	3							2	2		2	2	2				10
T14	L	E		X	L	3	5	2	2	2	2						2	2	2				14

T15	L	L	L	E		1	3					3	3			2			2	2			12
T16	E		E	L	E	2	4					3	3	2	2	2			2	2			16
T17		E	E		E	2	4	4	4	2	2			2								2	16
T18	E	L		X	E	1	3							2				2				2	8
T19	L	L	E	E		1	2			2	2					2							6
T20	X	X	L	X	L	1	3		2								2						4
T21		X	X	E		2	4	1	1	1	1			1	1		1	1	1				9
T22	L	X	L	L		3	5	1	1	1	1			2	2		2	2	2				14
T23		L	L	E	L	3	5	3	3	3	3			3			2	2					19
T24	E	E	E	L		1	1									3				2			5
T25		E		L	E	1	3	1	1	1	1			1	1		1	1	1				9
T26	L		E	E	L	2	4	2	2	2	2			2	2		1	1	1				15
T27	L		L	E	L	1	3	2	2	2	2						1	1	1				11
T28	E	E	E		E	1	2							2	2		1	1	1				7

**Table 1. Teacher availabilities, preferences and teaching hours per class**

Table 2 presents a compact form of all “lessons” that have to be taught to all classes or combinations of classes. Based on the observation that a timetable consists of meetings between teachers and classes for specific durations each table value represents a number of such meetings. For example, value “T2;TH;2” in the second row under column A1 means that the entire class A1 will have 2 lessons of 1 hour duration with teacher T2 teaching subject TH and these meetings should not be scheduled in the same day.

A different situation occurs in column A1 and row 15 having value “T26,T27;TE,PL;1”. This means that class A1 will be split into two groups and teacher T26 will teach subject TE to one group while teacher T27 will simultaneously teach possibly a different subject (in this case subject PL) to the other group. This will happen 2 times: one corresponding to row 15 and a second one corresponding to row 16. These lessons can coexist in the same day but certainly in different hours. Note that if those two cells were replaced by a single one with “T26,T27;TE,PE;2” this would have implied that the lesson taught by T26 and the lesson taught by T27 could not have coexisted in the same day. This is the case with row 17, column A2 having value “T20,T17;GL,FL;2”. So, teacher T26 teaches “Technology” while teacher T27 teaches “Informatics” in subclasses formed from class A1. On the other hand teacher T20 teaches “German Language” and teacher T17 teaches “French Language” to subclasses originated from class A2. The conclusion is that while “Informatics” and “Technology” lessons can be positioned in the same day for the same group of students, this could not happen for lessons “Germany Language” and “French Language” of class A2. This occurs because in the first case all students will be taught “Informatics” and “Technology” while in the latter case students will be taught either “Germany language” or the “French Language” based on their preference.

Another situation manifests itself with value “T15,T16;AG,AG;3” of merged columns A1 and A2 in row 17. This value means that class A1 and class A2 are joined and then split into 2 groups of students. Teacher T15 takes over one group while teacher T16 takes over the other group. During the week the 3 lessons of teacher T15 should be scheduled to different days and start at the same time with the corresponding lessons of teacher T16.

	A1	A2	A3	A4	B1	B2	C1	C2	C3
1	T2;TH;2	T2;TH;2	T2;TH;2	T2;TH;2	T2;TH;2	T2;TH;2	T1;TH;2	T1;TH;2	T1;TH;2
2	T5;NE;2	T6;NE;2	T3;NE;2	T8;NE;2	T7;NE;2	T6;NE;2	T5;NE;2	T4;NE;2	T6;NE;2
3	T8;AM;2	T7;AM;2	T3;AM;2	T3;AM;2	T7;AM;2	T5;AM;2	T5;AM;2	T4;AM;2	T6;AM;2
4	T8;AR;3	T7;AR;3	T3;AR;3	T7;AR;3	T7;AR;3	T5;AR;3	T5;AR;3	T4;AR;3	T6;AR;3
5	T8;GD;3	T3;GD;3	T3;GD;3	T8;GD;3	T7;GD;2	T6;GD;2	T5;GD;2	T4;GD;2	T6;GD;2
6	T11;MA;4	T11;MA;4	T11;MA;4	T11;MA;4	T10;MA;4	T9;MA;4	T10;MA;4	T10;MA;4	T10;MA;4
7	T12;GE;2	T0;GE;2	T12;GE;2	T0;GE;2	T12;GE;2	T12;GE;2	T26;SE;1	T26;SE;1	T26;SE;1
8	T14;BI;2	T14;BI;2	T14;BI;2	T14;BI;2	T12;XH;1	T12;XH;1	T12;XH;1	T12;XH;1	T12;XH;1
9	T21;KA;1	T21;KA;1	T21;KA;1	T21;KA;1	T21;KA;1	T21;KA;1	T21;KA;1	T21;KA;1	T21;KA;1
10	T22;OO;1	T22;OO;1	T22;OO;1	T22;OO;1	T22;OO;2	T22;OO;2	T22;OO;2	T22;OO;2	T22;OO;2
11	T23;GY;3	T23;GY;3	T23;GY;3	T23;GY;3	T23;GY;3	T24;GY;3	T23;GY;2	T23;GY;2	T24;GY;2
12	T25;MO;1	T25;MO;1	T25;MO;1	T25;MO;1	T25;MO;1	T25;MO;1	T25;MO;1	T25;MO;1	T25;MO;1
13	T17;IS;2	T17;IS;2	T17;IS;2	T17;IS;2	T16;IS;2	T16;IS;2	T4;IS;3	T4;IS;3	T8;IS;3
14	T17;FL;2	T26,T27;TE,PL;1	T19;2	T26,T27;TE,PL;1	T13;PH;2	T13;PH;2	T13;PH;2	T13;PH;2	T13;PH;2
15	T26,T27;TE,PL;1	T26,T27;TE,PL;1	T26,T27;TE,PL;1	T26,T27;TE,PL;1	T17;FL;2	T26,T28;TE,PL;1	T14;BI;2	T14;BI;2	T14;BI;2
16	T26,T27;TE,PL;1	T20,T17;GL,FL;2	T26,T27;TE,PL;1	T18,T19;HI,IL;2	T26,T28;TE,PL;1	T26,T28;TE,PL;1	T28,T27;TE,PL;1	T28,T27;TE,PL;1	T15,T16;AG,AG;2
17	T15,T16;AG,AG;3		T15,T16;AG,AG;3		T26,T28;TE,PL;1	T18,T19;HI,IL;2	T20,T18;GL,HI;2		T27,T28;TE,PL;1
18					T15,T16;AG,AG;2		T15,T16;AG,AG;2		
19							T17,T18;FL,GL;2		
Total	35	35	35	35	35	35	35	35	35

**Table 2. Lessons that have to be taught. Value between semicolons refers to the subject been taught**

### 3. The Algorithm

The algorithm can be considered as a two phase approach. During the first phase IP problems are iteratively solved in order to create a schedule for one day at a time. The first phase ends when all days are solved and after a number of improvements considering each day in isolation have been tried. The second phase systematically selects pairs of days and attempts to move teaching events between days. Again a series of IP problems are generated and gradually better solutions are reached. Firstly, a formal description of the mathematical model employed in solving each problem is presented and subsequently the two phases are analyzed. The second phase can be seen as a very large-scale neighbourhood (VLSN) search. More techniques about VLSN in timetabling problems can be consulted in (Meyers and Orlin, 2007).

#### 3.1 The Mathematical Model

The basic sets used in the problem's model definition are the following:

- $T$  is the set of teachers.
- $C$  is the set of classes.
- $D$  is the set of days in the timetable, usually 5 working days of a single week.
- $H$  is the set of teaching hours in a day.
- $E$  is the set of events.
- $R$  is the set of resources that might be shared by more than one lesson. A resource can be a room of a certain type, a room with special equipment, a video projector etc.

From those sets the following subsets are derived:

- $D_t$  is the set of days that teacher  $t$  is available.
- $E_t$  is the set of events that involve teacher  $t$ .
- $W_t$  is the set of legal daily schedules for teacher  $t$  for all days that he or she is available for work at the school.

Each legal daily schedule of a teacher is a combination of events and idle hours having length  $|H|$ . The set of legal daily schedules for a teacher is formed by creating all the combinations of events selected from set  $E_t$  and idle hours. Events that should not be scheduled in the same day (e.g. events of the same course) cannot exist in the same combination. The selected daily schedules for all teachers should cover the teaching workload of all days so the problem can be categorized as a Set Covering Problem.

Let  $x_{tdw}$  be a binary variable that assumes value 1 if teacher  $t$  at day  $d$  is assigned the daily schedule  $w$  and 0 otherwise. Let  $c_{tdw}$  be the cost of the daily schedule  $w$  for teacher  $t$  at day  $d$ . This cost comprises from:

- Penalty for idle hours between work in teacher schedules.
- Penalty for deviation from the lower or upper limit of the desired total teaching hours per day.
- Penalty for teaching in an hour that is not in the preferences of the teacher.

The mathematical model of the problem is presented below:

$$\text{Minimize } \sum_{t=1}^{|T|} \sum_{d=1}^{|D|} \sum_{w=1}^{|W_t|} c_{tdw} x_{tdw} \quad [1]$$

Subject to:

$$\sum_{t=1}^{|T|} \sum_{d=1}^{|D|} x_{tdw} = 1 \quad t \in T, d \in D, w \in W_t \quad [2]$$

$$\sum_{t=1}^{|T|} \sum_{d=1}^{|D|} \sum_{w=1}^{|W_t|} a_{tdwe} x_{tdw} = 1 \quad e \in E \quad [3]$$

$$\sum_{t=1}^{|T|} \sum_{w=1}^{|W_t|} a_{tdwhc} x_{tdw} = 1 \quad d \in D, h \in H, c \in C \quad [4]$$

$$\sum_{t=1}^{|T|} \sum_{w=1}^{|W_t|} a_{tdwhr} x_{tdw} = k_e y \quad d \in D, h \in H, e \in E \text{ with } T > 1 \quad [5]$$

$$\sum_{t=1}^{|T|} \sum_{w=1}^{|W_t|} a_{tdwhr} x_{tdw} = \text{Max} Q_r \quad d \in D, h \in H, r \in R \quad [6]$$

Eq. 1 is the objective function and represents the total cost of a solution that should be minimized. It is the sum of the cost over all selected daily schedules for all teachers and for all days of the teaching week.

Constraint shown in Eq. 2 states that one daily schedule should be selected for each teacher.

Eq. 3 states that all events must be assigned meaning that each event must exist at exactly one selected daily schedule of a teacher. In this equation parameter  $a_{tdwe}$  assumes value 1 if daily schedule  $w$  of teacher  $t$  in day  $d$  includes event  $e$  or 0 otherwise.

Eq. 4 guarantees that the schedule of each class will have no idle times. So, for each day and hour 1 event must exist that involves each class. It should be noted that  $a_{tdwhc}$  is a parameter that

assumes value 1 if the daily schedule  $w$  of teacher  $t$  in day  $d$  and hour  $h$  includes an event that involves class  $c$  and 0 otherwise.

Eq. 5 handles events with more than 1 teacher and instructs them to be included in the same day and hour in the daily schedules of the affected teachers. Binary variable  $y$  assumes value 1 when  $k_e$  teachers should teach concurrently and value 0 when no concurrent teaching occurs. Parameter  $a_{tdwhe}$  assumes value 1 if the daily schedule of teacher  $t$  in day  $d$  and hour  $h$  includes event  $e$  and 0 otherwise.

Eq. 6 appears in case room or other potentially shared resources exist.  $MaxQ_r$  is the available quantity for resource  $R$  while parameter  $a_{tdwre}$  is the number of the required quantities for resource  $r$  if daily schedule  $w$  of teacher  $t$  in day  $d$  and hour  $h$  includes an event that requires resource  $r$  and 0 otherwise.

The size of the problem is too big to be solved including all possible legal daily work schedules for all teachers and all days. So, an approximation method is employed where a preliminary solution is initially created and subsequently gets improved. Problem sizes in each step of the method are relatively small and can be solved in reasonable time.

### 3.2 First Phase: Solve by Day

An initial solution is generated by solving the mathematical model of the problem considering a single day at a time. Eq. 3 of the mathematical model is no longer needed because each single day can have a subset only of the events from  $E$  scheduled in it. Each column of the mathematical model represents a work schedule of a teacher for the day under consideration.

The order of days that will be considered is determined by estimating how difficult each day timetable construction might be. A day is considered more difficult than another when less teachers with a lot of teaching hours are available.

After the day is selected a list of yet unscheduled lessons that are legitimate to be taught is constructed for each teacher. Then, all possible combinations of each list's lessons that could have been scheduled in the available hours are generated. So, for each teacher a set of legitimate work patterns is assembled with the presupposition that the size of the set is manageable. Otherwise, subsets of combinations unlikely to be included in the final solution are prematurely rejected. For each teacher's set of patterns only one will be included in the schedule of the day under question. In order to solve the first day, daily work schedules of each teacher for this day are generated. For each teacher  $t$  all events  $E_t$  are considered. After each day gets solved daily schedules for all teachers are re-computed considering events that still remain unscheduled. In solving each day every legitimate daily work schedule for a teacher  $t$  should contain all events characterized as mandatory. If not then infeasible problems will result in solving next days.

Nevertheless, for teachers with many events the number of possible daily work schedules might be overwhelming. So, an artificial upper bound to the number of generated daily work schedules that contain a specific event at a specific hour is imposed. When this bound is reached, by counting work patterns, this event is removed from the list of possible events for this hour. So, gradually the number of combinations diminishes while daily work schedules of each teacher having each event in every possible hour do exist.

When the daily work schedules of all teachers are finished generated the linear relaxation of the mathematical model can be solved. The rather small problem size results in fast solve times for each relaxed problem. When the solution of the relaxed problem shows that a margin for better solutions does exist, the integrality constraints are enforced and the problem gets solved again this time as an IP problem. A time limit of 1 minute is allowed which might be less than the time needed for solving the problem to optimality. When results are obtained from solving the IP problem, even if they are suboptimal, they are used to define the daily work schedule for each teacher.

The process continues by generating a new problem for the same day that is examined to give better cost values. The solution derived from the relaxed problem of the previous step is exploited in two ways. Firstly, a column is selected that corresponds to the daily work schedule of a single teacher with the smallest cost contribution that has also the greatest value in the solution of the relaxed LP. The set of events that are included in the selected daily schedule are decided to “freeze” meaning that these events should be scheduled thereafter in the day and hour dictated by the relaxed LP. Such decisions ease subsequent daily work schedule generations by lowering the total number of combinations that have to be considered. Secondly, dual values derived from solving the LP problem are exploited in calculating the reduced cost value of each column if included in the solution. Reduced cost of a variable is the amount by which its coefficient in the objective function has to be decreased so as to be included in the solution.

Legitimate teacher day schedules that have negative reduced cost are stored so as to be included in the next problem that will be solved. This result in gradual decrease of the cost value found by the LP solver or in the worst case achieving the same cost value as before. By selecting columns from the LP problem that lowers the value of its objective function we hope to discover better values for the IP problem too. This process repeatedly occurs during our approach. A column, that represents a teacher day schedule, is stored if its cost is better than the cost of the best solution found so far for this day. Naturally, a column with cost equal or greater cannot be included in a solution better than the solution that we already have.

The cost of the best LP solution found so far is inserted as upper bound for the objective function in the IP problem. The new problem contains columns forming the basis of the LP model, columns of the best integer solution and the new columns that have been generated. The procedure repeats until no more columns can be found that can be frozen or when no more teacher daily schedule programs can be generated.

### **3.3 Second Phase: Solve by day pairs**

The previous phase results in an initial solution where all days have complete schedules for all teachers and classes. The second phase tries to find a better solution by selecting two days and trying to move teacher events between them. So, for every pair of days under investigation new daily teacher schedules are examined that improve the cost of the two day solution and therefore the cost of the full planning period solution. Teacher schedules in the days not belonging to the selected pair remain unaffected.

The same mathematical model described previously is used but in this case it includes two instead of one day. The set of events  $E_t$  that involve teacher  $t$  is formed from the original set  $E_t$  by removing events that are scheduled in the unaffected days of the planning period. The solution procedure is the same as that of the previous phase and it continues until no improvement can be found by any two days combination.

## 4. Implementation

The implementation of our approach was undertaken in Java using the open source mathematical solver GLPK (<http://www.gnu.org/software/glpk/>). Three key modules of the software can be identified that facilitated the process undertaken in the phases described in the previous paragraphs. These are: CostEvaluator, MandatoryFinder and CombinationsGenerator. A brief description of each module's role follows.

CostEvaluator is the component that calculates the cost of each solution which might be partial or complete. Furthermore, it computes the cost of every teacher daily schedule that is subsequently used as a coefficient in the mathematical problem.

On the other hand, given a partial solution MandatoryFinder calculates the events that have to be scheduled in the day under consideration so as to avoid infeasibilities that would occur later in case that these events were left to be scheduled later. For example, suppose that a lesson have to be taught 4 times in a week and after solving the first day all events of this lesson are still unscheduled. Then, for all subsequent days an event of this lesson has to be taught. MandatoryFinder also handles more complex situations like when an event involves more than one teacher.

CombinationsGenerator is a heavily used module that is responsible for computing the combinations of events that should be scheduled in the hours available in each day. In order to reduce the number of combinations generated that occurs when the number of possible events of a teacher is large, heuristics are used so as to select a representative subset of all available combinations.

## 5. Experimental Results

The datasets used in our experiments originate from High schools in Greece. We tested our approach in several problem instances with different characteristics and in all cases a feasible solution of High quality was able to be achieved. Execution times even for bigger schools (~50 teachers, ~15 classes) were less than 20 minutes in our test computer which was an Intel i3 380M (2.53GHz) with 3GBytes RAM running Windows 7 64 bit.

Among datasets used two of them, presented in Table 3, belong to the benchmarking project for High School Timetabling. These datasets alongside with solutions achieved by our and other approaches are publicly available in (<http://www.utwente.nl/ctit/hstt/>). The benchmarking project for High School Timetabling is a joint effort from researchers across several countries so as to create a common XML standard for exchanging datasets. Description of the XML format can be consulted in (Post et al, 2012) and (Post et al, 2011). In both datasets our approach found the

optimal solution in less than 10 minutes. Dataset GreeceThirdHighSchoolPatras2010 and GreeceThirdHighSchoolPreveza2008 don't have room constraints. Nevertheless, other custom datasets including such type of constraints were solved.

Dataset	Periods	Teachers	Classes	Events	Duration
GreeceThirdHighSchoolPatras2010	35	29	9	178	340
GreeceThirdHighSchoolPreveza2008	35	29	9	164	340

**Table 3. Datasets included in the benchmarking project for High School Timetabling**

The solution schedule that has been produced and satisfied all constraints for dataset GreeceThirdHighSchoolPatras2010 is presented in Table 4. Each asterisk represents a scheduled teaching for the associated teacher and period. It can be easily observed that the generated daily schedule of each teacher consists of consecutive busy periods.

TEACHER	Mon	Tue	Wed	Thu	Fri
T0	*----- *----- ----- *----- *-----				
T1	-----* -----* -*----- -*----- -----**				
T2	**----- -----** ***----- ***----- -----				
T3	---**** ****----- -----** -----** **-----				
T4	-----* ****----- -----** -----** -----**				
T5	****----- -----** ****----- **----- -----**				
T6	-----** -----** -----** -----** -----**				
T7	****----- ****----- -----** -----** -----**				
T8	-----** ****----- ****----- ****----- **-----				
T9	----- -*----- -----** -----** -----**				
T10	-----** -----** -----** -----** -----**				
T11	-----** -----** -----** -----** -----**				
T12	**----- -----** -----** -----** ****-----				
T13	-*----- -----** ****----- -----** ****-----				
T14	-----** ****----- -----** -----** -----**				
T15	-----** -----** -----** ****----- -----**				
T16	-----** -----** -----** ****----- -----**				
T17	****----- ****----- ****----- -----** ****-----				
T18	-*----- -----** ****----- -----** ****-----				
T19	-----* -----** -----** -----** -----**				
T20	----- -----** -----** -----** ****-----				
T21	-----** -----** -----** ****----- -----**				
T22	-----** -----** -----** ****----- ****-----				
T23	****----- ****----- -----** ****----- -----**				
T24	*----- *----- *----- -----** -----**				
T25	-----** -----** ****----- -----** ****-----				
T26	****----- **----- ****----- -----** -----**				
T27	-----** *----- ****----- ****----- -----**				
T28	-----** *----- -----** *----- -----**				

**Table 4. Optimal schedule for GreeceThirdHighSchoolPatras2010**

## 6. Conclusions

Quality schedules for High Schools are vital for the success of the education effort. Obtaining manually such a schedule is unlikely to occur so computer generated solutions seem to be the only logical option. Nevertheless, practical issues often prohibit schools of operating based on an optimal schedule for students and teachers. In this contribution we presented an approach for the Greek case of the High School problem and we showed its ability to generate very good and in some cases optimal results. The approach is based on a two phase process that incrementally



solves parts of the problem, using mathematical programming. Better schedules are unmasked while portions of the schedule remain fixed. The process gradually converges to better schedules.

We acknowledge the fact that adaptation of new automated timetabling solutions by High Schools is a target hard to achieve but we believe that the quality of the schedules that we produce should entice schools in Greece to test the proposed approach. It is in our plans to offer a web service that schools in Greece will be able to use so as to enter the schedule problem as it occurs in their specific case and our application will propose a High quality schedule based on this data. We also plan to offer a re-schedule service that a High School could use so as to make changes throughout the year to an existing schedule while keeping most of it unchanged.

## References

- Beligiannis, Grigorios N., Moschopoulos N. Charalampos, Kaperonis P. Georgios, and Likothanassis D. Spiridon. "Applying Evolutionary Computation to the School Timetabling Problem: The Greek Case." *Computers & Operations Research* 35, no. 4 (2008): 1265–1280.
- Birbas, T., S. Daskalaki, and E. Housos. "School Timetabling for Quality Student and Teacher Schedules." *Journal of Scheduling* 12, no. 2 (October 28, 2008): 177–197.
- Bondy J.A. and Murty U.S.R., Graph theory with applications, North-Holland (1976).
- Burke, Edmund K., Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. "A Graph-based Hyper-heuristic for Educational Timetabling Problems." *European Journal of Operational Research* 176, no. 1 (2007): 177–192.
- Csima J. Investigations on a Time-Table Problem, PhD Thesis, Institute of Computer Science, University of Toronto (1965).
- de Werra, D. On some combinatorial problems arising in scheduling. *INFOR*, 8, 165-175.
- Dempster, M. A. H. (1971). Two algorithms for the time-table problem, in *Combinatorial Mathematics and its Applications* (ed. D. J. A. Welsh), Academic Press, New York, pp. 63-85.
- Liu, Yongkai, Defu Zhang, and Stephen C.H. Leung. "A Simulated Annealing Algorithm with a New Neighborhood Structure for the Timetabling Problem." In *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, 381–386. GEC '09. New York, NY, USA: ACM, 2009. <http://doi.acm.org/10.1145/1543834.1543885>.
- Meyers, Carol, and James Orlin. "Very Large-Scale Neighborhood Search Techniques in Timetabling Problems." In *Practice and Theory of Automated Timetabling VI*, edited by Edmund Burke and Hana Rudová, 3867:24–39. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007.
- Post, Gerhard, Jeffrey H. Kingston, Samad Ahmadi, Sophia Daskalaki, Christos Gogos, Jari Kyngas, Cimmo Nurmi, et al. "XHSTT: An XML Archive for High School Timetabling Problems in Different Countries." *Annals of Operations Research* (November 10, 2011).
- Post, Gerhard, Samad Ahmadi, Sophia Daskalaki, Jeffrey Kingston, Jari Kyngas, Cimmo Nurmi, and David Ranson. "An XML Format for Benchmarks in High School Timetabling." *Annals of Operations Research* 194, no. 1 (2012): 385–397.

Schaerf, A. "A Survey of Automated Timetabling." *Artif. Intell. Rev.* 13, no. 2 (April 1999): 87–127.

Valouxis, Christos, and Efthymios Housos. "Constraint Programming Approach for School Timetabling." *Computers & Operations Research* 30, no. 10 (2003): 1555–1572.

Valouxis, Christos, Christos Gogos, George Goulas, Panayiotis Alefragis, and Efthymios Housos. "A Systematic Two Phase Approach for the Nurse Rostering Problem." *European Journal of Operational Research* 219, no. 2 (2012): 425–433.

---

## Near-Optimal MIP Solutions for Preference Based Self-Scheduling

Eyjólfur Ingi Ásgeirsson · Guðríður Lilla  
Sigurðardóttir

Received: date / Accepted: date

**Abstract** Making a high quality staff schedule is both difficult and time consuming for any company that has employees working on irregular schedules. We formulate a mixed integer program (MIP) to find a feasible schedule that satisfies all hard constraints while minimizing the soft constraint violations as well as satisfying as many of the employees' requests as possible. We present the MIP model and show the result from four real world companies and institutions. We also compare the results with those of a local search based algorithm that is designed to emulate the solution strategies when the schedules are created manually.

The results show that the using near-optimal solutions from the MIP model, with a relative MIP gap of around 0.01-0.1 allows us to find very good solutions in a reasonable amount of time that compare favorably with both the manual solutions and the solutions found by the local search based algorithm.

**Keywords** Staff Scheduling · Rostering · Mixed Integer Programming · Local Search

---

Work done in collaboration with Vaktaskipan ehf.

E. I. Ásgeirsson  
Reykjavík University  
Tel.: +354-599-6385  
Fax: +354-599-6201  
E-mail: eyjo@ru.is

G. L. Sigurðardóttir  
Reykjavík University

## 1 Introduction

Staff scheduling is a problem that is well known to all companies that have employees working on irregular schedules. Usually, the problem is to determine which employees should cover which shifts so that the demand for manpower is met at every time period and without breaking any regulations or contracts. Additionally, the schedule should be good, i.e. it should meet with the employees' approval and satisfy as many of their requests as possible.

The nurse rostering problem has been studied by personnel managers, operations researchers and computer scientists for over 50 years. Variations of the problem are NP-hard [21,10,29,25]. Because the nurse rostering problem is well known and since it can include many types of constraints and cover a large set of staff scheduling problems, a large part of the research on staff scheduling is focused on nurse rostering and the terms nurse rostering and nurse scheduling have been used over the years to cover several types of personnel scheduling problems [14].

In nurse rostering there are three key approaches: cyclical scheduling, preference scheduling and self scheduling [9]. A cyclical scheduling problem is a scheduling problem in which several sets of schedules are generated that cover a certain period of time i.e. a month or three months. Then the staff is assigned to a schedule that best fits their preferences so that all demands for manpower are met. The schedules are then repeated for each period. Cyclical scheduling is somewhat inflexible and therefore not able to adjust rapidly to changes in the environment [27]. The main advantage of the cyclical scheduling is that the employees know their schedule a long time in advance.

In preference scheduling, the employees list their preferences for the staff manager who then creates schedules, trying to fulfill as many preferences as possible but also makes sure that all demands for manpower and all work restrictions are met. Thus the personnel manager has a great deal of responsibility for the quality of the schedules. The preference scheduling has many advantages, the major ones being its flexibility and its individual tailoring. Preferences of the staff have become a vital feature of any successful scheduling system. Kellogg and Walczak [24] state that any academic nurse rostering model that does not include some opportunity for preference scheduling will probably not be implemented. The major downside to preference scheduling is the time it takes for the personnel manager to create a good schedule.

In self scheduling the employees themselves become responsible for creating the schedule, instead of the staff manager. These schedules are created by each employee signing up for their preferred shifts knowing the minimum and maximum number of staff needed for each shift with the requirement that the resulting schedule must be a feasible one. The biggest advantages of the self scheduling, beside possible time savings, are the potentially greater staff satisfaction, more commitment and reduced staff turnover, since the employees are empowered by making the schedules themselves. However until recently self scheduling has not been a good approach since it was too difficult to execute this method fairly [22,8], the order in which the personnel sign up

did matter, there was a possibility that the system might get manipulated by some personnel, new employers were unfamiliar with the system and might therefore be disadvantaged and some employees might not sign up for any shifts at all. With the advent of the Internet it has become easier to implement self scheduling fairly, putting self scheduling back on the map as a viable approach. However, having the employees involved in the decision process will always bring some risk of game playing, where employees try to manipulate the system for their own gain [4].

A good way to implement self scheduling is by mixing preference scheduling and pure self scheduling. Here the staff signs up for shifts, making a draft that the personnel manager then turns into feasible schedule. The personnel manager makes sure that the demand for manpower is met at every time and that no work regulations are broken. In this approach the personnel is responsible for creating a good preliminary schedule but the final responsibility of creating the schedule lies with the personnel manager, making this approach better and more fair than either pure preference scheduling or pure self scheduling. This is the approach used in this paper.

Due to the multiple and often changing objectives and goals of staff scheduling, the research on staff scheduling has included many different methods [19]. Mathematical programming techniques, such as column generation [5, 23] and branch and price methods [30], have shown good results. The research on staff scheduling has also focused on more flexible metaheuristic approaches such as genetic algorithms [1, 2, 18, 28] and variable neighborhood search [13], with Tabu-Search [12, 17] and Simulated Annealing [11] being particularly successful [20]. Good overviews of staff scheduling are [3, 20, 26].

The paper is structured as follows: Section 2 the problem is defined, in Section 3 the model is introduced while Section 4 contains the result of the model using real data, as well as a comparison to a local search based heuristic from [6]. Finally, Section 5 contains conclusions and suggestions for further work.

## 2 Problem Definition

Staff scheduling problems have a large number of constraints that need to be satisfied. Those constraints can be divided into two groups, hard constraints and soft constraints. Hard constraints must always be satisfied in order to have a feasible schedule. Hard constraints are often a result of physical resource restrictions and legislations. Soft constraints are requirements that are desirable but not obligatory and therefore allowed to be violated if necessary but it will result in a penalty in the model. Soft constraints violations are often used to evaluate the quality of feasible schedules.

## 2.1 Hard constraints

The hard constraints are mainly based on contracts with the employees and union contracts and must therefore be satisfied at all times. Not all the constraints are the same for all employees although usually the main constraints are the same. The constraints that are usually not the same for all employees are the work limit constraints. The labels in parenthesis after each constraint in the following lists refer to the classification introduced in [7]. In the model the following hard constraints are considered:

- **Restrictions on working hours and rest periods from union regulations and employee contracts.** The union regulations about rest periods, maximum lengths of continuous work within a day, maximum number of continuous days worked, minimum length of continuous rest between shifts and other limits have to be met. Employee contracts can include restrictions on when employee can work, for example an employee that will never work nights or weekends. (R1,R4,R5,R7,R8)
- **Vacation request.** Vacations are considered to be a hard constraint to ensure that no employee will be assigned to a shift while on a vacation. (R2)
- **Requests for time off.** Each employee has a right to some time off, how many hours depending on the company and the employee contract. In our settings this needs to be a hard constraint so these requests won't be violated. (R1)
- **Working weekends.** There can be limits on how many weekends employees are allowed to work in each scheduling period. Each employee has to receive at least A out of every B weekends off, where  $A \leq B$ . These are limits like 2 or 3 weekends off out of every 4 consecutive weekends. (R3)
- **Special shifts, training sessions or meetings.** Employees often have work related duties that are not flexible and are often not included in the number of employees on duty. Since training sessions and meetings are not flexible these constraints must be satisfied. (R6,O6)
- **Other limits on shifts or working hours, for example split shifts.** Split shifts are defined as two separate shifts within the same day, where the time between the shifts is less than the minimum resting period between shifts. It can differ between companies whether splits shifts are allowed or not. Splits shifts are only hard constraints when split shifts are not allowed. (R9)

Each company is different when it comes to number of employees, contracts, habits and regulations, therefore the constraints differs from one company to another. Each company wants to be able to quickly generate a high quality schedule that satisfies all hard constraints and as many of the soft constraints as possible.

## 2.2 Soft constraints

Each time a soft constraint is violated the schedule receives a penalty that appears in the objective function. How high the total penalty is depends on which constraints are violated and how often they are violated. The penalties have different weight factors, depending on how serious a violation of the relevant constraint would be. In the model the following soft constraints are considered:

- **Minimum and maximum staff level.** An estimate of the demand for manpower at every time slot over the whole period the schedule is supposed to cover is necessary. This estimate can vary greatly between companies depending on how good their forecast for the demand of manpower is. Some companies use minimum and maximum staff level for every time slot while others give exact number of employees needed for every time slot. We want the on-duty employees in the schedule to be between the minimum and maximum staff level or as close as possible to the exact number of required on-duty employees, otherwise the schedule will be penalized. (C2,C3)
- **Minimum and maximum number of on-duty hours for each employee.** In every employee contract a number of required on-duty hours are given. However since the employees are often working irregular hours, there must be some flexibility in required on-duty hours for each scheduling period. Therefore the required on-duty hours are interpreted as minimum and maximum number of on-duty hours for each employee. Minimum and maximum numbers of on-duty hours for each employee are calculated based on monthly working hours given in the contracts and accumulated deviations from the required on-duty hours from the previous period. (R1)
- **Employee requests for shifts.** The first step in making a schedule is to make each employee signs up for their preferred shifts knowing the minimum and maximum number of staff needed for each shift. This encourages employees to create their own work schedule and makes the schedule more acceptable for the employees. It is therefore important to meet as many requests as possible. (P3,P4)
- **Employees assigned to shifts on weekends before or after their vacations.** If an employee is finishing his vacation on Friday or beginning his vacation on Monday it is unlikely he wants to work the adjacent weekend. So unless otherwise requested we will try to have the adjacent weekend free. (E8)

## 3 The MIP Model

The model contains five sets of binary variables to keep track of the staff allocation. The binary variable  $y_{itk}$  determines if employee  $i \in I$  is working in timeslot  $t \in T$  on day  $k \in K$ , where  $I$  is the set of employees,  $T$  is the set of timeslots within a single day and  $K$  is the set of days in the scheduling period. Each day is partitioned into timeslots, which, in our examples are

usually 30 minutes, although some companies use 15 minutes or whole hours. The employees are assigned to shifts, where a shift is subset of the timeslots. The timeslots within a single shift are usually contiguous, although this is not necessary. The binary variable  $x_{ijk}$  determines whether employee  $i \in I$  is assigned to shift  $j \in J$  on day  $k \in K$ , where  $J$  is the set of allowed shifts. A shift can include timeslots from two consecutive days, so we say that shift  $j$  belongs to day  $k$  if the first timeslot in shift  $j$  is within day  $k$ .

Additionally, we have binary variables  $d_{ik}$  that determine if employee  $i \in I$  is working on day  $k \in K$ . Since some regulations concern weekends, we use binary variables  $\omega_{iw}$  that denote if employee  $i$  is working on the  $w$ -th weekend, where  $1 \leq w \leq |W|$  and  $W = [W_1, W_2, \dots] \subset (K \times K)$  is the set of all weekends, i.e. Saturdays and Sundays, in  $K$ , and  $W_w$  is a set containing the corresponding days for the  $w$ -th weekend.

Every employee contract we've seen so far contains the requirement that the employee must have a specific number of contiguous hours of rest in every 24 hour period. We generate so called rest-shifts to ensure that the employees get their contiguous rest. For each day, we generate all possible rest-shifts starting within that day of length equal to the required rest. Each employee is then assigned to one such rest shift every day. The binary variables  $z_{ilk}$  determine if employee  $i \in I$  is assigned to rest-shift  $l \in L$  on day  $k \in K$ , where  $L$  is the set of all possible contiguous timeslots of the required length, i.e. the rest-shifts.

The model also contains a number of variables to determine the soft constraint violations. These variables are called  $p_\gamma^\alpha$  where  $\alpha$  corresponds to the equation number where the penalty applies and  $\gamma$  denotes the indices over which the penalty variable is defined. The penalties are weighted, the constant  $c_\alpha$  is the weight of penalty  $p_\gamma^\alpha$ . One such penalty is a binary variable, while the other penalties are continuous.

The objective of the mixed integer model is to minimize the total weighted sum of the penalties that correspond to the soft constraint violations.

$$\begin{aligned} \min \quad & c_2 \times \sum_{t \in T} \sum_{k \in K} p_{tk}^2 + c_3 \times \sum_{t \in T} \sum_{k \in K} p_{tk}^3 + c_9 \times \sum_{i \in I} p_i^9 \\ & + c_{10} \times \sum_{i \in I} p_i^{10} + c_{11} \times \sum_{i \in I} p_i^{11} + c_{12} \times \sum_{i \in I} \sum_{k \in K} p_{ik}^{12} \\ & + c_{13} \times \sum_{i \in I} \sum_{t \in T} \sum_{k \in K} p_{itk}^{13} + c_{14} \times \sum_{i \in I} \sum_{k \in K} p_{ik}^{14} \\ & + c_{18} \times \sum_{i \in I} p_i^{18} \end{aligned} \quad (1)$$

$$\text{s.t.} \quad \sum_{i \in I} y_{itk} \geq \text{demand}_{tk}^{min} - p_{tk}^2 \quad \forall t \in T, k \in K \quad (2)$$

$$\sum_{i \in I} y_{itk} \leq \text{demand}_{tk}^{max} + p_{tk}^3 \quad \forall t \in T, k \in K \quad (3)$$



$$y_{itk} = \sum_{j \in \text{shifts}} x_{ijk} \quad \forall i \in I, t \in T, k \in K \quad (4)$$

$$y_{itk} \leq 1 - z_{ilk} \quad \forall i \in I, t \in T, l \in L, k \in K \text{ where } t \in L \quad (5)$$

$$\sum_{l \in L} z_{ilk} = 1 \quad \forall i \in I, k \in K \quad (6)$$

$$y_{itk} = 0 \quad \forall (i, t, k) \in \text{NotAvailable} \quad (7)$$

$$x_{ijk} = 0 \quad \forall (i, j, k) \in \text{NotAvailableShift} \quad (8)$$

$$\sum_{t \in T} \sum_{k \in K} y_{itk} \geq \text{time}_i^{\text{min}} - p_i^9 \quad \forall i \in I \quad (9)$$

$$\sum_{t \in T} \sum_{k \in K} y_{itk} \geq \text{time}_i^{\text{min}} * (1 - p_i^{10}) \quad \forall i \in I \quad (10)$$

$$\sum_{t \in T} \sum_{k \in K} y_{itk} \leq \text{time}_i^{\text{max}} + p_i^{11} \quad \forall i \in I \quad (11)$$

$$\sum_{t \in T} y_{itk} \leq \text{timeperday}_{ik}^{\text{max}} + p_{ik}^{12} \quad \forall i \in I, k \in K \quad (12)$$

$$y_{itk} = 1 - p_{itk}^{13} \quad \forall (i, t, k) \in \text{Requests} \quad (13)$$

$$\sum_{j \in J} x_{ijk} = d_{ik} + p_{ik}^{14} \quad \forall i \in I, k \in K \quad (14)$$

$$x_{ijk} \leq d_{ik} \quad \forall i \in I, j \in J, k \in K \quad (15)$$

$$\sum_{\delta=0}^{d_{\text{max}}} d_{i(k+\delta)} \leq d_{\text{max}} \quad \forall i \in I, k \in \{K : k \leq |K| - d_{\text{max}}\} \quad (16)$$

$$d_{ik} \leq \omega_{iw} \quad \forall i \in I, 1 \leq w \leq |W|, k \in W_w \quad (17)$$

$$\sum_{\delta=0}^{W^{\text{max}}} \omega_{i(w+\delta)} \leq W^{\text{max}} + p_i^{18} \quad \forall i \in I, 1 \leq w \leq |W| - W^{\text{max}} \quad (18)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in I, j \in J, k \in K \quad (19)$$

$$y_{itk} \in \{0, 1\} \quad \forall i \in I, t \in T, k \in K \quad (20)$$

$$z_{ilk} \in \{0, 1\} \quad \forall i \in I, l \in L, k \in K \quad (21)$$

$$d_{ik} \in \{0, 1\} \quad \forall i \in I, k \in K \quad (22)$$

$$w_{i\omega} \in \{0, 1\} \quad \forall i \in I, \omega \in W \quad (23)$$

$$p_i^{10} \in \{0, 1\} \quad \forall i \in I \quad (24)$$

$$p_{tk}^2, p_{tk}^3 \geq 0 \quad \forall t \in T, k \in K \quad (25)$$

$$p_i^9, p_i^{11}, p_i^{18} \geq 0 \quad \forall i \in I \quad (26)$$

$$p_{ik}^{12}, p_{ik}^{14} \geq 0 \quad i \in I, k \in K \quad (27)$$

$$p_{itk}^{13} \geq 0 \quad \forall i \in I, t \in T, k \in K \quad (28)$$

Constraints 2 and 3 handle the number of on-duty employees at all times, demand $_{tk}^{\text{min}}$  and demand $_{tk}^{\text{max}}$  denote the minimum and maximum estimated demand for manpower during timeslot  $t$  in day  $k$ . The penalty variable  $p_{tk}^2$  counts how many employees are needed to achieve the minimum number of

employees in each timeslot, while  $p_{tk}^3$  count how many employees are above the maximum number of required on-duty employees. By summing  $p_{tk}^2$  over all  $t \in T$  and  $k \in K$ , we get the total number of understaffed man-hours that are understaffed, and similarly for the total number of overstaffed man-hours using  $p_{tk}^3$ . We use constraint 4 to connect timeslots to shifts. Constraint 5 ensures that the employees are not working during their mandatory daily rest while constraint 6 ensures that each employees gets a rest period exactly once every day.

The availability of an employee is determined by various factors, such as contracts (e.g. no night shifts), vacations or requests for time off. We use constraints 7 and 8 and the sets `NotAvailable` and `NotAvailableShift` to limit the availability of employees. Constraints 9 and 11 make sure that the total working hours for each employee is within given limits, while constraint 10 is used to count how many employees are below the minimum required working hours over the planning period. Constraint 12 ensures that the number of working hours within a single day.

Employees sign up for shifts, but we translate those into timeslots, and use constraint 13 to figure out which requested timeslots are fulfilled. This method of fulfilling requests is the same as the one used in [6]. Constraints 14 and 15 are used to determine if an employee is working on a specific day, while the penalty associated with constraint 14 is used to determine if split shifts are allowed or not. Most employee contracts state the maximum number of consecutive days that the employee is allowed to work. Constraint 16 ensures that this is not violated. The constant  $d_{\max}$  is the maximum allowed consecutive working days. Finally, constraint 17 is used to determine if an employee is working on a weekend while constraint 18 handles the maximum number of consecutive working weekends that are allowed, the constant  $W^{\max}$  denotes the maximum number of consecutive working weekends.

Since the scheduling period is not isolated from the day to day running of the company or institution, we need to be careful with the boundary conditions, e.g. if an employee is working on the last shift before the start of the scheduling period, we cannot assign him/her to a shift at the start of the scheduling period. These boundary conditions can be encoded into the sets that we use to determine availability and preprocessed as a fixed assignment in the instances where an employee is working on a shift that straddles the boundary of the scheduling period.

### 3.1 Model limitations

One of the major limitations of the MIP model is the issue of fairness. The model does not include any notion of fairness to employees, so we might get a solution where no requests are granted for one employee while other employees have all their wishes fulfilled. We could improve the fairness, e.g. by adding a bound on the fraction of fulfilled requests, but that would also require additional penalties and weights.

Other limitation is that, if the workforce doesn't match the requirements so that the company or institution is forced to have understaffing or overstaffing, our partners would prefer if this is spread somewhat equally, instead of having a massive under- or overstaffing on a single shift and no problems at other times. Our model does not include any mechanism for leveling out any potential deviations. However, by adjusting the required manpower based on the workforce, this leveling could be achieved.

For some of the cases that we tried, the running time it took the solver to find an optimal solution was not within reasonable limits for the optimal solution. Instead of finding the optimal solution, we settle for a near optimal solution, i.e. a solution that is provably within some fraction of the optimal solution. Using our test cases, we found a solution that was within 1% of the optimal solution in a reasonable timeframe. By increasing the relative MIP gap, i.e. the difference between our solution and the bound on the optimal solution, we can speed up the solution time, but at the cost of a higher objective value. However, since a large part of the input, such as the manpower estimates, is often only based on a best guess it is debatable whether finding the optimal solution is actually worthwhile, especially if the time required is orders of magnitude larger than the time it takes to find a solution within 1%-5% of the optimum.

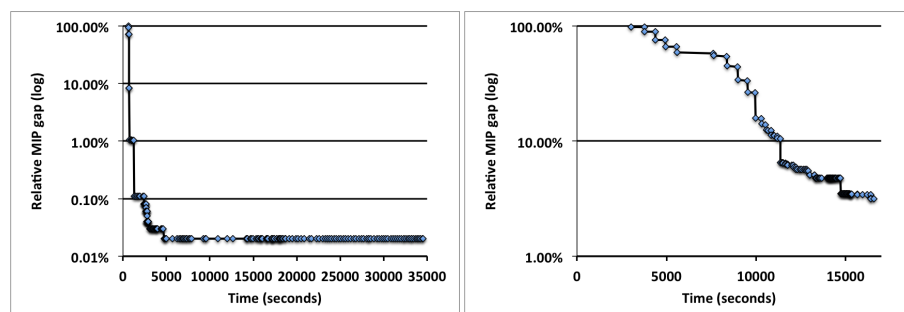
## 4 Experimental Results

To evaluate the performance of our algorithm, we use actual data from four companies and institutions. These companies and institutions include a nursing home, call centers and an airport service company. We will present the details of each problem instance and show examples of the preliminary schedule and the near-optimal solution. To get a better feeling for the complexity of the problem and the quality of the solutions, we also compare the near-optimal MIP solution to the solution of the local search algorithm introduced in [6]. The local search algorithm is designed to emulate the behavior of a typical staff manager. The local search algorithm iterates through multiple stages, with different objectives and different neighborhoods for all the stages, each of which is designed to emulate a specific action taken by the staff manager. For our examples, the scheduling period is usually 6 weeks, but we will plot the preliminary schedule and the improved schedule for only a single week for each problem instance. We tried to select a typical week for each instance. The data for the problem instances is available online [31]. Most of the instances include employees that have predefined or fixed schedules. For the purpose of the the MIP problem, we preprocess these employees, so they are not included in the MIP problem, and adjust all parameters such as staffing levels accordingly.

Figure 1 shows a log-scale of the relative MIP gap as a function of running time for two of the examples that we have. The problems were all solved using Gurobi 4.6.1 on a laptop with a 2.5 GHz Intel Core 2 Duo processor and 4 GB of memory, running Mac OS X 10.6. As can be seen from Figure 1, the

**Table 1** Default penalty weights for soft constraint violations.

$c_2$	$c_3$	$c_9$	$c_{10}$	$c_{11}$	$c_{12}$	$c_{13}$	$c_{14}$	$c_{18}$
15	2	10	100	10	10000	1	10000	10000

**Fig. 1** The logarithm of the relative MIP gap for two instances as a function of the running time. These instances are of the two call centers from our datasets.

solver manages to find a solution within 1% of optimum after 21 minutes in the first instance. After running the solver for over 8.5 hours for the first instance, the solver still had not found an optimal solution. The second instance is significantly harder, there we were only below 10% after around 3 hours and around 3% after more than 4.5 hours. The easiest problem in our problem sets was however solved to optimality within 2 minutes. Since these schedules are usually created for periods of 4-6 weeks, a few hours of computation are usually acceptable. However, the difference between the two graphs in Figure 1 emphasizes the unreliability of the MIP approach with regards to the running time. When presented with a new instance, it can be difficult to guess if the solution will be ready in minutes or if we will have to wait a few hours until we have a good solution.

The default penalty weights that we used are shown in Table 1. The weights for maximum hours per day for an employee, more than one shift per day and consecutive working weekends is set very high, so these constraints are effectively hard constraints. Our partners prefer overstaffing to understaffing, so overstaffing has a penalty of 2 while understaffing carries a penalty of 15 for each timeslot. To minimize the number of people below their minimum duty hours, the corresponding penalty weight of 100 is relatively high, while the penalty for each timeslot over or under the duty hour limits carries a penalty of 10. Finally, the penalty for each unfulfilled requested timeslot carries a penalty of 1. By adjusting these penalties, we can tailor our model to different company cultures, e.g. put more emphasis on satisfying employee requests by increasing the corresponding penalty.

Using the notation introduced by De Causmaecker [15,16], we can describe the following problem instances as (AS|TVNO|PLGO).

**Table 2** Results for the nursing home instance.

	Preliminary	Local Search	<b>MIP</b>
Scheduled hours	4669	5198	<b>4774</b>
Man-hours overstaffed	478	220	<b>0</b>
Man-hours understaffed	777	21	<b>109</b>
Employees below minimum duty hours	3	0	<b>0</b>
Requested hours granted		0.97	<b>0.98</b>

#### 4.1 Problem instance: Nursing home

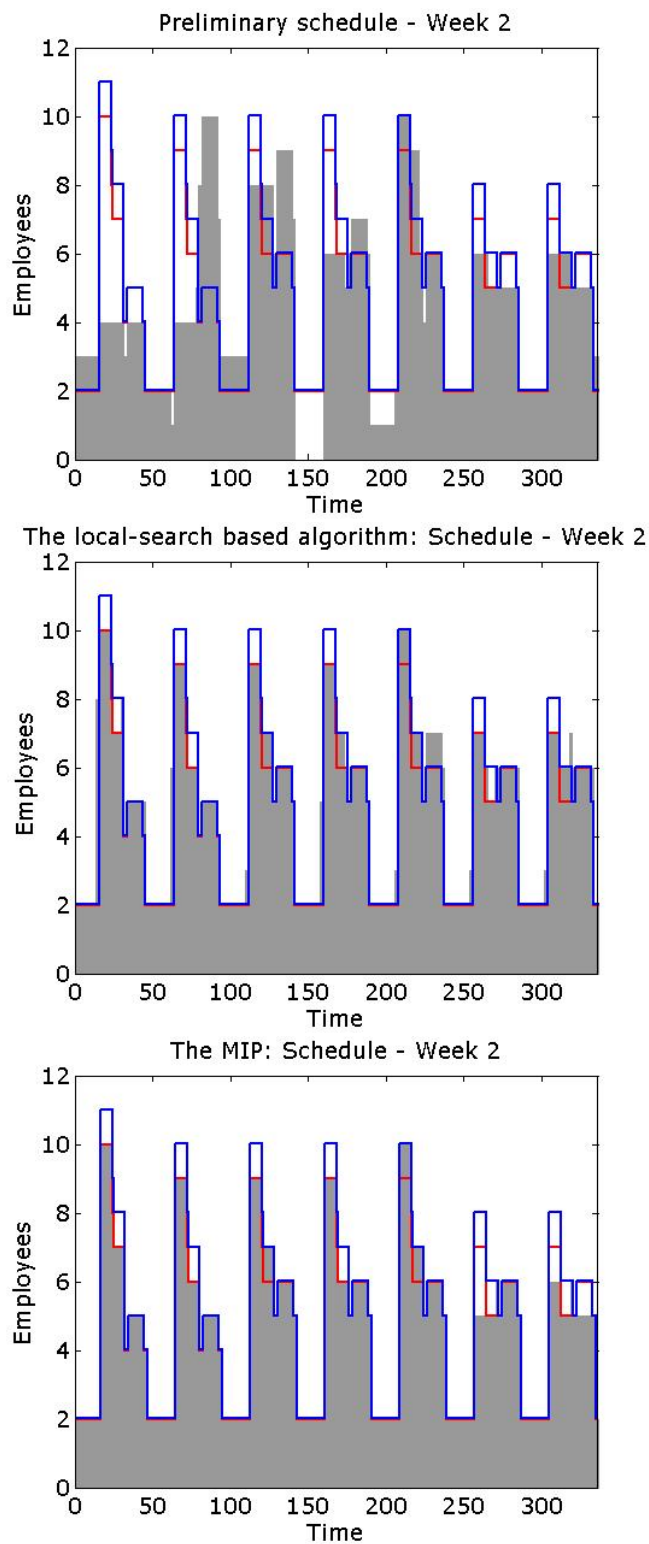
The first problem instance comes from a nursing home with 55 employees. The scheduling period is 6 weeks with 30 minute intervals. Each day contains of 18 shifts. The length of each shift ranges from 4 hours up to 12 hours. After preprocessing 5 employees with fixed schedules, we're left with 50 employees. If we sum up the total maximum working hours over the scheduling period, we get that the maximum number of hours that we can assign, without any overstaffing, is 5217 hours. However, the total duty hours of the employees is 5333 hours, so unless we violate the overstaffing constraint, we can never satisfy all duty hour requirements for the employees. Table 2 shows the soft constraint violations for the preliminary schedule, the local search algorithm from [6] and the near-optimal MIP solution.

The nursing home has the following hard constraints. An employee cannot work on more than 6 consecutive days, the maximum length of a shift is 9 hours while the minimum length of a shift is 4 hours. In any 24 hour period, each employee must get at least 8 consecutive hours of rest, while the maximum number of working hours in any 24 hour period is 9 hours. Initially, the problem had 1.204.482 rows, 351.032 columns and 3.378.221 nonzeros. After presolve the problem had 50.116 rows, 57.409 columns and 591.422 nonzeros. Gurobi managed to solve the problem to optimality within 2 minutes.

Figure 2 shows the preliminary schedule, the local search solution and the MIP solution for a typical week in the scheduling period. We see that the MIP solution has almost the same fraction of satisfied requests as the local search solution and the plan fits better into the manpower limits, although we're still left with a slight understaffing problem. All employees are within their duty-hours limits. We can see the slight understaffing in the MIP solution in Figure 2 over the last two days, i.e. the solution is a couple of employees short during the morning shift on the weekend.

#### 4.2 Problem instance: Call center A

The second problem instance is the first of the two call centers in our datasets. Call center A has 92 employees and the scheduling period is 6 weeks in 30 minute intervals. After preprocessing, i.e. removing employees that have fixed



**Fig. 2** Staffing levels for the nursing home problem instance. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

**Table 3** Results for the call center A problem instance.

	Preliminary	Local Search	MIP
Scheduled hours	9424	11920	<b>10578</b>
Man-hours overstaffed	390	791	<b>38</b>
Man-hours understaffed	1560	14	<b>77</b>
Employees below minimum duty hours	19	5	<b>8</b>
Requested hours granted		0.96	<b>0.79</b>

schedules, the number of employees drops down to 74. Each day consists of 97 shifts, whose lengths vary from 4 hours up to 11 hours. The total maximum required on-duty employees is 11582, while the total duty hours for all employees is 12054, so it will be impossible to satisfy both the duty hour constraints and the overstaffing constraints. The maximum number of consecutive working days is 6, the maximum number of working hours in each 24 hour period is 9 hours. In any 24 hour period, each employee must get at least 11 consecutive hours of rest. Table 3 shows the data from the preliminary schedule, the results of the local search and the results of the near-optimal MIP solution.

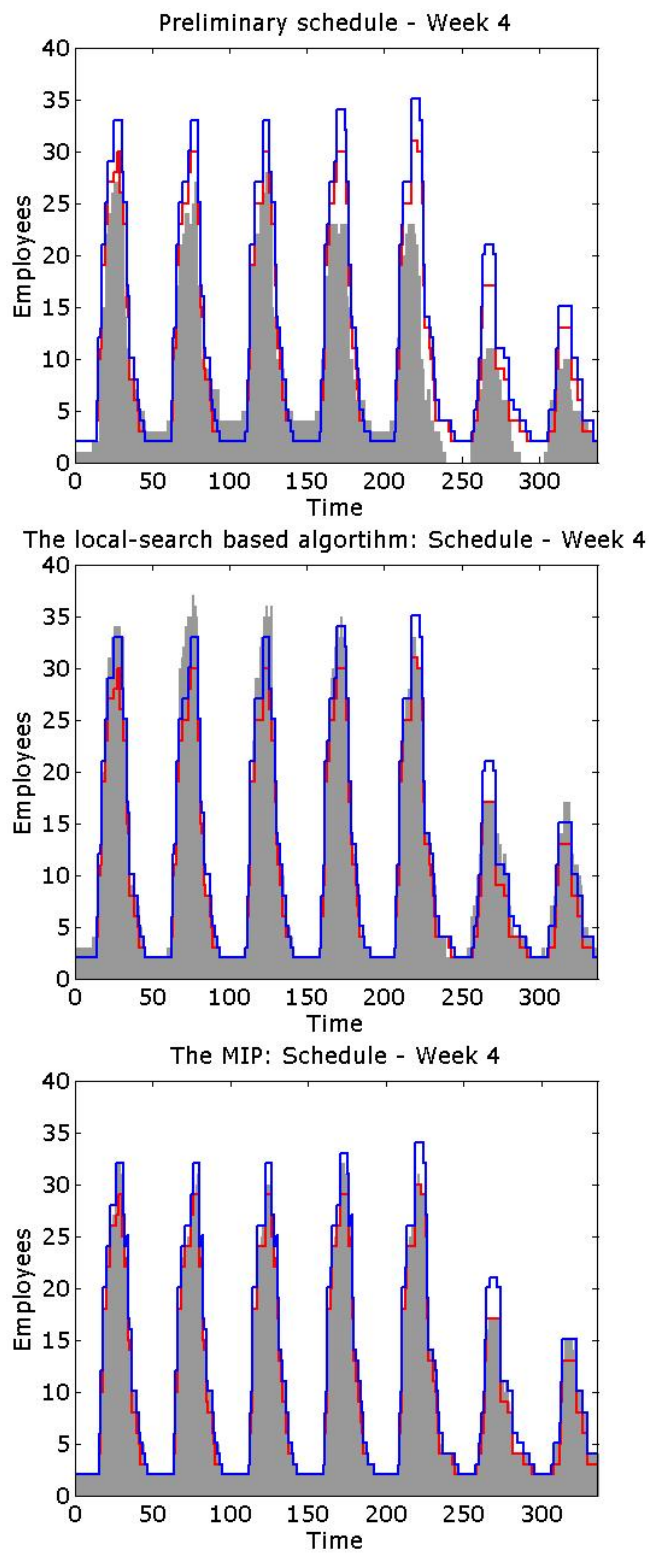
The MIP problem for call center A had 2.419.592 rows, 763.124 columns and 9.794.776 nonzeros. After presolve we were left with 118.445 rows, 267.630 columns and 3.277.415 nonzeros. Gurobi managed to solve the problem to within 1% of optimality within 22 minutes. The solution we show here is within 0.2% of optimum, after 8.5 hours of computations.

We see from Table 3 that the near-optimal MIP solution manages to satisfy the manpower requirements extremely well, with just a few hours of over/understaffing. However, this comes at the cost of satisfying employee requests, which is down to 79%. We could improve the request ratio by modifying the manpower requirements or decreasing the penalty of overstaffing. However, for these instances, we would recommend that the unfulfilled requests would be better handled manually, i.e. by allowing the staff manager to decide whether to accept a request or to have the number of on-duty staff within limits. The solution to the MIP problem along with a list of unsatisfied requests would make such a task relatively easy.

Figure 3 shows a typical week for call center A. We see that the preliminary schedule has problem with both under- and overstaffing. The local search puts emphasis on satisfying employee requests at the expense of overstaffing, while the near-optimal MIP solution focuses on the staffing levels while sacrificing a larger share of the employee requests.

#### 4.3 Problem instance: Call center B

Call center B does the planning for only 4 weeks in advance, specifying exactly how many should be on duty in every 15 minute interval. The total number of employees at call center B is 62, which, after preprocessing drops down to 46.



**Fig. 3** Staffing levels for call center A. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.



**Table 4** Results for the call center B problem instance.

	Preliminary	Local Search	MIP
Scheduled hours	6623	7554	<b>6071</b>
Man-hours overstaffed	795	609	<b>234</b>
Man-hours understaffed	1306	189	<b>37</b>
Employees below minimum duty hours	15	7	<b>17</b>
Requested hours granted		0.86	<b>0.65</b>

The call center is overstaffed, the total available man-hours for the scheduling period is 8134 hours while the total required man-hours over the same period is 7134 hours. There are 115 possible shifts for each day. The hard constraints that must be satisfied for call center B are that employees cannot be working on more than 6 consecutive days, in every 24 hour period there must be at least 11 consecutive hours of rest and at most 11 hours of work. The maximum length of a shift is 11 hours while the length of a shift must be at least 4 hours.

The MIP problem consists of 5.110.509 rows, 585.322 columns and 10.399.996 nonzeros. After presolve, the problem has 126.839 rows, 218.628 columns and 4.598.686 nonzeros. Call center B was by far the most challenging instance that we tried. The Gurobi solver didn't find a feasible solution until after 50 minutes and we had to wait for more than 3 hours before the value of solution was within 10% of optimum.

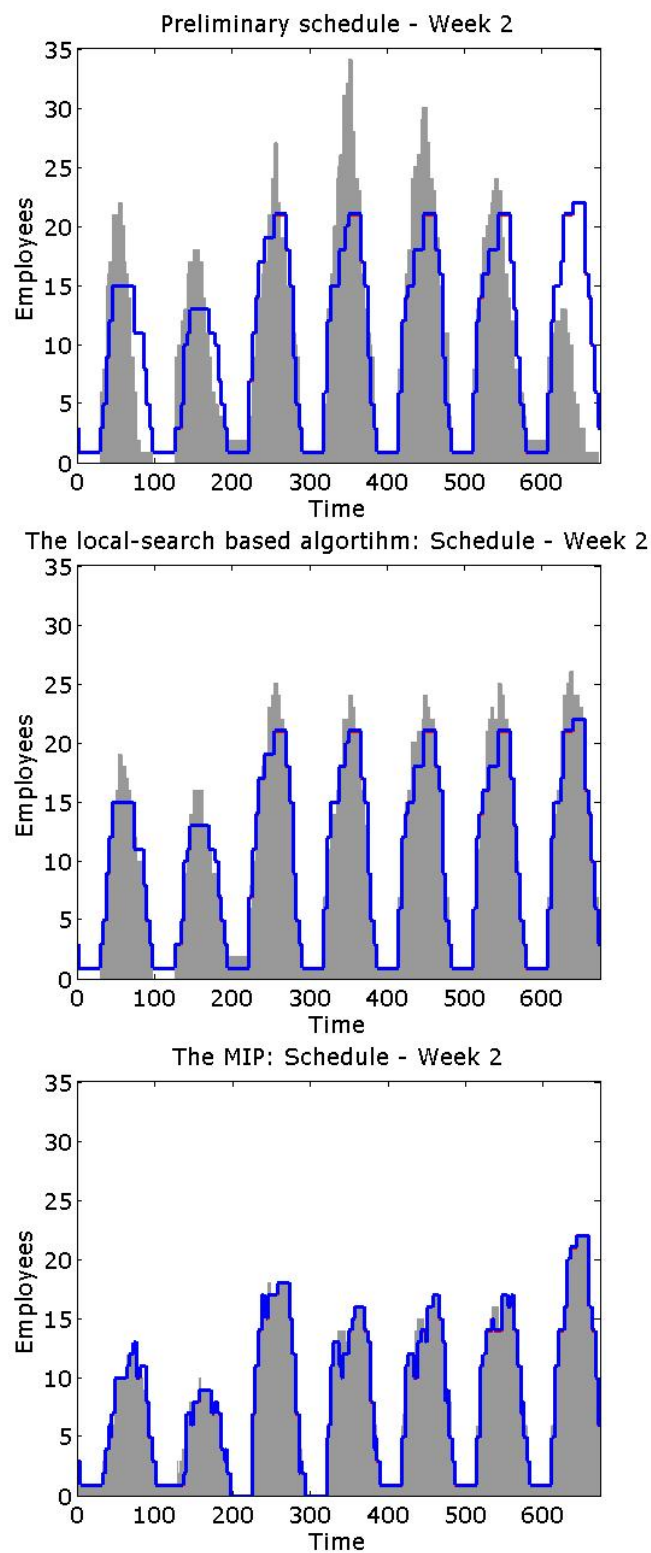
Table 4 shows the results for the local search procedure, the near optimal MIP solution as well as the preliminary schedule.

Figure 4 shows the staffing levels for a typical week. The three figures contain the preliminary schedule, the local search solution and the solution of the MIP problem. The required staffing level for the MIP solution has a different shape than the other two due to the fixed assignments that were preprocessed in the MIP solution, but kept as a part of the input in the preliminary schedule and the local search solution.

Since there is no flexibility in the staffing level limits, both algorithms have some problems with both under- and overstaffing. As seen in Figure 4 and Table 4, the near-optimal MIP solution manages to stay close to the required staffing levels, but at the expense of employee requests. Since the call center is overstaffed, it is not surprising that both solutions have some employees that are below the minimum required duty hours as well as overstaffing problems.

#### 4.4 Problem instance: Airport ground service.

The fourth problem instance is an airport ground service company. The scheduling period is six weeks in 30 minute intervals. The demand for on-duty employees depends on the flight schedules at the airport. In this particular instance, there are many flights that leave during the early morning, and then there is another concentration of flights in the afternoon. Since there are almost no



**Fig. 4** Staffing levels for call center B. The gray area denotes the number of employees on duty while the solid line denotes the exact number of employees that are required to be on duty at each time.

**Table 5** Results for the airport ground services problem instance.

	Preliminary	Local Search	<b>MIP</b>
Scheduled hours	3997	6670	<b>6608</b>
Man-hours overstaffed	192	641	<b>50</b>
Man-hours understaffed	2464	517	<b>11</b>
Employees below minimum duty hours	23	0	<b>0</b>
Requested hours granted		0.94	<b>0.89</b>

flights scheduled at any time apart from the morning and afternoon busy periods, the requirements for employees peaks during the two busy periods but drops sharply during other times. Each day contains 53 different shifts and the airport ground service has 53 employees, one of which has a fixed schedule. Due to the structure of the manpower requirements, the employees often work a short morning shift and then another short afternoon shift with a few hour break in-between. This means that split shifts are allowed so we change the weight  $c_{14}$  to zero. This problem instance is understaffed compared to the previous examples, here the total maximum required man-hours is 8152 hours over the scheduling period while the available man-hours is only 6350. Table 5 shows the preliminary schedule, the results of the local search method and the near-optimal MIP solution.

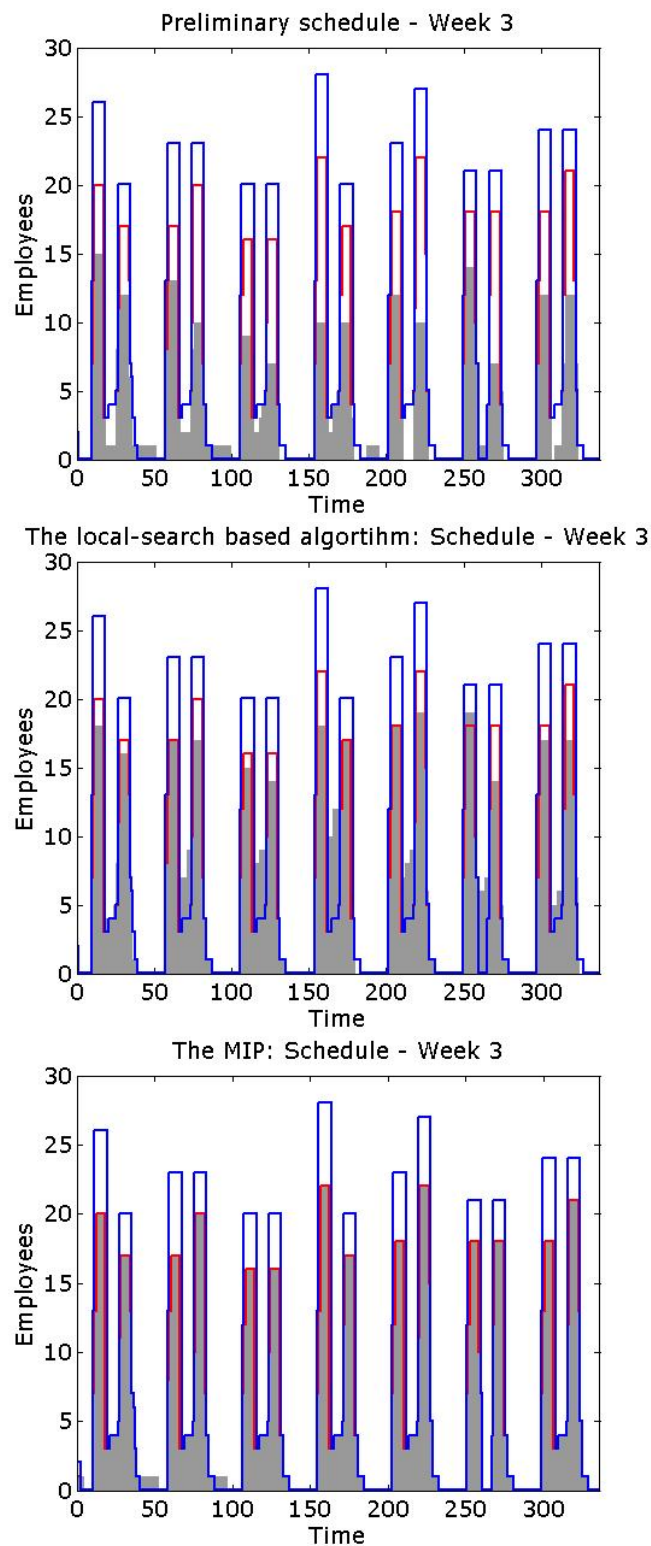
The hard constraints for the airport ground service are that there must be a minimum continuous rest of 11 hours in any 24 hour period, each employee can not work more than 5 consecutive days, employees cannot work more than 12 consecutive hours while the number of working hours in any 24 hour period is also 12 hours.

The MIP problem consists of 1.716.477 rows, 441.352 columns and 5.342.624 nonzeros. After presolve, the problem has 113.047 rows, 170.851 columns and 2.132.307 nonzeros. It took Gurobi around 25 minutes until the value of solution was within 1% of optimum.

As we can see in Figure 5, the near-optimal MIP solution manages to fit the staffing levels almost perfectly, while still satisfying the majority of the employees requests. For comparison, the local search method has a slightly higher ratio of fulfilled requests, but both the over- and understaffing is at least an order of magnitude larger than for the MIP solution.

## 5 Conclusions

In this paper we have introduced a mixed-integer programming (MIP) formulation to create a high quality feasible staff schedule from a partial staff schedule based on requests from employees. The results from the four different companies and institutions show that it is possible to find high quality schedules in a reasonable amount of time by using mixed integer programming. The data for these problem instances is available online [31]. Our MIP model al-



**Fig. 5** Staffing levels for the airport ground services. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

lows flexibility in terms of shifts lengths and shifts starting times, as well as handling all the constraints and requirements of our real-life instances. Due to time constraints, we only solved one of the four instances to optimality. The other three were solved until we had a solution that was within 1% of optimum for the easier problems and within 10% for the most difficult problem. The longest time it took for such a solution was around 3-4 hours, which, is acceptable in these cases, since these scheduling problems are only solved every 4-6 weeks.

We also compared the near-optimal solution to a solution from a local search based method [6]. The results show that the MIP solution does an extremely good job of keeping the number of on-duty personnel within the required limits, at the expense of employees requests, while the local search had more emphasis on satisfying requests. For the easier problems, the local search method does quite well, so that there is not much of a difference between the solutions. However, when the problems became more difficult, with complicated shift structures, and in one case, split shifts, the MIP solver produced solutions that were vastly superior to the local search solutions.

## References

1. Ahmad, J., Yamamoto, M., Ohuchi, A., Evolutionary Algorithms for Nurse Scheduling Problem. Proceedings of CEC00, San Diego, 196–203 (2000).
2. Aickelin, U., Dowsland, K., Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling* 3(3), 139–153 (2000).
3. Alfares, H.K., Survey, categorization and comparison of recent tour scheduling literature. *Annals of Operations Research* 127, 145–175 (2004).
4. Alsheddy, A., Edward, T., Empowerment scheduling for a field workforce. *Journal of Scheduling*, 1–16 (2011).
5. Al-Yakoob, S. M., Sherali, H. D., A column generation approach for an employee scheduling problem with multiple shifts and work locations. *Journal of the Operational Research Society*, 59(1), 34–43 (2008).
6. Ásgeirsson, E. I., Bridging the gap between self schedules and feasible schedules in staff scheduling, *Annals of Operations Research* (2012)
7. Ásgeirsson, E.I., Kyngäs, J., Nurmi, K. and Stølevik, M., A Framework for Implementation-Oriented Staff Scheduling, In Proc of the 5th Multidisciplinary Int. Scheduling Conf.: Theory and Applications (MISTA), Phoenix, USA, (2011).
8. Bailyn, L., Collins, R., Song, Y., Self-scheduling for hospital nurses: an attempt and its difficulties, *Journal of Nursing Management*, 15(1),72–77 (2007).
9. Bard, J. F., Purnomo, H. W., Preference Scheduling for Nurses Using Column Generation. *European Journal of Operational Research*, 164, 510–534 (2005).
10. Bartholdi, J.J., A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering. *Operations Research* 29, 501–510 (1981).
11. Brusco, M. J., Jacobs, L. W., Cost analysis of alternative formulations for personnel scheduling in continuously operating organisations. *European Journal of Operational Research*, 86, 249–261 (1995).
12. Burke, E. K., De Causmaecker, P., Vanden Berghe, G. (1999). A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem. SEAL'98, LNCS 1585, 187–194.
13. Burke, E. K., De Causmaecker, P., Petrovic, S., Vanden Berghe, G., Variable neighborhood search for nurse rostering problems. *Metaheuristics: computer decision-making*, 153–172 (2004).
14. Burke, E. K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H., The State of the Art of Nurse Rostering. *Journal of Scheduling*, 7, 441–499 (2004).

15. De Causmaecker, P., Vanden Berghe, G. (2011). Towards a reference model for timetabling and rostering. *Annals of Operations Research*.
16. De Causmaecker, P., Vanden Berghe, G., A categorisation of nurse rostering problems. *Journal of Scheduling*, 14, 3–16 (2011).
17. Dowsland, K., Nurse scheduling with Tabu Search and Strategic Oscillation. *European Journal of Operations Research*, 106, 393–407 (1998).
18. Easton, F., Mansour, N., A Distributed Genetic Algorithm for Employee Staffing and Scheduling Problems. *Conference on Genetic Algorithms, San Mateo*, 360–367 (1993).
19. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D., An Annotated Bibliography of Personnel Scheduling and Rostering. *Annals of Operations Research*, 127, 21–144 (2004).
20. Ernst, A. T., Jiang, H., Krishnamoorthy, M., Sier, D., Timetabling and Rostering. *European Journal of Operational Research*, 153(1), 3–27 (2004).
21. Garey, M.R., Johnson, D.S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, Freeman (1979).
22. Hung, R., Improving Productivity and Quality through Workforce Scheduling. *Industrial Management*, 34(6) (1992).
23. Jeroen, B., Demeulemeester, E., On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. *Annals of Operations Research*, 155(1), 143–166 (2007).
24. Kellogg D.L., Walczak S., Nurse Scheduling: From Academia to Implementation or Not. *Interfaces* 37(4), 355–369 (2007).
25. Lau, H. C., On the Complexity of Manpower Shift Scheduling. *Computers and Operations Research* 23(1), 93-102 (1996).
26. Meisels, A., Schaerf, A., Modelling and solving employee timetabling problems. *Annals of Mathematics and Artificial Intelligence* 39, 41–59 (2003).
27. Rowland, H. S., Rowland, B. L., *Nursing administration handbook 4th ed.*, Gaithersburg, Maryland, (1997).
28. Tanomaru, J., Staff Scheduling by a Genetic Algorithm with Heuristic Operators. *Proceedings of CEC95*, 456–461 (1995)..
29. Tien J, Kamiyama A., On Manpower Scheduling Algorithms. *SIAM Rev.* 24 (3), 275–287 (1982).
30. van der Veen E., Veltman B., Rostering from staffing levels: a branch-and-price approach. *Proceedings of the 35th International Conference on Operational Research Applied to Health Services (ORAHS)*, 1–10 (2009).
31. Datasets for experimental results: <http://www.ru.is/kennarar/eyjo/staffscheduling.html>

---

# Application of Particle Swarm Optimization to the British Telecom Workforce Scheduling Problem

Maik Günther · Volker Nissen

Received: date / Accepted: date

**Abstract** When technicians are deployed in field service, they must be allocated to the correct assignment and their routes should also be optimised as a part of that process. Data from a practical case of British Telecom has been used widely in the literature to test many different solution methods. This work discusses the modification of particle swarm optimization (PSO) for this problem and compares the performance of this hybrid approach to competing solution methods. PSO produces better results than the currently best-known solution that was achieved using fast guided local search. Combined with our previous results on sub-daily staff scheduling in logistics this result underlines the potential of PSO to solve complex workforce scheduling problems.

**Keywords** Combinatorial Optimization · Workforce Scheduling · Particle Swarm Optimization · Hybrid Metaheuristics

## 1 Introduction

The combination of route and personnel scheduling arises in many different applications, for example for the assignment of technicians in field service, for the allocation of care workers, transportation companies or security services. Each industry has its own unique characteristics. Patients should be cared for by the same care worker as much as possible and in transportation companies basic routes are usually dictated. Complex planning demands especially arise when technicians are being assigned. Many different qualifications, timeslots for tasks, limited employee availability, individual performance figures and a complex street

---

M. Günther  
Burghart 14a, 86920 Denklingen, Germany  
Tel.: +49-08243-9930844  
E-mail: maik.guenther@gmx.de

V. Nissen  
Ilmenau University of Technology, Faculty of Economic Sciences  
Department of Information Systems in Services  
P.O. Box 100565, D-98684 Ilmenau, Germany  
Tel.: +49-3677-694047  
Fax: +49-3677-694219  
E-mail: volker.nissen@tu-ilmenau.de

network are only some of the constraints which must be taken into account. For these reasons, this work looks into technician assignment at British Telecom (BT). In contrast, for instance, to the France Telecom problem used in the more recent ROADEF-challenge [9], the BT-problem assumes individual workers instead of temporary teams to work on a particular job. This makes the results more easily transferable to other branches of industry with similar characteristics.

For almost two decades, various methods have been tested using BT's data, including simulated annealing (SA), genetic algorithms (GA), constraint logic programming (CLP), local search (LS) fast local search (FLS) and fast guided local search (fast GLS). Because very good results have already been achieved using hybrid versions of particle swarm optimization (PSO) on similar scheduling problems [27, 17], this solution approach will be investigated for use on the British Telecom problem.

The research goals we pursue are twofold. First, we aim for good solutions to a meaningful and complex practical application that is of significant economic value in such diverse industries as logistics, maintenance work, mobile health care and security services. Second, we want to contribute to the comparison of modern metaheuristics on practical problems of realistic size and complexity.

First, section 2 gives a description of the problem space, especially regarding the appropriate representation of the problem for PSO as well as the complexity. Then we discuss related work from the literature in section 3. The PSO approach is outlined in section 4. An experimental assessment of PSO and a comparison with prior solution methods is done in section 5. The paper concludes with a short summary of main results and some avenues for further research.

## 2 Description of the Problem Space

The problem discussed here comes from British Telecom and consists of a planning scenario in which 118 technicians are to handle 250 spatially separated jobs in one day (actual data can be found in [4]). The working time models – i.e., starting and ending times – of the technicians are given for the day to be planned and may not be changed during planning.

A total of 250 jobs exist  $J = \{1, \dots, J\}$ . Each job  $j$  consists of a five-element tuple: job number, map coordinate  $x$ , map coordinate  $y$ , duration and job type. The  $x$ - and  $y$ -coordinates can be used to calculate the travel costs  $ct$  (interpreted as error points of the respective solution) for the paths traveled as follows:

$$ct((x_1, y_1), (x_2, y_2)) = \begin{cases} \frac{\frac{1}{2} * \Delta_x + \Delta_y}{8}, & \text{if } \Delta_x > \Delta_y \\ \frac{\frac{1}{2} * \Delta_y + \Delta_x}{8}, & \text{else} \end{cases} \quad (1)$$

The duration  $d_j$  of job  $j$  can be between 10 and 513 minutes. This value does not represent the actual job time but rather the time required by the average qualified technician  $E = \{1, \dots, E\}$ . The actual job duration time  $rd_j$  is highly dependent on the experience level  $r_e$  of the technician. Formula 2 shows the calculation of that factor. A task can only be carried out by one technician alone, eliminating the possibility of accelerating job time through cooperation. Neither is it allowed to change technicians during the fulfilment of a job.

Job type refers to the time of day when it is to be carried out. These requirements are hard constraints, meaning they must be fulfilled. Three different job types are distinguished:

- Morning: The job must begin before 12:00.



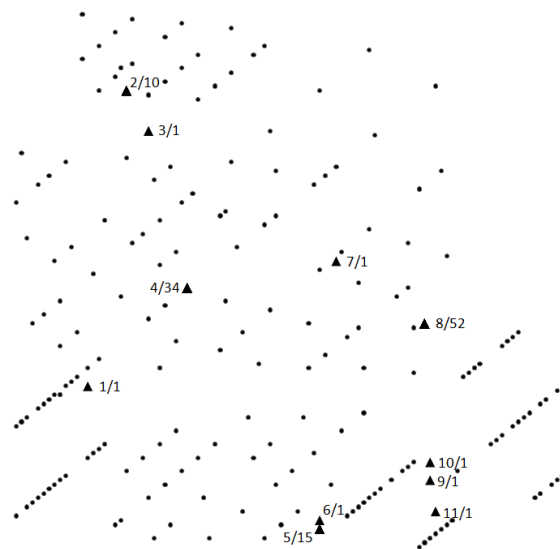
- Afternoon: The job must begin after 12:00.
- No preference: No requirement has been given regarding starting time.

Each of the 118 technicians is deployed by contract for 8 hours. Starting time is either 8:00 or 8:30 with corresponding ending times of 16:00 or 16:30. A technician does not necessarily have to be either traveling or completing a job during the 8-hour shift. It is possible for a technician not to be assigned any jobs during the allotted time. A tuple for a technician consists of five elements: working time start, working time end, experience level  $r_e$  and the  $x$ - and  $y$ -coordinates of his or her service center.

With the help of a technician's experience level  $r_e$  and the average duration  $d_j$  of a job the actual job time  $rd_j$  of job  $j$  can be calculated. For example, assuming a technician with experience level 8 (values below ten signify above average experience) and an average job duration of 20 minutes, the actual completion time  $rd_j$  is 16 minutes.

$$rd_j = d_j * \frac{r_e}{10} \quad (2)$$

Technicians begin their routes at their respective service centers at the start of their working day and must reach the centers again after completing the last job while within their total working time. A total of 11 service centers exist. Figure 1 shows the positions of the jobs and the service centers. The numbers next to the service centers indicate the number of technicians assigned to each of them.



**Fig. 1** Job and service center position [22, 14] (numbers: service center ID/number of assigned technicians)

Table 1 gives the number of technicians  $E_i$  for each service center and the total available capacity of technicians, which is determined using daily working time in minutes.

Qualifications represent hard constraints. Jobs may not be assigned to unqualified technicians. There exists for each job a set of employees who are eligible for that job. The number can vary between a single qualified technician up to 107 technicians.

**Table 1** Service Center Capacity [22, 18]

Service center ID	1, 3, 6, 7, 9, 10, 11	2	4	5	8
Number of technicians $E_i$	1	10	34	15	52
Capacity in min.	480	4.800	16.320	7.200	24.960

Due to the aforementioned constraints and restrictions (job duration, travel times, job type, working times etc.) it is possible for some jobs to remain unfulfilled. As part of the objective function, the volume of incomplete work is to be minimized. For this, an error value  $f$  is added to the average job duration  $d_j$  of job  $j$ , with the value of  $f$  set to 60 for this problem, according to the literature. A binary variable  $s_j$  indicates whether the job will be completed ( $s_j = 0$ ) or not ( $s_j = 1$ ). The error value  $f$  simply increases the weight of the unassigned jobs in the objective function. Also, travel time should be kept as small as possible, since it is not a value-producing activity and utilizes resources. The objective function that minimizes the total number of error points can then be calculated as follows:

$$c = \sum_{e=1}^E ct_e + \sum_{j=1}^J (d_j + f) * s_j \quad (3)$$

## 2.1 Classification of the Problem Space

Ernst et al. [13] provide a comprehensive overview of problems and solution methods for personnel scheduling and rostering from more than 700 analysed sources. Moreover, they classify the different problems they encountered in the literature. According to this classification, the British Telecom problem can be viewed as a problem of “task-based demand” because demand arises from the jobs to be completed. These tasks have an earliest start time, a duration and a latest end time. Also, the employees are assigned to shifts before job allocation is done so that absent employees are known. Therefore, the British Telecom problem also belongs to the group “task assignment”.

In addition to the classification of scheduling problems according to Ernst et al., the present problem can also be classified within the context of the traveling salesman problem. Azarmi and Abdul-Hameed [1] place it in the class of multi-time-constraint traveling salesman problems (multi-TCTSP). Technicians must travel to a series of locations in the shortest order and return to their respective starting positions while adhering to their working time allotment. The problem then becomes a multi-TCTSP because multiple technicians are available and each technician route is a TCTSP. Also, there are restrictions with respect to job time constraints. For this reason, the problem becomes a multi-TCTSPTW (TW = time windows). Furthermore, the addition of qualifications turns the problem into a multi-SDTCTSPTW (SD = Site Dependent). Finally, the existence of multiple service centers brings about the full classification as a multi-MDSDTCTSPTW (MD = Multi-Depot).

## 2.2 Problem Representation

In order to use the various solution methods, the problem must be represented in an appropriate way. For this, Tsang et al. [33] use a permutation of all jobs. The permutation is transformed into an assignment plan using the objective function. However, with this method,

some regions of the solution space are excluded from the start. This means that some shorter paths may not necessarily be found. Therefore, this method is not used in the present work even though it would reduce the number of plausibility checks and correction mechanisms in the utilized solution method.

In the present work, each technician is assigned his or her own permutation of jobs to be completed. The permutation can possibly contain all 250 jobs, which however will not occur in practice. This forms a two-dimensional matrix, in which the rows represent the technicians and the columns stand for series of jobs. Each matrix element contains a job number and the job order in each permutation determines when the job is to be completed, while travel and completion times as well as restrictions on start times (where applicable) are accounted for. Matrix elements without a job receive a uniform dummy value. Each job must be assigned to exactly one technician. If a technician is allocated more jobs than he or she can complete, the objective function marks the excess jobs as incomplete.

### 2.3 Complexity

With respect to assignment planning, Garey and Johnson showed in 1979 [14] that even the simplest forms of staff scheduling are NP complete. Bartholdi [3] proved the NP completeness of personnel scheduling with shift cycles, in which the employees are available with interruptions. In 1982, Tien and Kamiyama [29] showed that practical personnel scheduling problems are more complex than the traveling salesman problem (TSP), which is already NP complete by itself. From an experimental point of view, the works of Easton and Rossin [12] as well as Brusco and Jacobs [5] suggest that general personnel assignment planning problems are difficult to optimize while Cooper and Kingston [8] demonstrated that they are even NP complete. Finally, Kragelund and Kabel [23, 12–15] proved that the general employee timetabling problem is NP hard.

The NP-hard British Telecom problem encompasses 118 employees and 250 jobs, which results in a two-dimensional matrix with 29,500 elements, of which 250 (the actual jobs) do not contain the dummy value. The complexity of the problem space is  $J^S$ , where  $J$  is the total number of jobs and  $S$  the number of jobs for which the average technician is qualified. This yields  $250^{27}$  combinations (approx.  $10^{56}$ ) [34].

## 3 Related Work

Many solution methods have been tested in the past for solving the British Telecom problem. However, the problem space was sometimes modified, such that not all approaches can be compared. This is true, for instance, for the work by Kokkoras et al. [22], [28]. They generate an agent for each service center and solve the sub-problems using CLP from Yang [34]. All solution methods which have been tested on the original version of the British Telecom problem are listed in table 2. The results are also shown in section 5.

The first publication on the British Telecom problem was done by Baker in 1993 [2]. He uses SA, which was used at that time in the British Telecom's Software Work Manager. He represents the problem space as a set of routes, in which each technician is assigned one route, which can also be empty. Jobs which cannot be completed are allocated to a dummy technician. Four different actions are available for the generation of a move.

In the same year, Muller et al. presented their work [25] using distributed working GAs. Each of the GAs can have its own method to transform a chromosome into an assignment

plan. Additionally, they can differ with respect to their behavior. Shared memory exists in which the respective best chromosome is saved. This provides access to chromosomes generated from different approaches.

Two CLP prototypes were presented by Azarmi and Abdul-Hameed in 1995 [1]. The first is tour generation (CLP TG), in which modeling of the problem space is done by assigning each job to a technician and excess jobs are assigned to a dummy employee. The second method involves implemented compact generation (CLP CG), which allows parallel processing. The main difference between this method and CLP TG is that a technician's route is immediately resequenced into an optimum order as soon as a new job is assigned.

A year later, Yang published his CLP approach with and without forward checking [34]. He also tested three variations for ordering during allocation of jobs to technicians.

Tsang and Voudouris utilized LS for the British Telecom problem in 1997 [33]. In order to represent the problem space, they use a permutation of the jobs to be completed (see section 2.2). LS proceeds by exchanging two jobs within the permutation if this improves solution quality. Starting with LS, Tsang and Voudouris integrate FLS and fast GLS together. FLS has the goal of accelerating the optimization method. But this has the consequence that good results could remain out of consideration. There exists for the position of each job in the permutation an activation bit, through which it is determined whether a job remains in consideration or not. For GLS, the objective function is expanded to take into account additional error points due to other rules, so that the search can escape from local optima and can extend into other regions. In summary, fast GLS has yielded the best results up to now.

**Table 2** Solution Methods and Authors for the (original) British Telecom Problem

Method	Author(s)	Year	Source
SA	Baker	1993	[2]
GA	Muller, Magill, Prosser and Smith	1993	[25]
CLP	Azarmi and Abdul-Hameed	1995	[1]
CLP	Yang	1996	[34]
LS	Tsang and Voudouris	1997	[33]
FLS	Tsang and Voudouris	1997	[33]
Fast GLS	Tsang and Voudouris	1997	[33]

Several modifications of the BT problem occur in the literature. Kliem and Anderson [21], for instance, expanded the original problem by investigating correct team formation. Relationships between success of a project and the personalities of the team members were analysed to be able to choose the person with the right personality for a certain project.

Naveh et al. [26] focus on matching the right employee with the appropriate job by considering different individual characteristics. They use constraint programming, which is especially suited to this problem.

During the ROADEF 2007 competition [9] various solution methods were analysed using data sets from France Telecom. These vary between 5 and 150 employees and from 5 and 800 jobs. Qualifications as well as job completion order and priority are taken into account. Additionally, the number of vehicles is limited. In contrast to the British Telecom problem, in which employees always work alone, here teams are created which exist for multiple days. In the ROADEF competition computing time was a limiting factor. Therefore,

constructive methods were used quite often which were highly varied with respect to their functionality, also explaining the standings. At times, the problem space was significantly fragmented in order to reduce complexity.

In 2005 and 2008, Tsang et al. presented an agent-based application based on the RECONET protocol for solving the British Telecom problem using dynamic changes [30], [31]. Experiments were only done on randomly generated problem spaces. Whereas Kokkoras et al. [22], [28] only generated one agent for each service center, agents will additionally be generated for regions with jobs. Also, there is a superior agent (manager), who controls the other agent types.

In 2008, Tsang et al. [32] extended the multi-agent system by adding aspects to support employee self-determination. However, their work only describes an idea – no experiments were carried out.

In the following section a new solution approach based on an adaptation of particle swarm optimization (PSO) is described. The choice of PSO as a metaheuristic approach to solve the original BT problem was motivated by the very good performance of PSO on a roughly similar staff scheduling problem from logistics [27],[17].

## 4 Particle Swarm Optimization for the British Telecom Problem

### 4.1 Particle Swarm Optimization

PSO is a population-based metaheuristic based on the concept of swarm intelligence. It was originally developed by Kennedy and Eberhart [18] for the solution space of the form  $\mathbb{R}^d$ . In PSO there exists a swarm of particles and each particle knows its current position in the solution space (one solution to the problem), its personal best position (pBest) and the best position of its global (gBest) or local (lBest) neighbourhood. The basic PSO procedure is given in algorithm 1.

---

#### Algorithm 1 Basic PSO Procedure

---

```

1: initialize the swarm
2: calculate the fitness of initial particles
3: determine pBest for each particle and gBest (or lBest)
4: repeat
5:   for i = 1 to number of particles do
6:     calculate new position
7:     calculate fitness
8:     new pBest and new gBest (or lBest)?
9:   end for
10: until termination criterion reached

```

---

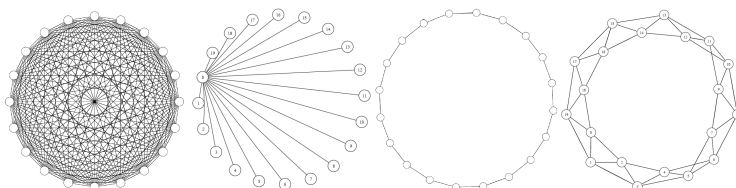
The British Telecom problem is a combinatorial problem space in which integers are used, which means PSO must be modified in an appropriate way. In 2006 Chu et al. [7] adapted PSO for exam scheduling. They changed PSO in such a way that velocity is no longer calculated in order to determine the new position of a particle. Instead, the new position of a particle in each iteration results from the exchange of two allocations in one particle as well as from copying an allocation from pBest or gBest into the new particle position. Brodersen and Schumann [6] build upon this approach and use it for university schedule generation. We have previously expanded this approach by adding probabilities of different

actions and this expanded method has been successfully applied to personnel assignment in logistics [27], [16], [17]. The PSO algorithm modified for the British Telecom problem is based on that previous work and will be explained in detail below. First, however, we discuss methods to avoid premature convergence on a local suboptimum.

In connection with premature convergence, choosing an appropriate neighbourhood topology is important. The topologies used most often in the original form of PSO are the gBest and lBest topologies. In gBest the swarm members are connected in such a way that each particle is a neighbour of every other particle. This means that each particle immediately knows the best global value found up to that point. All particles are included in the position calculation of gBest. If the global optimum is not located near enough to the best particle, it can be difficult for the swarm to search other parts of the solution space, possibly converging instead to a local optimum [24].

Avoiding such convergence to a sub-optimum is one of the goals of the lBest topology, in which a particle is only connected to its immediate neighbours. The parameter  $k$  represents the number of neighbours of a particle. With  $k=2$  the topology is a circle (or ring). Increasing  $k$  to particle count minus 1 yields a gBest topology. In an lBest topology, each particle only possesses information about itself and its neighbours. The swarm converges slower than with gBest but also has a higher chance of finding the global optimum [19].

Another neighbourhood form is the wheel. There exists a central particle which is connected to all other particles. These particles only have that central particle as its neighbour and are isolated from all others. This arrangement prevents a new best solution from being immediately distributed throughout the swarm.



**Fig. 2** Neighbourhood topologies gBest, wheel, circle ( $k=2$ ) and lBest ( $k=4$ ) [24].

Results on neighbourhood topologies and premature convergence are ambiguous in the literature, as is further discussed in [16]. Therefore, in the experimental section, all topologies outlined here are tested and compared.

Another option for preventing premature convergence for the British Telecom problem is to outfit each particle with the capability of looking ahead to its new position (forward checking) and to decide whether that position is potentially worthwhile. This is sensible because the particles in the British Telecom problem could possibly penetrate into regions that would lead to increasing deterioration of the solution. After several iterations the particles will have been changed so much that they can no longer escape from such a local optimum. The particles no longer communicate just with each other. They can also “see”. There is still a small probability for the particles to accept worsening solutions. This prevents the particle from becoming trapped in a local optimum, unable to move in the solution space.

Our version of PSO that was adapted for the British Telecom problem is shown in algorithm 2. In particular, the calculation of the new particle position has been modified compared to standard PSO. Velocity is no longer required. This means that the constriction factor and the inertia weight can be omitted. The same applies to dimension overrun. One may ar-

gue that the resulting method should no longer be called PSO. However, inertia weight and constriction factor were also not present in the original version of standard PSO, but later added since this helped to improve results. Moreover, in our view, the basic properties of PSO are swarm intelligence and a combination of individual and social behavior. These remain intact in our combinatorial variant of PSO.

The new particle position is calculated in line 7 of algorithm 2. The calculation occurs within a loop with the index  $w$ . This is necessary because the two-dimensional matrix of a particle cannot be systematically processed in our approach to determine the new particle position, yet several changes are to be carried out in each iteration. Prior tests were used to heuristically determine a value for  $w$  of 300. In practice, however, 300 changes are never applied to a particle in one iteration. Some changes, for instance, cannot be carried out because of missing qualifications. Moreover, it might occur during copying from pBest or gBest that no actual change is made to the particle position because the copied element is already located at the specified position. Some changes may be rejected by the particle due to an unacceptable deterioration of the solution.

---

#### Algorithm 2 Modified PSO

---

```

1: initialize the swarm using constructive heuristic of algorithm 1
2: calculate the fitness of the particle
3: determine pBest and gBest (or lBest)
4: repeat
5:   for  $i = 1$  to number of particles do
6:     for  $w = 1$  to 300 do
7:       calculate the new particle position with the help of 6 actions
8:     end for
9:     repair the particle using repair heuristic
10:    calculate fitness
11:    new pBest and new gBest (or lBest)?
12:   end for
13: until termination criterion reached

```

---

There are now 6 actions used to determine the new position. The probability of occurrence was heuristically determined using prior tests. The actions are:

1. 0.05%: Exchange two job assignments (without rejection): Two jobs are randomly chosen and the currently assigned technicians are identified. Then, the technicians exchange job assignments, with the new job placed at the same spot in the respective technician's permutation as the old job. Qualifications are taken into account during this action by possibly repeating the choice. This step may not be rejected by the particle even if it leads to worse fitness of the solution.
2. 24.95%: Exchange two job assignments (with possible rejection): The procedure is analogous to the above action, the difference being that the particle does not carry it out if fitness would worsen.
3. 0.25%: Move (without rejection): One technician is randomly chosen and the last job in his permutation is moved to the end of the permutation of another qualified technician, if available. If no other qualified technician can be found, nothing occurs. The particle may not reject the action even if fitness suffers.
4. 14.75%: Move (with possible rejection): The process is analogous to action 3, the difference being that the particle may reject the action if it would worsen fitness.

5. 20%: Insert a value from pBest: Choose a random technician from pBest and a random job within that technician's permutation. Insert that job into the new particle position at the same location as in pBest. If another job is already located at that position, postpone the rest of the jobs by one slot in order to make room. If the job is isolated within the permutation, it is shifted until it borders an occupied slot. This action may be rejected by the particle if it deteriorates fitness.
6. 40%: Insert a value from gBest: This action is analogous to action 5 but gBest (or lBest in other topologies) is used instead of pBest.

Qualifications are especially critical in the British Telecom problem and compliance represents a hard constraint. Therefore, a solution is only valid without qualification errors. In order to remove qualification errors that do occasionally arise, they are repaired using a heuristic approach. This repair heuristic searches for violations and assigns an incorrectly allocated job to a randomly determined qualified employee, where the job is inserted in an appropriate time slot. If a gap has arisen in the job sequence for the unqualified technician it is closed by shifting jobs so that the solution becomes valid.

#### 4.2 Initialization of PSO

An initial solution for PSO is created by applying a constructive method. More specifically, a solution is constructed that respects the BT problem's hard constraints, such as availability of technicians and required qualifications. In addition, the allocation of assignments to technicians is based on capacity still available and the distance of a job from the service center. The exact distances from one job to the next job cannot be applied to the calculation of the remaining capacity because the optimum job order is not yet known. Capacity is therefore determined approximately using the regional allocation of jobs to service centers. The initialization procedure is shown in algorithm 3.

---

#### Algorithm 3 Initialization

---

```

1: set the capacity  $Cap_e$  of technician  $e$  to 0
2: repeat
3:   choose a random job  $j$ 
4:   write all technicians qualified for job  $j$  in list Quali
5:   calculate the travel time  $ct_e$  for each technician from his or her service
   center  $i$  to job  $j$ 
6:   while Quali is not empty do
7:     choose the technician  $e$  with the shortest distance  $ct_e$  to job  $j$ 
8:     remove technician  $e$  from Quali
9:     calculate the completion time  $rd_j$  by technician  $e$ 
10:    if  $Cap_e + rd_j < 8$  hours then
11:      add job  $j$  to the end of the permutation for technician  $e$ 
12:      remove all technicians from Quali
13:      update  $Cap_e$ 
14:    end if
15:  end while
16:  if job cannot be assigned then
17:    choose a technician  $e$  who is qualified for job  $j$ 
18:    add job  $j$  to the end of the permutation for technician  $e$ 
19:    update  $Cap_e$ 
20:  end if
21: until all jobs have been assigned

```

---



## 5 Results and Discussion

The choice of an appropriate termination criterion for a solution method strongly influences result quality and required CPU time. The experiments discussed here uniformly used the number of objective function evaluations (set to 20 million according to pre-tests) as the termination criterion. This offers excellent comparability of the solution methods. Moreover, it will allow for fair comparisons with results of others in the future, as CPU-time is less meaningful in the light of ever increasing computing power. All results using PSO are based on 30 independent runs. They were performed on a PC with an Intel Core Quad 4x1.66 GHz with 4 GB of RAM. When comparing our results to the literature, we will focus on solution quality not speed. CPU-time would be hard to compare, as hardware is always very different. In some cases, CPU-requirements were not even published. More importantly, computing time is not a significant limiting factor in the present problem. It can be assumed that authors who previously worked on the BT-problem terminated their runs when no further improvement appeared possible in reasonable time.

Interpretation of competing solution methods from the literature as introduced in section 3 will only be done briefly here. The respective publications can be referred to for more comprehensive information. Only the minimum value (best solution found) is known for these methods, which reduces comparability. An indication of mean values and the number of replications could not be found. For GA, GA + R (R=repair heuristic) and SA, CPU time is not available. In general, it can be noted that agents and CLP are significantly faster than metaheuristic methods, such as GA, SA and PSO. Table 3 shows the results of PSO with different neighbourhood topologies as well as the methods listed in section 3.

The first set of results was published for SA. Here, a value of 21,050 error points was achieved. In comparison, the results of distributed GA with repair (22,570) and without repair (23,790) are significantly worse. The repair heuristic, however, does improve results for the GA. Azarmi and Abdul-Hameed tested the two CLP variants CLP-CG + R (21,292) and CLP-TG + R (22,241), both using a repair heuristic. Using these solution approaches, more jobs were able to be assigned than in GA and GA + R because the repair in the former method focuses on reducing the number of incomplete jobs. For GA and GA + R on the other hand, the focus of the repair is on the reduction of total error points.

Yang tested various CLP approaches, the best variant of which is shown in table 3 – the variant with forward-checking (FC) and the selection heuristic same direction first (SD) for technician order (CLP SD + FC). There, 20,981 error points were achieved, which was the best value up to that point. Using significantly more CPU time, Tsang and Voudouris were able to produce even better results. They implemented LS, FLS and fast GLS, which all produced better results than previous solution methods. Fast GLS yielded the best error point value of 20,433 which essentially marked the final achievement on this practical problem so far.

Using PSO, a new best solution with an objective function value of 20,193 was found. Comparing different swarm sizes, it became evident that a gBest topology with a swarm of 10 particles performs significantly better than a larger swarm with 200 particles. This result is consistent with our previous results with a similar modified PSO algorithm for a logistics problem [27], [16] [17]. The swarm requires a high number of iterations to get from the initial solution to a very good solution. Using the number of fitness calculations as termination criterion means that significantly more iterations can be performed on a small swarm size than on a large one. The advantage of a large swarm that more knowledge is available on the solution space in each iteration is apparently less important.

**Table 3** Results for the original BT problem (BT\_Mod-250-118). Results that improve the previous best known solution (generated by FLS) are bold. New best solution is set bold and underlined. If solutions are repaired w.r.t hard constraints this is indicated with +R. If a solution method uses forward checking of potential new solutions this is indicated with FC. Results for PSO are based on 30 independent runs each.

Method	Error			Travel costs	Error open jobs	Number open jobs	CPU time in sec.
	∅	minimum	std. dev.				
SA [2]	-	21,050	-	4,390.0	16,660.0	56.0	-
GA [25]	-	23,790	-	-	-	67.0	-
GA + R [25]	-	22,570	-	-	-	54.0	-
CLP TG + R [1]	-	22,241	-	5,269.0	16,972.0	48.0	600
CLP CG + R [1]	-	21,292	-	4,902.0	16,390.0	53.0	600
CLP SD + FC [34]	-	20,981	-	4,716.0	16,220.0	54.0	97
LS [33]	-	20,788	-	4,604.0	16,184.0	50.0	20,056
FLS [33]	-	20,732	-	4,608.0	16,124.0	49.0	1,242
Fast GLS [33]	-	20,433	-	4,707.0	15,726.0	48.0	9,183
PSO (10) gBest + R	20,585.5	<b>20,371</b>	164.8	4,221.5	16,363.9	55.9	10,306
PSO (200) gBest + R	21,184.1	20,958	112.2	4,374.4	16,809.7	60.2	10,607
PSO (10) Wheel + R	20,637.7	<b>20,340</b>	162.2	4,169.5	16,468.2	56.6	10,668
PSO (10) Circle + R	20,505.4	<b>20,273</b>	112.8	4,185.0	16,320.4	55.0	10,572
PSO (10) lBest + R	20,435.9	<b><u>20,193</u></b>	130.9	4,168.0	16,267.8	54.5	10,615

Based on this insight, all further experiments for the wheel, lBest ( $k=4$ ) and circle topologies were performed with 10 particles. With respect to the minimum objective function value, the lBest neighbourhood performs best, followed by circle, wheel and gBest. Considering the mean values, lBest again significantly outperforms the other topologies, followed by circle, gBest and wheel. Results of respective t-tests can be found in table 4. It can therefore be worthwhile not to immediately distribute information to all particles. Using smaller neighbourhoods helps to avoid premature convergence to a local optimum on this problem. This result concurs with many statements found in the literature and also with the basic idea behind neighbourhood topologies [20], [10]. However, it contradicts experience gained using a similarly modified PSO [16] on a different combinatorial problem. There, gBest almost always yielded the best results because good solutions were very rare in the solution space. If these good solutions were not almost immediately passed on to all particles, there was a danger of them being lost. The British Telecom problem does not seem to suffer from that effect in any great amount.

With respect to CPU time, PSO requires roughly 3 hours for one run. Since CPU time is not a limiting factor in this type of application, the CPU requirements of PSO can be regarded sufficient, considering the final solution quality produced.

**Table 4** t-test results (one-tailed) for pairwise comparison of neighbourhood topologies in PSO

$H_1$	$T$	$df$	signi- ficance $H_0$	95% confidence interval of differences	
				lower	upper
PSO (10) lBest + R < PSO (10) Circle + R	-2,205	58	0.016	-122,254	-16,812
PSO (10) lBest + R < PSO (10) gBest + R	-3,893	58	< 0.001	-213,832	-85,368
PSO (10) lBest + R < PSO (10) Wheel + R	-5,303	58	< 0.001	-265,415	-138,185

## 6 Conclusions and Outlook

The British Telecom problem is derived from a practical situation and has been intensively analysed in the literature for almost two decades. In addition to the previously tested methods, this work assessed an adapted hybrid form of particle swarm optimization that integrates an initialization and a repair heuristic. Moreover, various neighbourhood topologies and swarm sizes of PSO were tested.

PSO with a small population size of 10 particles produced better results than the previous best known solution, independent of the neighbourhood topology. Among the neighbourhoods, lBest performed best, which is in line with the finding of others in the literature. Taken together with the results presented for a logistics problem elsewhere [17], there is a strong indication that hybridising a metaheuristic with a problem-specific repair heuristic is a useful approach of resolving the conflict between domain-specific characteristics of a real-world problem and the desire to employ a generic optimisation technique, at least in the domain of workforce management. Moreover, it seems to pay off to use available knowledge in the initialization phase of PSO in order to respect the hard problem constraints right from the start instead of having more diversity using random initialization.

Even the best schedules contain technicians who are not assigned to any jobs. This is due to the focus on the reduction of travel costs and unfinished jobs in this particular optimization problem. However, it is worth mentioning that a great economic potential also lies in the reduction of employee idleness.

The BT-problem only considers coordinates of the service centers and job location positions. Related real-world applications might require to use actual distances instead. Another interesting extension would be to view the BT-problem as a multiobjective optimization problem where total travel distance and the number of unassigned jobs are simultaneously optimized. This would require different solution approaches that can effectively search for the Pareto front.

In the current planning process technicians are allocated a shift model in a previous step. Only then does the planning problem discussed here begin. The merging of both planning phases would obviously render a lot of further potential for improvement.

## References

1. Azarmi N., AbdulHameed W., Workforce scheduling with constraint logic programming, In: BT Technology Journal, 13(1):81–94, 1995
2. Baker S., Applying simulated annealing to the workforce management problem, In: ISR, British Telecom Laboratories, Martlesham Heath, Ipswich, 1993
3. Bartholdi J. J., A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering, In: Operations Research, 29(3):501–510, 1981
4. British Telecom Laboratories, 250-118 data set of WMS problem, <http://dces.essex.ac.uk/CSP/WFS>, last visited: 08.01.2010

5. Brusco M. J., Jacobs L. W., A Simulated Annealing Approach to the Solution of Flexible Labour Scheduling Problems, In: *Journal of the Operational Research Society*, 44:1191–1200, 1993
6. Brodersen O. B., *Eignung schwarmintelligenter Verfahren für die betriebliche Entscheidungsunterstützung*, Cuvillier, 2008
7. Chu S.-C., Chen Y.-T., Ho J.-H., Timetable Scheduling Using Particle Swarm Optimization, *Proceedings of the First International Conference on Innovative Computing, Information and Control (ICICIC 2006)*, 3:324–327, Beijing, 2006
8. Cooper T. B., Kingston J. H., The Complexity of Timetable Construction Problems, In: Burke E. K., Ross P. (eds.), *Practice and Theory of Automated Timetabling, First International Conference (PATAT 1995)*, Vol. 1153, LNCS, 283–295, Edinburgh, U.K., Selected Papers, Berlin et al.: Springer, 1996
9. Cung V.-D., Briant O., Challenge ROADEF 2007. Technicians and Interventions Scheduling for Telecommunications; <http://www.g-scop.inpg.fr/ChallengeROADEF2007>
10. Czogalla J., Fink A., Particle Swarm Topologies for Resource Constrained Project Scheduling, In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008)*, 236:61–73, Berlin: Springer, 2009
11. Dutot P.-F., Laugier A., Bustos A.-M., Technicians and Interventions scheduling for Telecommunications, 2006
12. Easton F. F., Rossin D. F., Equivalent Alternate Solutions for the Tour Scheduling Problem, In: *Decision Sciences*, 22:985–1007, 1991
13. Ernst A. T. et al., An Annotated Bibliography of Personnel Scheduling and Rostering, In: *Annals of Operations Research*, 127:21–144, 2004
14. Garey M. R., Johnson D. S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, In: *Series of Books in the Mathematical Sciences*. Freeman, 1979
15. Günther, M. (2011). *Hochflexibles Workforce Management: Herausforderungen und Lösungsverfahren*. Dissertation (in German), TU Ilmenau 2011
16. Günther M., Nissen V., A Comparison of Neighbourhood Topologies for Staff Scheduling With Particle Swarm Optimisation, In: Mertsching B., Hund M., Zaheer A. (eds.): *KI 2009: Advances in Artificial Intelligence*, LNCS 5803:185–192, Berlin: Springer, 2009
17. Günther M., Nissen V., A Comparison of Three Heuristics on a Practical Case of Sub-Daily Staff Scheduling, In: McCollum B., Burke E., White G. (eds.): *Proceedings PATAT 2010 - 8th Int. Conf. on the Theory and Practice of Automated Timetabling*, 2010, 224–240
18. Kennedy J., Eberhart R. C., *Particle Swarm Optimization*, *Proceedings of the IEEE International Conference on Neural Networks, 1942–1948*, Perth, Australia, IEEE Service Center, 1995
19. Kennedy J., Eberhart R. C., Shi Y., *Swarm Intelligence*, San Francisco: Kaufmann, 2001
20. Kennedy J., Mendes R., Population structure and particle swarm performance, In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, 2:1671–1676, Honolulu, USA, 2002
21. Kliem R. L., Anderson H. B., Styles and Their Impact on Project Management Results, In: *Project Management Journal*, 27(1):41–50, 1996
22. Kokkoras F., D-WMS: A Distributed Version of a CLP-based Workforce Management System, Computer Science Department, University of Bristol, UK, 1996
23. Kragelund L., Kabel T., *Employee Timetabling. An Empirical Study of Solving Real Life Multi-Objective Combinatorial Optimisation Problems by means of Constraint-Based Heuristic Search Methods*, Master's Thesis in Computer Science, Department of Computer Science, University of Aarhus, 1998
24. Mendes R., *Population Topologies and Their Influence in Particle Swarm Performance*, PhD Thesis, Departamento de Informática, Escola de Engenharia, Universidade do Minho, 2004
25. Muller C., Magill E. H., Prosser P., Smith D. G., Distributed genetic algorithms for resource allocation, In: Dorn J., Fröschl, K. (eds.), *Scheduling of production processes*, 70–78, Ellis Horwood, 1993
26. Naveh Y. et al. Workforce optimization: identification and assignment of professional workers using constraint programming, In: *IBM Journal of Research and Development archive*, 51(3):263–279, 2007
27. Nissen, V., Günther, M. (2009). Staff Scheduling with Particle Swarm Optimization and Evolution Strategies. In Cotta, C., Cowling, P. (Eds.). *Proceedings of EvoCOP 2009* (pp. 228–239). LNCS 5482, Berlin: Springer.
28. Sakellariou I., Kokkoras F., Vlahavas I., Applying a Distributed CLP Platform to a Workforce Management Problem, In: *Proceedings of the 12th Conference on Intelligent Systems Application to Power Systems (ISAP 2003)*, Lemnos, Griechenland, 2003
29. Tien J. M., Kamiyama A., Manpower Scheduling Algorithms, In: *SIAM Review*, 24(3):275–287, 1982
30. Tsang E. P. K. et al., Retractable Contract Network for Distributed Scheduling, In: *Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory & Applications (MISTA 2005)*, 485–500, New York, USA, 2005
31. Tsang E. P. K. et al., Retractable contract network for empowerment in workforce scheduling, In: *Special Issue on Negotiation Mechanisms, Multiagent and Grid Systems*, 4(1):25–44, 2008

32. Tsang E. P. K. et al., Multi-Agent Systems for Staff Empowerment, In: Voudouris C., Owusu G., Dorne R., Lesaint D. (eds.), Service Chain Management. Technology Innovation for the Service Business, 236–274, Springer, 2008
33. Tsang, E. P. K., Voudouris C., Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem, In: Operations Research Letters, 29(3):119–127, 1997
34. Yang R., Solving a Workforce Management Problem with Constraint Programming, In: Proceedings of the 2nd International Conference on the Practical Application of Constraint Technology (PACT 1996), 373–387, London, UK, 1996

---

# Integer Programming Techniques for the Nurse Rostering Problem

Santos, H.G., Toffolo, T.A.M., Ribas, S.,  
Gomes, R.A.M.

Received: date / Accepted: date

**Abstract** This work presents Integer Programming (IP) techniques to tackle the problem of the International Nurse Rostering Competition. Starting from a compact and monolithic formulation on which the current generation of solvers performs poorly, improved cut generation strategies and primal heuristics are proposed and evaluated. A large number of computational experiments with these techniques produced the following results: the optimality of the vast majority of instances was proved, the best known solutions were improved up to 15% and strong dual bounds were obtained. In the spirit of reproducible science, all code was implemented using the COmputational Infrastructure for Operations Research (COIN-OR).

**Keywords** Nurse Rostering · Integer Programming · Cutting Planes · Heuristics

## 1 Introduction

A significant amount of research has been devoted to the computational solution of the Nurse Rostering Problem [15]. Much of previous work, however, concentrates on specific case studies, focusing in particularities of certain institutions. For these works, comparison between different search strategies is a very difficult task. Recently, the International Nurse Rostering Competition (INRC) [28] was organized to stimulate the research in this area. An instance set was proposed and a significant number of different algorithms has been empirically evaluated using it. As a result, best known solutions have been updated since then.

In this work we present a monolithic compact Integer Programming formulation, i.e. a formulation with a polynomial number of constraints and vari-

---

Computing Department  
Federal University of Ouro Preto  
Ouro Preto, Minas Gerais, Brazil 35400-000

ables, for the problem described in INRC. We propose and evaluate techniques for improving the performance of state-of-art integer programming solvers using this formulation.

The proposed techniques can be divided in two groups: the first group is devoted to the improvement of dual bounds. In this case we are not only interested in the quick production of high quality solutions but also interested in having a precise estimate for a lower bound on the optimal solution cost. This obviously incurs additional processing time but it is a critical step for methods aiming at proving the optimality. In the second group we present techniques to speedup the production of near optimal feasible solutions. These latter techniques can be applied alone for those interested in solving real world situations.

Our computational experiments showed that fast methods for the production of good primal and dual bounds can be obtained using the proposed techniques: our algorithms provided not only very competitive heuristics but we also proved the optimality for the vast majority of INRC instances and improved best known solutions for the remaining instances up to 15%.

In the spirit of the reproducible science, the implementation of the cut generation procedures was made using the open source branch-and-cut software[24] of the COIN-OR Foundation, CBC[33]. We proposed alternative cut separation routines for two of the cut generators included on CBC and showed that these routines significantly outperform the included ones considering the required time to produce better lower bounds for INRC instances. These routines are being made available also as an open source project.

The paper is organized as follows: in section 2 an informal description of the problem is presented along with a brief description of previous algorithms proposed in the literature. In section 3 the NRP problem is formally stated using our proposed formulation. Sections 4 and 5 present our proposals for dual and primal bound improvements, respectively. Section 6 includes computational experiments to evaluate our proposals. Finally, in section 7, conclusions and future works are discussed.

## 2 The Nurse Rostering Problem

The nurse rostering problem can be described by a nurse-day view, a nurse-task view, or a nurse-shift pattern view [16]. In the nurse-day view, allocations are indexed for each nurse and each day. This way, a solution can be directly represented by a matrix where each cell  $m_{i,j}$  contains a set of shifts to be performed by the nurse  $i$  in the day  $j$ . Broadly speaking this set may have any number of shifts, but in the INRC problem and most practical cases a nurse performs only one shift per day – which may include morning shift (M), evening shift (E), night shift (N), day-off (-), among others. Table 1 presents part of a weekly roster which indicates the shifts allocated to the nurses, in a nurse-day view.

**Table 1** Example of an NRP solution in a nurse-day view

Nurse	Mon	Tue	Wed	Thu	Fri	Sat	Sun
N1	M	M	N	-	-	E	E
N2	E	N	E	E	M	-	-
N3	-	E	M	-	M	-	N

In the nurse-task view, the decision variable is indexed for each nurse and each task that the nurse performs in the scheduling period. This decision variable may assume a value of 1 if the nurse is assigned to the task, or 0 otherwise. In the nurse-shift pattern view, the decision variable is indexed for each nurse and each pattern of shifts available. Cheang et al. [16] presents a bibliographic survey of the many models and methodologies available to solve the NRP.

In this work, we address the problem defined in the first International Nurse Rostering Competition, sponsored by the leading conference in the Automated Timetabling domain, PATAT. Competitors were allowed to submit a specific technique for each instance type. Here follow brief descriptions of approaches that succeeded in the competition.

Valoux et al. [44], winners of the challenge, developed a two phase strategy where in the first phase the workload for each nurse and for each day of the week was decided while in the second phase the specific daily shifts were assigned. Since the competition imposed quality and time constraint requirements, they partitionated the problem instances into sub-problems of manageable computational size which were then solved sequentially using Integer Mathematical Programming. Also, they applied local optimization techniques for searching across combinations of nurses' partial schedules. This sequence was repeated several times depending on the available computational time.

Burke and Curtois [12] applied an ejection chain based method for sprint instances and a branch and price algorithm for medium and long instances. Problem instances have been converted to the general staff rostering model proposed and documented by the same team. Then, their software Roster Booster which included the above mentioned algorithmic approaches was used.

Bilgin et al. [6] applied a hyper-heuristic approach combined with a greedy shuffle heuristic. The hyper-heuristic consisted of a heuristic selection method and a move acceptance criterion. The best solution found was further improved by exploring swaps of partial rosters between nurses.

Nonobe [37] modified the general purpose constraint optimization problem tabu search based solver presented in [38]. Their main idea is to spend less time in developing algorithms since after reformulating the problem as a constraint optimization problem only user defined constraints have to be implemented.

More details about these approaches can be consulted on the site of the competition<sup>1</sup>.

Another recent work developed by Burke et al. [13] is a hybrid multi-objective model that combines integer programming (IP) and variable neigh-

<sup>1</sup> <https://www.kuleuven-kulak.be/nrpscompetition/competitor-ranking>



bourhood search (VNS) to deal with NRP. An IP formulation is first used to solve the subproblem which includes the full set of hard constraints and a subset of soft constraints. Next, a basic VNS follows as a postprocessing procedure to further improve the IP's resulting solutions. The major focus of the VNS is the satisfaction of the excluded constraints from the preceding IP model.

## 2.1 Constraints

Combinatorial optimization problems generally carry hard and soft constraints. Roughly, the difference is that hard constraint must be met and soft constraint violations should be avoided. A single violation of a hard constraint renders the solution infeasible. In this work, we considered the following hard constraints, as defined in INRC:

- a nurse can not work more than one shift per day;
- all shift type demands during the planning period must be met.

and the following soft constraints:

- minimum/maximum number of shifts assigned to a nurse;
- minimum/maximum number of consecutive free days;
- minimum number of consecutive working days;
- maximum number of consecutive working weekends;
- number of days off after a series of night shifts;
- maximum number of working weekends in four weeks;
- complete weekends: if a nurse has to work only on some days of the weekend then penalty occurs;
- identical shift types during the weekend: assignments of different shift types to the same nurse during a weekend are penalized;
- day on/off request: requests by nurses to work or not to work on specific days of the week should be respected, otherwise solution quality is compromised;
- shift on/off request: similar to the previous but now for specific shifts on certain days;
- unwanted patterns: an unwanted pattern is a sequence of assignments that is not in the preferences of a nurse based on her contract;
- alternative skill: if an assignment of a nurse to a shift type requiring a skill that she does not have occurs, then the solution is penalized accordingly;
  - unwanted patterns not involving specific shift types;
  - unwanted patterns involving specific shift types.

In the next section our compact Integer Programming formulation for the INRC problem will be presented.

### 3 An Integer Programming Formulation for the INCR Problem

In this section we present an Integer Programming formulation which successfully models all constraints considered in instances of the International Nurse Rostering Competition.

#### 3.1 Input Data

$N$  set of nurses

$C$  set of contracts

$\tilde{c}_n$  contract of nurse  $n$

$S$  set of shifts

$\tilde{S}$  set of night shifts

$D$  set of days with elements sequentially numbered from 1

$\Pi$  set of all ordered pairs  $(d_1, d_2) \in D \times D : d_1 \leq d_2$  representing windows in the planning horizon

$\tilde{W}_c$  set of weekends in the planning horizon according to the weekend definition of contract  $c$ , with elements numbered from 1 to  $\tilde{w}_c$

$\tilde{D}_{ic}$  set of days in the  $i$ -th weekend of contract  $c$

$\tilde{r}_{ds}$  number of required nurses at day  $d$  and shift  $s$

$\tilde{P}_c$  set of unwanted working shift patterns for contract  $c$

$\hat{P}_c$  set of unwanted working days patterns for contract  $c$

The configuration of soft constraints depends on each contract  $c$ . Thus, each contract has an associated weight (which may be null) for penalizing the violation of each soft constraint. Limits informing how tight is a given soft constraint are also contract related.

We divide soft constraints in two groups. In the first group, denoted here by *Ranged Soft Constraints*, we include constraints which state a range of valid integer values for a variable in the format  $\underline{v} \leq v \leq \bar{v}$ . Values outside this range need to be penalized in slack variables according to its distance to the closest valid value.

In the second group, the *Logical Soft Constraints*, are those constraints which are satisfied or not, i.e. the maximum distance to feasibility is one.

In Table 2 each soft constraint is associated with an index. This index will be used to express constants which state the minimum and maximum limit for a given ranged soft constraint  $i$  and a contract  $c$ , which will be denoted here by  $\underline{\gamma}_c^i$  and  $\bar{\gamma}_c^i$ , respectively. The weight for violating the  $i$ -th minimum and maximum limit of these constraints is denoted by  $\underline{\omega}_c^i$  and  $\bar{\omega}_c^i$ , respectively. For logical soft constraints the weight of penalizing them is defined by  $\omega_c^i$ . Finally, we denote by  $\underline{\alpha}_n^i, \bar{\alpha}_n^i, \alpha_n^i$  the slack variables associated with the the violation of lower/upper limit of ranged and logical soft constraints  $i$  for nurse  $n$ , respectively. Additional indexes in the  $\alpha_n^i$  variables may be used, for example, when this violation must be computed for a specific location, so that  $\alpha_{nk}^i$  is the slack variable related to the violation of the complete weekends soft constraint for nurse  $n$  at the  $k$ -th weekend.

Ranged Soft Constraints	
1	total number of allocations
2	contiguous working days
3	contiguous resting days
4	total number of working weekends in four weeks
5	consecutive working weekends
6	number of resting days after a night shift
Logical Soft Constraints	
7	complete weekends
8	no night shift before free weekend
9	same shift on weekends
10	alternative shifts
11	undesired working shifts pattern
12	undesired working days pattern
13	undesired shifts and days

**Table 2** Indexes for ranged and logical soft constraints

Some specific sequences of working shifts (soft constraint 11) may be unwanted, e.g.: Late, Evening, Late. The set of these patterns for contract  $c$  is specified in  $\hat{P}_c$  and each pattern  $\hat{p} \in \hat{P}_c$  has a size  $\tilde{s}(\hat{p})$  and contents  $\hat{p}[1], \dots, \hat{p}[\tilde{s}(\hat{p})] \in S$ . Day related patterns are also considered in soft constraint 12: sequences of working/resting days should be avoided, e.g.: not working on Friday and working on the succeeding weekend. The set of these patterns is defined by  $\hat{P}_c$  with elements  $\hat{p} \in \hat{P}_c$  with size  $\tilde{s}(\hat{p})$ . To specify which days from the pattern represent the “not working” option we define a set of virtual days  $\hat{D}$  with negative numbered days representing this option, so that pattern elements  $\hat{p}[1], \dots, \hat{p}[\tilde{s}(\hat{p})]$  are restricted to be in  $D \cup \hat{D}$ .

### 3.2 Decision variables

The main decision variables are the three indexed  $x_{nsd}$  binary variables:

$$x_{nsd} = \begin{cases} 1 & \text{if nurse } n \text{ is allocated to shift } s \text{ and day } d \\ 0 & \text{otherwise} \end{cases}$$

additionally, there are the following auxiliary variables:

$$y_{ni} = \begin{cases} 1 & \text{if nurse } n \text{ works at weekend } i \\ 0 & \text{otherwise} \end{cases}$$

$$w_{nd_1d_2} = \begin{cases} 1 & \text{if nurse } n \text{ works from day } d_1 \text{ until day } d_2 \\ 0 & \text{otherwise} \end{cases}$$

$$r_{nd_1d_2} = \begin{cases} 1 & \text{if nurse } n \text{ rests from day } d_1 \text{ until day } d_2 \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ni_1i_2} = \begin{cases} 1 & \text{if nurse } n \text{ works from weekend } i_1 \text{ until weekend } i_2 \\ 0 & \text{otherwise} \end{cases}$$

To simplify the statement of constraints we consider additional variables  $y_{n0}$ , which are always fixed to zero.

### 3.3 Objective Function

Before presenting the objective function we remark that some slack variables (and their respective constraints) do not need to be explicitly included. This is the case of constraints which are directly linked to the selection of a specific working/resting window from the set  $\Pi$  by activating variables  $w_{nd_1d_2}$  and  $r_{nd_1d_2}$ , respectively. This is obviously the case for soft constraints 2 and 3 (Table 2) and also the case for soft constraint 7, since every activation of  $w_{nd_1d_2}$  finishing/starting in the middle of a weekend must be penalized. We denote by  $\sigma_{cd_1d_2}$  and  $\tau_{cd_2}$  the weighted penalty of all violations incurred from working (resting) continuously in a block starting at day  $d_1$  and finishing at day  $d_2$  for nurses of contract  $c$ , respectively. Soft constraints 10 and 13 are also directly penalized in  $x_{nsd}$  variables with coefficients  $\nu_{nsd}$ . Analogously, soft constraint 5 is penalized in variables  $z_{ni_1i_2}$  with coefficients  $\psi_{ni_1i_2}$ .

*Minimize:*

$$\sum_{n \in N} \left[ \begin{aligned} & \sum_{(d_1d_2) \in \Pi} (\sigma_{\bar{c}_n d_1 d_2} w_{nd_1 d_2} + \tau_{\bar{c}_n d_1 d_2} r_{nd_1 d_2}) + \\ & \sum_{s \in S} \sum_{d \in D} \nu_{nsd} x_{nsd} + \bar{\omega}_{\bar{c}_n}^1 \bar{\alpha}_n^1 + \underline{\omega}_{\bar{c}_n}^1 \underline{\alpha}_n^1 + \\ & \sum_{i \in \{1 \dots \bar{w}_c\}} (\omega_{\bar{c}_n}^4 \alpha_{ni}^4 + \underline{\omega}_{\bar{c}_n}^8 \underline{\alpha}_{ni}^8 + \underline{\omega}_{\bar{c}_n}^9 \underline{\alpha}_{ni}^9) + \\ & \sum_{i_1, i_2 \in \bar{W}_{\bar{c}_n}: i_2 \geq i_1} \psi_{ni_1 i_2} z_{ni_1 i_2} + \\ & \sum_{d \in D} \underline{\omega}_{\bar{c}_n}^6 \underline{\alpha}_n^6 + \sum_{\hat{p} \in \hat{P}_{\bar{c}_n}} \alpha_{n\hat{p}}^{11} + \sum_{\hat{p} \in \hat{P}_{\bar{c}_n}} \alpha_{n\hat{p}}^{12} \end{aligned} \right]$$

### 3.4 Constraints

Constraints are presented in the following. Constraints **1** and **2** model the two hard constraints of the INRC problem : to provide sufficient coverage of nurses for every day and shift and to limit working shifts for nurses to a maximum of one per day. Constraints **3** and **4** link the activation of variables  $x$  with the activation of  $y$  variables which indicate working weekends. Constraints from **5** to **9** ensure that every working window activation ( $w$  variables) is immediately followed by the activation of a  $r$  variable with the corresponding resting window and vice versa. This implies the selection of contiguous working and resting periods of different sizes for the whole planning horizon.

The following constraints are all soft-constraints, which means that they can be violated since they include a slack variable (variables  $\alpha$ ) which will be penalized in the objective function when activated. Ranged constraints **10** model the minimum and maximum working days in the planning horizon. Constraints **11** limit the maximum number of working weekends in four weeks. Constraints **12** consider the maximum number of consecutive weekends. Constraints **13** impose a minimum number of resting days after a sequence of night shifts. Constraints **14** ensure that a nurse is not allocated to a night shift in a day preceding a free weekend. For a weekend, allocated shifts should be equal for every working day, as stated in constraints **15** and **16**. Undesired patterns for days and shifts are modeled in constraints **17** and **18**.

$$\sum_{n \in N} x_{nsd} = \tilde{r}_{ds} \quad \forall d \in D, s \in S \quad (1)$$

$$\sum_{s \in S} x_{nsd} \leq 1 \quad \forall n \in N, d \in D \quad (2)$$

$$y_{ni} \geq \sum_{s \in S} x_{nsd} \quad \forall n \in N, i \in \tilde{W}_{\tilde{c}_n}, d \in \tilde{D}_{i\tilde{c}_n} \quad (3)$$

$$y_{ni} \leq \sum_{s \in S, d \in \tilde{D}_{i\tilde{c}_n}} x_{nsd} \quad \forall n \in N, i \in \tilde{W}_{\tilde{c}_n} \quad (4)$$

$$\sum_{s \in S} x_{nsd} = \sum_{(d_1, d_2) \in \Pi : d \in \{d_1, \dots, d_2\}} w_{nd_1 d_2} \quad \forall n \in N, d \in D \quad (5)$$

$$\sum_{s \in S} x_{nsd} = 1 - \left( \sum_{(d_1, d_2) \in \Pi : d \in \{d_1, \dots, d_2\}} r_{nd_1 d_2} \right) \quad \forall n \in N, d \in D \quad (6)$$

$$\sum_{(d_1, d_2) \in \Pi : d \in \{d_1, \dots, d_2\}} (w_{nd_1 d_2} + r_{nd_1 d_2}) = 1 \quad \forall n \in N, d \in D \quad (7)$$

$$\sum_{d' \in \{1, \dots, d\}} w_{nd' d} + \sum_{d'' \in D : d'' \geq d+1} w_{n, d+1, d''} \leq 1 \quad \forall n \in N, d \in D \quad (8)$$

$$\sum_{d' \in \{1, \dots, d\}} r_{nd' d} + \sum_{d'' \in D : d'' \geq d+1} r_{n, d+1, d''} \leq 1 \quad \forall n \in N, d \in D \quad (9)$$

$$\underline{\gamma}_{\tilde{c}_n}^1 - \underline{\alpha}_n^1 \leq \sum_{s \in S, d \in D} x_{nsd} \leq \bar{\gamma}_{\tilde{c}_n}^1 + \bar{\alpha}_n^1 \quad \forall n \in N \quad (10)$$

$$\sum_{i' \in \{i, \dots, i+3\}} y_{ni'} \leq \bar{\gamma}_{\tilde{c}_n}^A + \bar{\alpha}_{ni}^A \quad \forall n \in N, i \in \{1, \dots, \tilde{w}_{\tilde{c}_n} - 3\} \quad (11)$$

$$\sum_{i' \in \{i, \dots, i + \bar{\gamma}_{\tilde{c}(n)}^5\}} y_{ni'} \leq \bar{\gamma}_{\tilde{c}_n}^5 + \bar{\alpha}_{ni}^5 \quad \forall n \in N, i \in \{1, \dots, \tilde{w}_{\tilde{c}_n} - \bar{\gamma}_{\tilde{c}_n}^5\} \quad (12)$$

$$\begin{aligned} \sum_{s' \in \tilde{S}} \underline{\gamma}_{\tilde{c}_n}^6 x_{ns'd} + \sum_{s \in S \setminus \tilde{S}, d' \in \{d+1, \dots, d + \underline{\gamma}_{\tilde{c}(n)}^6\}} x_{nsd'} \\ \leq \underline{\gamma}_{\tilde{c}_n}^6 + \underline{\alpha}_{nd}^6 \quad \forall n \in N, d \in D : d \leq |D| - \underline{\gamma}_{\tilde{c}(n)}^6 \end{aligned} \quad (13)$$

$$\sum_{s \in \tilde{S}} \sum_{d \in \tilde{W}_{i\tilde{c}_n} : d \geq 2 \wedge d \leq d' \forall d' \in \tilde{W}_{i\tilde{c}_n}} x_{n,s,d-1} + y_{ni} \leq 1 + \alpha_{ni}^8 \quad \forall n \in N, i \in \tilde{W}_{i\tilde{c}_n} \quad (14)$$

$$\alpha_n^9 \geq x_{nsd_1} - x_{nsd_2} \quad \forall n \in N, s \in S, i \in \tilde{W}_{\tilde{c}_n}, d_1, d_2 \in \tilde{D}_{i\tilde{c}(n)} : d_1 < d_2 \quad (15)$$

$$\alpha_n^9 \geq x_{nsd_2} - x_{nsd_1} \quad \forall n \in N, s \in S, i \in \tilde{W}_{\tilde{c}_n}, d_1, d_2 \in \tilde{D}_{i\tilde{c}(n)} : d_1 < d_2 \quad (16)$$

$$\begin{aligned} \sum_{j \in \{1, \dots, \tilde{s}(\hat{p})\}} x_{n,\hat{p}[1],d+j-1} \\ \leq \tilde{s}(\hat{p}) + \alpha_{n\hat{p}}^{11} \quad \forall n \in N, \hat{p} \in \hat{P}_{\tilde{c}_n}, d \in \{1, \dots, |D| - \tilde{s}(\hat{p}) + 1\} \end{aligned} \quad (17)$$

$$\begin{aligned} \sum_{s \in S} \sum_{j \in \{1, \dots, \tilde{s}(\hat{p}) : \hat{p}[j] \geq 1\}} x_{n,s,\hat{p}[j]} + \\ \sum_{j \in \{1, \dots, \tilde{s}(\hat{p}) : \hat{p}[j] \leq -1\}} (1 - \sum_{s \in S} x_{n,s,-\hat{p}[j]}) \leq \tilde{s}(\hat{p}) + \alpha_n^{12} \quad \forall n \in N, \hat{p} \in \hat{P}_{\tilde{c}_n} \end{aligned} \quad (18)$$

#### 4 Dual Bound Improvement : Cutting Planes

The problem considered contains mostly binary variables linked by several GUB (generalized upper bound) constraints. Constraints of this type define an implicit conflict graph [3] indicating the set of pairs of variables whose simultaneous activation is forbidden. Linear programming relaxations for these problems can be significantly strengthened by the inclusion of inequalities derived from the set packing polytope (SPP) [39]. The most common classes of cuts for SPP are the clique cuts and the odd-hole cuts. A clique inequality for a set  $C$  of conflicting variables has the form  $\sum_{j \in C} x_j \leq 1$  and an odd-hole inequality with conflicting variables  $C$  can be define as:  $\sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor$ . It is well known that in practice clique cuts are by far the most important ones [7]. The impact of these cuts has been explored for some hard timetabling problems [4, 14]. Considering generic clique separation routines, the most common ones are the star clique and the row clique method [21, 29, 7]. These are fast separation routines which are used in the current version of the COIN-OR Cut Generation Library. Our algorithm proposal considers aggressive clique separation: instead of searching for *the* most violated clique inequality we search for *all* violated clique inequalities. Some previous results indicate that this is the best strategy. In [14], for example, although authors used a branch-and-bound code to search for the most violated clique, computational results motivated the inclusion of non-optimally violated cuts found during the search. This result is consistent with reports of application of other cuts applied to different models, such as Chvátal-Gomory cuts [23]. The option for inserting a large

number of violated inequalities at once is also responsible for reviving the gomory cuts importance [17]. The proposed clique separation routine has two main components:

1. a module to separate all violated cliques in the conflict subgraph induced by the fractional variables;
2. a lifting module which extends generated cliques considering the original conflict graph.

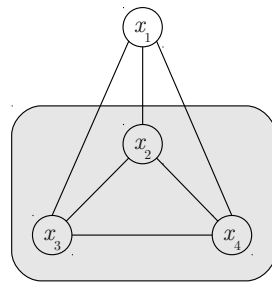
The clique separation module was implemented using an improved version of the Bron-Kerbosch algorithm [11]. This version implements an optimized pivoting rule [10] to speed up the discovery of maximal cliques with large weight. This rule assigns the highest priority for visiting first nodes with large modified degree (summation of node degree and of its neighbors) and weight. Although this algorithm has an exponential worst case performance, the heuristic pivot rules make the algorithm suitable not only for running in the enumeration context but also for executing with restricted times, since larger violated cliques tend to be discovered first. Nevertheless, our experiments showed that all violated inequalities for all instances can be enumerated in a fraction of a second using our implementation. It is important to remark also that even if a subset of cliques were inserted, the optimal solution would not be missed, branching would take care of the rest. This situation does not occur in column generation: an interruption of the pricing algorithm before the optimal column to be discovered in the last iteration would make it impossible to prove the optimality of the discovered solution. In other words, for exact algorithms the cut separation problem can be hard, but column generation cannot, as pointed in [40]. The importance of lifting clique inequalities can be explained with the conflict graph in Figure 1. Nodes inside the gray area indicate variables with non-zero values in the fractional solution. In this solution, only nodes  $x_2, \dots, x_4$  could contribute to define a maximally violated clique inequality. Nevertheless, subsequent linear programming relaxations could include three different violated  $k_3$ <sup>2</sup> cliques by alternating the inactive variable. If the  $k_4$  clique inequality were inserted at the first fractional solution additional re-optimizations of the linear program could be saved, furthermore, a less dense constraint matrix will be obtained with the insertion of these dominant constraints first.

It is well known that the separation of odd-holes contributes only marginally for lower bound improvement [7, 35]. Nevertheless, its inclusion in the branch-and-cut procedure is cheap, since these inequalities can be separated in polynomial time using shortest path algorithms [25]. Odd hole inequalities can be strengthened by the inclusion of a wheel center, such as variable  $x_6$  in the conflict graph presented in Figure 2. In fact, for an odd hole with variables  $C$  and  $W$  being the set of candidates to be included as wheel centers of  $C$ , the following inequality is valid:

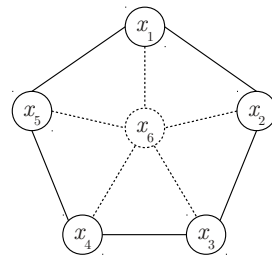
$$\sum_{j \in W} \lfloor \frac{|C|}{2} \rfloor x_j + \sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor \quad (19)$$

---

<sup>2</sup> a clique with three nodes



**Fig. 1** Example of a  $k_3$  which could be lifted to a  $k_4$



**Fig. 2** Example of an odd hole and its possible extension to a wheel

The conflict graph is built by the analysis of the constraint matrix. Although the presented formulation is complete for modeling the INRC problem, we observed that solvers can detect a larger conflict graph if the following valid inequalities are inserted:

$$\sum_{d' \in \{1..d_1\}} r_{nd'd_1} + \sum_{d'' \in \{d_2..|D|\}} w_{nd_2d''} \leq 1 \quad \forall n \in N, (d_1, d_2) \in \Pi : d_2 - d_1 = 2 \quad (20)$$

$$\sum_{d' \in \{1, \dots, d_1\}} w_{nd'd_1} + \sum_{d'' \in \{d_2, \dots, |D|\}} r_{nd_2d''} \leq 1 \quad \forall n \in N, (d_1, d_2) \in \Pi : d_2 - d_1 = 2 \quad (21)$$

We also observed that one subset of variables is directly linked to most of the costs in the objective function: variables  $w_{nd_1d_2}$  and  $r_{nd_1d_2}$ . In the optimal solution of the linear programming relaxation these variables often appear with fractional values, weakening the quality of the dual bound. Since the number of these active variables per nurse is quite limited, we opted for a specific cut separation for these variables. Our routine, which separates the the fractional value for a restricted group of these variables per nurse was implemented using the Fenchel cutting planes [8,9]. These cuts will be called hereafter *Window cuts*.



## 5 Primal Bound Improvement : MIP heuristics

The use of MIP (Mixed Integer Programming) solvers in a heuristic context, i.e. for producing good quality solutions in very restricted times is a growing trend in optimization [34, 43]. A pervasive term in this area is *subproblem optimization*. To speedup the improvement of feasible solutions, solvers work on smaller problems performing local search. Subproblems can be defined either with soft-fixation of variables, as in *Local Branching* and similar methods [22, 27], or with hard fixation of variables as in *Relaxation Induced Neighborhood Search* (RINS)[19]. This latter work presented better results in tests with MIPLIB[31] instances.

The proposed MIP heuristic employs a hybrid heuristic subproblem optimization scheme. Heuristic rules are used to create subproblems  $\mathcal{P}'(\mathcal{H})$ , defined by hard fixation of a given set of variables  $\mathcal{H}$  of the original problem  $\mathcal{P}$ . The algorithm consists basically of two subsequent phases: construction phase and local search phase. The construction phase builds a feasible initial solution using simple heuristic rules, outside the MIP framework. MIP search is used in all the remaining time for exploring large neighborhoods until a local minimum is found. The procedure returns either the local optimum solution of all the neighborhood structures or the best solution found within  $max_{time}$  seconds.

Before presenting the MIP heuristic developed, we present a simple procedure to build feasible solutions which will be used in our experiments.

### 5.1 A Greedy Constructive Algorithm

This method builds an allocation matrix  $M_{|N| \times |D|}$ , initializing all  $m_{ij}$  cells with days off. Sequentially, for each day  $d$  and shift  $s$ , the demand  $\tilde{r}_{ds}$  is satisfied by selecting, one by one, a nurse  $n$  for which this new allocation incurs the smallest increase in the objective function considering augmented partial solution defined in  $M_{|N| \times |D|}$ . This process is repeated until all the demand units are allocated. The algorithm has time complexity of  $O(|N|^2 \times |D|)$ .

### 5.2 Neighborhood structures

The local search phase explores the search space through several neighborhoods, using a VND (Variable Neighborhood Descent) [26] scheme. Considering the results obtained with recent uses of RINS heuristics [18], we proposed two different neighborhood structures that are based on the resolution of small partitions of the original problem to optimality. The differences between the neighborhoods lie on the rules considered to generate such subproblems.

Given a feasible solution  $S_0$ , a neighbor is obtained basically by (i) defining a set of nurse allocations that will be fixed, according to the solution  $S_0$  and (ii) solving to optimality the NRP subproblem obtained with the fixations. In preliminary experiments, two of the evaluated neighborhoods presented much better results. The following paragraphs describe these two neighborhoods.

### 5.2.1 *Fix Days neighborhood structure*

In the *Fix Days* neighborhood structure, the NRP subproblems are generated by fixing all the nurse allocations of  $|D| - ndays$  days of the month, where  $ndays$  is a parameter of the neighborhood.

In the first iteration ( $iter = 0$ ), a subproblem is created from a solution  $S$  by fixing every nurse allocation but the ones on the days from 1 to  $ndays$ . The subproblem is then solved to optimality. If the solution is improved,  $S$  is updated. In the next iteration, another subproblem is generated by fixing all nurse allocations but those on the days between  $day_A$  and  $day_B$  (equations (22) and (23)):

$$day_A = 1 + (iter \times step) \quad (22)$$

$$day_B = ndays + (iter \times step) \quad (23)$$

The equations (22) and (23) calculate, respectively, the beginning and the end of a time window in which the nurse allocations remain unfixed. In these equations,  $iter$  is the number of the current iteration and  $step$  is a parameter that defines the number of days between two consecutive subproblems. If the value of  $day_A$  or  $day_B$  is greater than the number of days,  $|D|$ , we consider the day to be the remainder of its value divided by  $|D|$ . The algorithm proceeds until  $|D|/step$  consecutive iterations without improvement are reached, which indicates that a local optimum for the neighborhood was found. Figure 3 shows how the search on the solution space is performed by the *Fix Days* neighborhood structure.

Nurse	Unfixed days							Mon	Tue	Wed	Thu	Fri	Sat	Sun
	Mon	Tue	Wed	Thu	Fri	Sat	Sun							
N1	M	M	N	-	-	E	E	M	M	N	-	-	E	E
N2	E	N	E	E	M	-	-	E	N	E	E	M	-	-
N3	-	E	M	-	M	-	N	-	E	M	-	M	-	N
	iter=0		iter=1		iter=2									

**Fig. 3** *Fix Days* neighborhood subproblems with  $ndays = 3$  and  $step = 2$ .

The neighborhood has two parameters,  $step$  and  $ndays$ . As said before, the first one indicates the number of days between two consecutive subproblems. The smaller the value, the greater is the number of different subproblems in the neighborhood. The other parameter,  $ndays$ , defines the size of each subproblem and so is a critical one. Small values may create subproblems that do not contain any better solution and large values may create unmanageable subproblems.

### 5.2.2 *Fix Shifts neighborhood structure*

In the *Fix Shifts* neighborhood structure, the subproblems are created by fixing all the allocations of  $|S| - 1$  shifts. On the first iteration, only the allocations of the first shift remain unfixed. On the second iteration, only the allocations of the second shift aren't fixed, and so on. Since the number of different subproblems generated in this neighborhood is equal to  $|S|$ , the algorithm proceeds until  $|S|$  consecutive iterations without improvement are reached. The neighborhood doesn't have any parameter.

Given that an average instance has from 3 to 5 shifts, it may seem that the subproblems of this neighborhood are hard to solve. But such subproblems can actually be solved in very small time, as seen on section 6.

### 5.3 Mathematical programming heuristic

In the latter section, we presented the neighborhood structures used by our mathematical programming heuristic (MPH). The heuristic works in a VND fashion, searching each one of the neighborhoods until their local minima are found. We decided to use the following neighborhoods in our algorithm:

- $\mathcal{N}_1^m$  : *Fix Days* neighborhood structure with  $ndays = 2m$  and  $step = m$
- $\mathcal{N}_2$  : *Fix Shifts* neighborhood structure

First, the algorithm performs a complete search on the neighborhoods  $\mathcal{N}_1^m$  and  $\mathcal{N}_2$ . After that, the algorithm increases the value of  $m$  by 1, searching for the best solution on the neighborhood  $\mathcal{N}_1^{m+1}$ . If any improvement is produced by the latter search, the algorithm searches the neighborhood  $\mathcal{N}_2$  before incrementing the value of  $m$ . Otherwise, the algorithm just increases the value of  $m$  by 1, moving to the neighborhood  $\mathcal{N}_1^{m+1}$ . This procedure repeats until  $m > |D|/2$  or until the time limit is reached.

Figure 4 shows the pseudocode of the proposed heuristic. In this figure, the procedures  $\mathcal{N}_1^m(S_0)$  and  $\mathcal{N}_2(S_0)$  return, respectively, the best neighbors of  $S_0$  in the neighborhoods  $\mathcal{N}_1^m$  and  $\mathcal{N}_2$ . If no better solution than  $S_0$  is found, then  $S_0$  is returned.

It is important to note that on a standard VND, it is typical to choose a specific order of neighborhoods to be searched. If one neighborhood is able to improve the solution, VND moves back to the first neighborhood, restarting the search. Our algorithm always increments the value of  $m$ . We decided to do so because we observed that searching again the neighborhood  $\mathcal{N}_1^m$  almost never improves the solution, but takes considerable processor time. Since neighborhood  $\mathcal{N}_1^m$  is smaller than  $\mathcal{N}_1^{m+1}$ , and most neighbors of  $\mathcal{N}_1^m$  are also in the neighborhood  $\mathcal{N}_1^{m+1}$ , looking again for a better solution on  $\mathcal{N}_1^m$  can be a waste of time.

Another interesting thing to note is that if a time limit is not set, the original problem will be solved in the last iteration of the heuristic. This occurs because finding the local minimum of any solution  $S$  in the neighborhood  $\mathcal{N}_1^{|D|}$  is the same as solving the original problem itself.

```

Require:  $S_0, m$ 
1:  $S^* \leftarrow \mathcal{N}_1^m(S_0)$ 
2:  $S^* \leftarrow \mathcal{N}_2(S^*)$ 
3:  $m = m + 1$ 
4: while  $m \leq |D|/2$  and time limit not reached do
5:    $S \leftarrow \mathcal{N}_1^m(S^*)$ 
6:   if  $S$  is a better solution than  $S^*$  then
7:      $S^* \leftarrow \mathcal{N}_2(S)$ 
8:   end if
9:    $m = m + 1$ 
10: end while
11: return  $S^*$ ;

```

**Fig. 4** Pseudocode of MPH Algorithm

## 6 Computational Experiments

Our code was written in C++ using the open source COIN-OR libraries. This approach allowed us to deeply integrate our routines with the COIN-OR MIP solvers [24, 42, 32] and also communicate with commercial solvers through the Open Solver Interface (OSI) [41]. Closed source solvers can also have additional code integrated by using callbacks, but this approach is ultimately limited by which callbacks are available and how many decisions they delegate. Thus, all cut generation routines were implemented and tested using the COIN Branch-and-Cut solver (CBC) [24], which is the fastest open source mixed integer programming solver available [36].

The code was compiled on GCC/g++ version 4.6. We ran all the experiments on several Core i7 3.4GHz computers with 16Gb of RAM memory running Linux Ubuntu 10.10 64-bits. We used CPLEX version 12.2.0 and COIN-OR CBC 2.7.6.

The instance set used within the experiments was the same used during the INRC, including the harder *hidden* instances. Further information about these instances can be found in [28] or at the competition website<sup>3</sup>.

Before proceeding to the evaluation of our proposals, we present some experiments with a state-of-art integer programming solver. The objective is to determine how powerful these solvers are handling the INRC instances with the proposed formulation and the application of only small additional settings, if any.

### 6.1 Standalone solvers

We included experiments with the commercial CPLEX [30] standalone solver.

In Table 3 the results of CPLEX running with different optimization emphasis with execution times time restricted to 10 minutes and one hour are included. The final lower and upper (lb/ub) bounds are presented with the computed gap  $\frac{ub-lb}{lb} \times 100$ .

<sup>3</sup> <http://www.kuleuven-kulak.be/nrpcompetition>

Results are summarized per instance group, including the maximum value, average and standard deviation (m/a/s.d.) gap values.

Since the production of a feasible solution for INRC instances can be done very quickly using a greedy heuristic (see section 5.1) and CPLEX standalone solver can read initial solutions, we also included experiments where CPLEX starts from an already produced feasible solution (columns with  $s = gr(\cdot)$ ). For sprint instances CPLEX always found the optimal solution in a few minutes, so we did not report results for these instances.

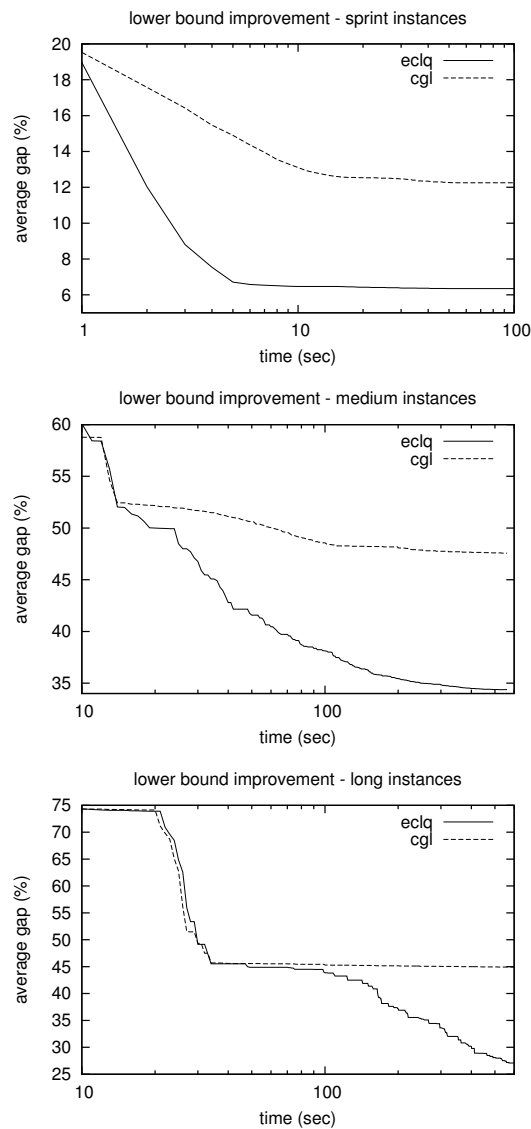
Results in Table 3 indicate that although there are large instances which are easy for the standalone solver, so that optimality was proven in less than 10 minutes, there are several instances where the solver alone (columns  $s = \emptyset$ ) could not reach *any* feasible solution in one hour of processing time, even with the activation of the heuristic emphasis. Even though entering one initial solution (columns  $s = gr(\cdot)$ ) solves the feasibility problem, the final solution quality after one hour of processing time is still far from acceptable, with gaps of about 70% appearing in some cases. These results show that in spite of the progresses in generic MIP solvers, in many cases of real world applications the hybridization of these solvers with methods which consider problem specific information is still very important and in many cases absolutely necessary.

## 6.2 Cutting planes

The objective of the separation procedure is to speed up the improvement of the lower bound and consequently to prove the optimality faster. In the first experiment we ran several rounds of cut separation using our proposed clique separation procedure, named here as **ec1q** and the clique separation routine included in the COIN-OR Cut Generation Library, denoted here as **cg1**, restricted by the following time limit: 100 seconds for sprint instances and 600 seconds for larger instances. To measure the improvements we computed for each instance and time instant the relative distance (gap) to best upper bound: the optimal solution or best known solution. Let a given lower bound  $lb$  and an upper bound  $ub$  the gap is  $\frac{ub-lb}{ub} \times 100$ . In Figure 5 the evolution of the average gap for groups of instances in time is presented. It can be observed that the inclusion of our lifted inequalities allows a faster reduction in the gap. Furthermore, **ec1q** cuts still make progress when **cg1** cuts cannot perform any significant change in the dual limit. The separation of odd holes showed no surprises for us: as previous works say, they have no significant impact for dual bound improvement. One reason for this is that most violated odd-holes found are  $k_3$ , so that the clique separation routines already finds it. Violated odd holes of size 5 or more are scarce in the root node relaxation. Nevertheless, these are safe cuts (i.e. they do not depend on rounding numbers computed with limited floating point accuracy) which can be instantly separated, so they are worth keeping in branch-and-cut procedure even if they have a marginal contribution observed in the root node relaxation.

Instance	CPLEX12.1 default settings												CPLEX12.1 heuristic emphasis															
	10min / s = 0				1hour / s = 0				1hour / s = gr(.)				10min / s = gr(.)				1hour / s = 0				1hour / s = gr(.)							
	LB	UB	gap		LB	UB	gap		LB	UB	gap		LB	UB	gap		LB	UB	gap		LB	UB	gap		LB	UB	gap	
long	01	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0	197	197	0.0
	02	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0	219	219	0.0
	03	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0
	04	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0
	05	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0	284	284	0.0
hidden	01	319	∞	∞	319	487	34.5	337	615	45.2	334	374	10.7	319	∞	∞	319	487	34.5	337	602	44.0	334	380	12.1			
	02	81	∞	∞	81	111	27.0	81	153	47.1	86	92	6.5	81	∞	∞	81	111	27.0	81	143	43.4	86	93	7.5			
	03	17	∞	∞	18	61	71.0	17	∞	∞	25	55	55.2	17	∞	∞	18	61	71.0	26	∞	∞	26	51	49.9			
	04	14	∞	∞	14	44	68.8	14	∞	∞	19	44	56.1	14	∞	∞	14	44	68.8	19	∞	∞	14	42	67.2			
	05	36	∞	∞	36	130	72.7	40	∞	∞	41	41	0.0	36	∞	∞	36	130	72.7	40	∞	∞	40	46	14.1			
late	01	212	∞	∞	204	385	46.9	231	∞	∞	232	237	2.1	212	∞	∞	204	385	46.9	231	∞	∞	233	267	12.7			
	02	214	∞	∞	208	409	49.2	229	282	18.8	229	229	0.0	214	∞	∞	208	409	49.2	229	517	55.7	229	229	0.0			
	03	213	∞	∞	212	391	45.8	218	250	12.8	218	221	1.4	213	∞	∞	212	391	45.8	216	295	26.8	218	225	3.1			
	04	196	∞	∞	197	310	36.5	213	∞	∞	213	230	7.4	196	∞	∞	197	310	36.5	213	∞	∞	212	234	9.3			
	05	79	635	87.5	79	270	70.9	79	542	85.4	80	229	65.3	79	635	87.5	79	270	70.9	79	545	85.5	80	269	70.4			
(m / a / s.d.)	(∞ / 65.8 / 48.3)	(72.7 / 34.9 / 29.0)	(∞ / 47.3 / 45.1)	(65.3 / 13.6 / 23.7)	(∞ / 65.8, 48.3)	(∞ / 65.8, 48.3)	(72.7 / 34.9 / 29.0)	(∞ / 50.4 / 43.8)	(70.4 / 16.4 / 24.8)																			
medium	01	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0
	02	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0	240	240	0.0
	03	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0	236	236	0.0
	04	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0	237	237	0.0
	05	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0	303	303	0.0
hidden	01	60	1093	94.5	59	314	81.3	62	287	78.3	61	227	73.0	60	1093	94.5	59	314	81.3	66	1060	93.8	64	220	71.1			
	02	183	∞	∞	180	352	48.9	188	312	39.6	188	258	27.0	183	∞	∞	180	352	48.9	189	291	35.2	188	291	35.3			
	03	22	449	95.0	22	93	76.0	23	93	74.9	24	93	74.7	22	449	95.0	22	93	75.99	24	60	60.3	24	49	51.9			
	04	58	∞	∞	59	136	57.0	60	97	38.6	63	96	34.8	58	∞	∞	59	136	56.99	62	96	35.8	62	109	42.8			
	05	56	∞	∞	56	233	75.9	84	488	82.9	78	233	66.7	56	∞	∞	56	233	75.89	84	427	80.4	82	205	60.0			
late	01	148	171	13.5	145	215	32.5	152	157	3.1	152	157	2.9	149	166	10.5	147	212	30.77	151	158	4.4	151	157	3.5			
	02	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0
	03	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0	29	29	0.0
	04	33	35	4.4	34	37	9.4	34	35	2.9	35	35	0.0	35	35	0.0	35	37	9.8	35	35	0.0	35	35	0.0	35	35	0.0
	05	104	114	9.2	103	120	13.9	106	107	1.0	107	107	0.0	104	123	15.8	103	216	52.55	107	107	0.0	107	107	0.0	107	107	0.0
(m / a / s.d.)	(∞ / 34.4 / 46.6)	(81.3 / 26.3 / 32.4)	(82.9 / 21.4 / 32.5)	(74.7 / 18.6 / 29.4)	(∞ / 34.4 / 46.7)	(∞ / 34.4 / 46.7)	(81.3 / 28.8 / 32.9)	(93.8 / 20.7 / 32.8)	(71.1 / 17.6 / 26.4)																			

Table 3 Results of the standalone commercial solver CPLEX



**Fig. 5** Dual bound improvement for COIN-OR built in cut generator (cgl) and the proposed cut cut separation procedure (eclq)

After a series of experiments with all cuts available in the COIN-OR Cut Generation Library (CGL), we found out that the following generic cutting planes were also useful in improving the dual bound: Mixed Integer Gomory[5], Two-Step Mixed Integer Rounding[20], RedSplit[1] and the Zero-Half ( $0/\frac{1}{2}$ )[2] cuts. Zero-Half cuts are not available yet in the latest formal CGL release,

		Window	Zero-Half	Gomory	RedSplit	TwoMIR
sprint	max.	50.1	51.5	93.7	52.9	52.8
	min.	1.9	3.7	3.7	3.7	3.7
	av.	19.7	21.6	26.2	22.3	22.6
	std.dev	15.2	15.3	23.1	15.8	15.9
medium	max.	100.0	100.0	100.0	100.0	100.0
	min.	0.0	0.0	0.0	0.0	0.0
	av.	50.1	51.2	51.9	50.7	41.1
	std.dev	46.6	46.3	47.7	46.7	47.9
long	max.	100.0	100.0	58.9	100.0	36.5
	min.	0.0	0.0	0.0	0.0	0.0
	av.	37.5	38.7	14.5	38.0	5.9
	std.dev	39.1	39.3	17.8	40.3	9.3

**Table 4** Contribution of different cuts to improve root node relaxation lower bound

but authors gently offered the code for our experiments. The contribution of all these additional cuts applied jointly with our clique cuts for improving the lower bound for each group of instances is shown in Table 4. Considering the linear programming relaxation limit (lp) and the lower bound obtained at the end of the root node cut application in CBC (lb) we computed for each instance the improvement:  $\min\{\frac{lb-lp}{lp+\epsilon} \times 100, 100\}$ , where  $\epsilon$  is a small constant to avoid division by zero. A summary of these results is presented in Table 4. As it can be seen, although gomory cuts are of crucial importance for small instances, its relevance diminishes in larger instances. The reason is that these cuts tend to produce very dense constraints for large linear programs and are probably discarded by the branch and cut code of CBC in large instances. The proposed Window cuts, on the other hand, appear to be more important in larger instances.

### 6.3 Mathematical Programming Heuristic

The *MPH* uses CPLEX within the algorithm to solve the subproblems. Parallel mode was disabled, so both the heuristic and CPLEX ran sequentially. All 60 instances from the INRC [28] were tested with the parameter  $m$  assuming values from 1 to 9. The results are reported in Table 5.

In this table, the column BKS shows the best known solutions, including the ones found in this work (marked with a  $\otimes$ ). The following columns show the best results found in the literature (PUB), the best results by CPLEX, the best results obtained by the *MPH* with  $m$  in the range of 1 to 9 and the best results obtained in each one of these. For each result, the table reports the best upper bound (ub) obtained and the gap between this upper bound ( $ub$ ) and the best known solution (BKS):  $\frac{ub-BKS}{ub} \times 100$ .

The results for “sprint” instances weren’t reported in Table 5 because, for all of them, the heuristic was able to find the optimal solution within 3 minutes with  $m$  starting with values in the range [1,9].

Some comments about the results shown in Tables 5:



Instance	BKS	PUB		CPLEX (m = 14)		best MPH (m ∈ [1, 9])		MPH results with different values for m																					
		UB	gap	UB	gap	UB	gap	UB	gap	m = 1		m = 2		m = 3		m = 4		m = 5		m = 6		m = 7		m = 8		m = 9			
										UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap	UB	gap
long	01	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0	197	0.0		
	02	219	0.0	219	0.0	219	0.0	219	0.0	220	0.5	220	0.5	220	0.5	220	0.5	220	0.5	220	0.5	219	0.0	219	0.0	219	0.0		
	03	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0
	04	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0
	05	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0	284	0.0
hidden	01	⊗ 346	363	4.7	487	29.0	346	0.0	347	0.3	348	0.6	346	0.0	346	0.0	346	0.0	359	3.6	346	0.0	355	2.5	534	35.2	506	31.6	
	02	⊗ 89	90	1.1	111	19.8	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	89	0.0	
	03	38	38	0.0	61	37.7	39	2.6	39	2.6	40	5.0	41	7.3	41	7.3	42	9.5	41	7.3	42	9.5	130	70.8	119	68.1	118	67.8	
	04	22	22	0.0	44	50.0	22	0.0	22	0.0	22	0.0	22	0.0	22	0.0	22	0.0	22	0.0	22	0.0	37	40.5	41	46.3	112	80.4	
	05	41	41	0.0	130	68.5	41	0.0	43	4.7	45	8.9	42	2.4	43	4.7	43	4.7	43	4.7	43	4.7	44	0.0	49	16.3	174	76.4	
late	01	235	235	0.0	385	39.0	239	1.7	239	1.7	241	2.5	239	1.7	241	2.5	242	2.9	242	2.9	244	3.7	244	3.7	244	3.7	244	3.7	
	02	229	229	0.0	409	44.0	235	2.6	235	2.6	235	2.6	237	3.4	235	2.6	239	4.2	242	5.4	242	5.4	235	2.6	242	5.4	243	5.8	
	03	220	220	0.0	391	43.7	220	0.0	221	0.5	225	2.2	221	0.5	220	0.0	233	5.6	239	7.9	233	5.6	233	5.6	220	0.0	380	42.1	
	04	221	221	0.0	310	28.7	222	0.5	225	1.8	231	4.3	228	3.1	224	1.3	229	3.5	227	2.6	222	2.6	222	2.6	222	2.6	228	3.1	
	05	83	83	0.0	270	69.3	83	0.0	86	3.5	83	0.0	83	0.0	83	0.0	83	0.0	83	0.0	83	0.0	92	9.8	159	47.8	240	65.4	
gap (avg./std. deviation)				(0.4/1.2)	(28.6/24.7)	(0.5/0.9)		(1.2/1.5)	(1.8/2.6)	(1.1/1.6)	(1.3/2.2)	(2.1/2.5)	(2.3/3.3)	(9.1/20.0)	(15.1/22.7)	(26.0/31.6)													
medium	01	240	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	
	02	240	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	240	0.0	
	03	236	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	236	0.0	
	04	237	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	237	0.0	
	05	303	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303	0.0	303
early	01	⊗ 111	130	14.6	314	64.6	111	0.0	116	4.3	111	0.0	118	5.9	118	5.9	116	4.3	117	5.1	118	5.9	118	5.9	120	7.5	307	63.8	
	02	221	221	0.0	352	37.2	221	0.0	221	0.0	223	0.9	224	1.3	227	2.6	223	0.9	235	6.0	221	0.0	226	2.2	256	13.7			
	03	⊗ 34	36	5.6	93	63.4	34	0.0	34	0.0	40	15.0	41	17.1	38	10.5	36	5.6	38	10.5	36	5.6	36	5.6	36	5.6	41	17.1	
	04	⊗ 78	80	2.5	136	42.6	80	2.5	80	2.5	80	2.5	84	7.1	81	3.7	82	4.8	81	3.7	81	3.7	81	3.7	80	2.5	147	46.9	
	05	⊗ 119	122	2.5	233	48.9	119	0.0	121	1.7	120	0.8	123	3.3	121	1.7	120	0.8	119	0.0	127	6.3	121	1.7	232	48.7			
late	01	⊗ 157	158	0.6	215	27.0	157	0.0	161	2.5	160	1.9	160	1.9	161	2.5	161	2.5	157	0.0	159	1.3	161	2.5	164	4.3			
	02	18	18	0.0	18	0.0	19	5.3	19	5.3	19	5.3	22	18.2	23	21.7	24	25.0	20	10.0	20	10.0	20	10.0	24	25.0			
	03	29	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0	29	0.0			
	04	35	35	0.0	37	5.4	35	0.0	35	0.0	36	2.8	35	0.0	36	2.8	36	2.8	36	2.8	37	5.4	37	5.4	35	0.0	36	2.8	
	05	107	107	0.0	120	10.8	108	0.9	108	0.9	114	6.1	113	5.3	118	9.3	114	6.1	114	6.1	114	6.1	119	10.1	109	1.8	112	4.5	
gap (avg./std. deviation)				(1.7/3.9)	(20.0/24.9)	(0.6/1.5)		(1.1/1.7)	(2.4/4.0)	(4.0/6.0)	(4.1/6.0)	(3.5/6.4)	(3.1/3.8)	(3.2/3.7)	(2.2/3.1)	(15.1/21.3)													

Table 5 Results of the Mathematical Programming Heuristic

- The impact of the parameter  $m$  is very perceptible and the best average result were obtained when  $m = 1$ .
- The upper bounds provided by the MPH are certainly very good, especially when  $m = 1$  in the first iteration. Considering all the runs ( $m$  starting from 1 to 9), the heuristic was able to outperform the best solution on the literature for 6 instances. Considering the “sprint” instances, the gap was greater than 0 only for 7 out of 60 instances. The average gap from the best known solution was also very low: 0.5% for “long” instances and 0.6% for “medium” instances, with a maximum gap of 5.3% considering all the instances, which is specifically related to one unit on the objective function value of instance `medium_late02`.
- Since the MPH was able to robustly find good solutions in up to 10 minutes of sequential processor time, we decided not to report results of longer runs. Such decision is easy to be explained, since running the heuristic for longer times may result in solving the original problem (see section 5.3), which can take a very long time for some instances.
- The comparison with the upper bounds of the method and the best results from literature should take into account the difference of order of magnitude in the running times for “long” instances. While MPH running times are limited to 600 seconds, Valouxis et al. [44] report times of up to 36,000 seconds on these instances. The comparison is more fair when considering the “medium” instances, since running times are similar.

Figure 6 shows the improvement in time with regard to the upper bound by MPH using different values for the initial  $m$ . From this figure we can conclude that for values lower than 7 for the initial  $m$ , the MPH is capable to produce good solutions in the early stages of the search. The figure also shows that, as the value of the initial  $m$  becomes larger, the heuristic takes more time to generate better solutions. Such additional time is not worthwhile, since the final solution is still worse than the ones produced by MPH with smaller  $m$  in the first iteration.

#### 6.4 Best Results

The best results obtained from all experiments are presented in Table 6. Table cells marked with  $\otimes$  indicate some improvement over the best known solution as reported in the INRC site at time of the writing of this work. It is important to remember that this site has received updates in the years following the competition, so the previous best known solutions (column PUB) were already very hard to find. Most instances were solved to optimality and the harder among the unsolved instances is `medium_hidden05` where the lower bound distance is now at 23.7%.

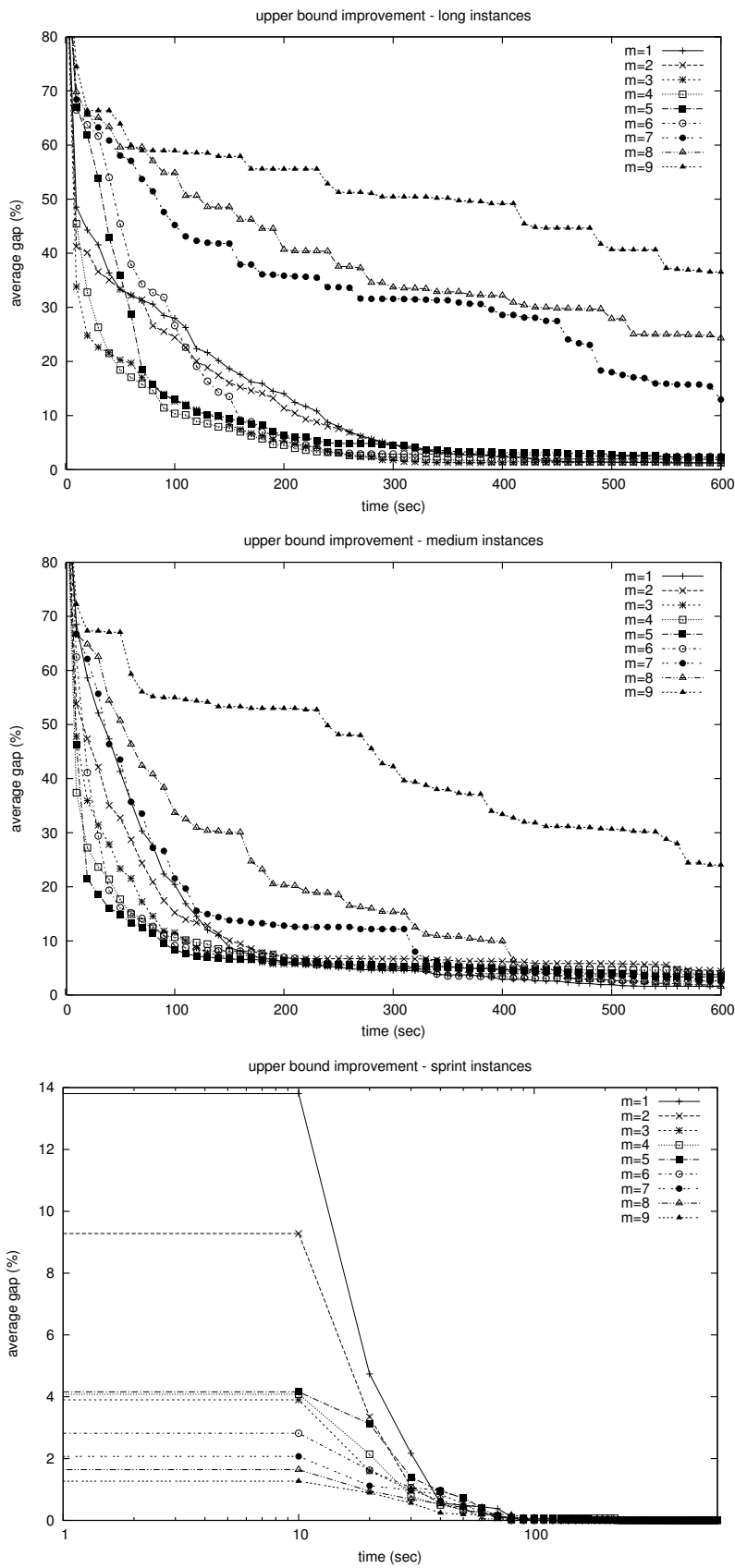


Fig. 6 Improvement in time graphic for MPH with different values for the initial  $m$ .

Instance			LB	PUB	UB	GAP
long	early	01	197.0	197	197	0.0
		02	219.0	219	219	0.0
		03	240.0	240	240	0.0
		04	303.0	303	303	0.0
		05	284.0	284	284	0.0
	hidden	01	341.0	363	Ⓢ <b>346</b>	1.4
		02	86.0	90	Ⓢ <b>89</b>	3.4
		03	35.3	38	38	7.1
		04	19.0	22	22	13.8
		05	41.0	41	41	0.0
	late	01	232.0	235	235	1.3
		02	229.0	229	229	0.0
		03	219.0	220	220	0.5
		04	214.6	221	<b>222</b>	3.3
		05	83.0	83	83	0.0
medium	early	01	240.0	240	240	0.0
		02	240.0	240	240	0.0
		03	236.0	236	236	0.0
		04	237.0	237	237	0.0
		05	303.0	303	303	0.0
	hidden	01	87.2	130	Ⓢ <b>111</b>	21.5
		02	196.6	221	221	11.1
		03	27.7	36	Ⓢ <b>34</b>	18.5
		04	72.8	80	Ⓢ <b>78</b>	6.7
		05	90.8	122	Ⓢ <b>119</b>	23.7
	late	01	155.7	158	Ⓢ <b>157</b>	0.8
		02	18.0	18	18	0.0
		03	29.0	29	29	0.0
		04	35.0	35	35	0.0
		05	107.0	107	107	0.0

Instance			LB	PUB	UB	GAP
sprint	early	01	56.0	56	56	0.0
		02	58.0	58	58	0.0
		03	51.0	51	51	0.0
		04	59.0	59	59	0.0
		05	58.0	58	58	0.0
		06	54.0	54	54	0.0
		07	56.0	56	56	0.0
		08	56.0	56	56	0.0
		09	55.0	55	55	0.0
		10	52.0	52	52	0.0
	late	01	32.0	33	Ⓢ <b>32</b>	0.0
		02	32.0	32	32	0.0
		03	62.0	62	<b>62</b>	0.0
		04	66.0	67	Ⓢ <b>66</b>	0.0
		05	59.0	59	59	0.0
06		130.0	134	Ⓢ <b>130</b>	0.0	
07		153.0	153	153	0.0	
08		204.0	209	Ⓢ <b>204</b>	0.0	
09		338.0	338	338	0.0	
10		306.0	306	306	0.0	
hidden	01	37.0	37	37	0.0	
	02	42.0	42	42	0.0	
	03	48.0	48	48	0.0	
	04	73.0	75	Ⓢ <b>73</b>	0.0	
	05	44.0	44	44	0.0	
	06	42.0	42	42	0.0	
	07	42.0	42	42	0.0	
	08	17.0	17	17	0.0	
	09	17.0	17	17	0.0	
	10	43.0	43	43	0.0	

**Table 6** previous upper bound (PUB), updated lower (LB) and upper (UB) bounds

## 7 Conclusions and Future Works

This work presented Integer Programming techniques for the Nurse Rostering Problem. Although there are several detailed results published in the literature for heuristics evaluated using the INRC instance set, we believe that this is the first work which also devotes a considerable attention to the computational production of strong dual bounds obtained from the linear programming relaxation. These bounds allowed us to prove the optimality for many instances, and its importance is not restricted to exact methods: improved dual bounds allows a more effective pruning of nodes in the search tree, which is useful to speedup MIP heuristic search in large neighborhoods, as the one presented in this work. The large number of experiments made using the open source CBC solver allowed us to spot existing previously unknown CBC bugs. These bugs were subsequently fixed by CBC developers. We proposed and implemented a much better clique cut generator for CBC, showing that it can produce better dual bounds in the early stages of the search. This code will also be released as open source. This work was not enough to make CBC competitive with the best commercial solvers when solving INRC instances, since to develop a competitive MIP heuristic we still needed to rely on CPLEX. Nevertheless, we believe that this is a step towards validating and improving this important open source integer programming solver.

The proposed MIP heuristic, built upon the presented formulation and evaluated with the state-of-art CPLEX solver, improved several best known solutions, requiring very short computing times and still being competitive with the best heuristics for this problem. We believe that the new improved

primal and dual bounds will allow a more precise evaluation of the quality of available heuristics for this problem.

## 8 Acknowledgements

The authors would like to thank FAPEMIG (grant APQ-01779-10) and CNPq (grant 480388/2010-5) for supporting the development of this research and the anonymous reviewers of this paper for the detailed suggestions and corrections.

## References

1. Andersen, K., Cornuejols, G., Y., L.: Reduce-and-split cuts: Improving the performance of mixed integer gomory cuts. *Management Science* **51**, 1720–1732 (2005)
2. Andreello, G., Caprara, A., Fischetti, M.: Embedding cuts in a branch and cut framework: a computational study with 0,1/2-cuts. *INFORMS Journal on Computing* **19**(2), 229–238 (2007)
3. Atamtürk, A., Nemhauser, G.L., Savelsbergh, M.W.P.: Conflict graphs in solving integer programming problems. *European Journal of Operational Research* **121**, 40–55 (2000)
4. Avella, P., Vasil'ev, I.: A computational study of a cutting plane algorithm for university course timetabling. *Journal of Scheduling* **8**, 497–514 (2005)
5. Balas, E., Ceria, S., Cornuejols, G., Natra, N.: Gomory cuts revisited. *Operations Research Letters* **19**, 1–10 (1996)
6. Bilgin, B., Demeester, P., Mısıır, M., Vancroonenburg, W., Berghe, G., Wauters, T.: A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. In: the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT10)-the Nurse Rostering Competition (2010)
7. Borndorfer, R.: Aspects of set packing, partitioning, and covering. Ph.D. thesis, Faculty of Mathematics at Technical University of Berlin (1998)
8. Boyd, E.: Fenchel cutting planes for integer programming. *Operations Research* **42**, 53–64 (1992)
9. Boyd, E.: Solving 0/1 integer programs with enumeration cutting planes. *Annals of Operations Research* **50**, 61–72 (1994)
10. Brito, S., Santos, H.G.: Pivoting in the bron-kerbosch algorithm for maximum-weight clique detection (in portuguese). In: *Anais do XLIII Simpsio Brasileiro de Pesquisa Operacional* (2011)
11. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* **16**, 575–577 (1973). DOI <http://doi.acm.org/10.1145/362342.362367>. URL <http://doi.acm.org/10.1145/362342.362367>
12. Burke, E., Curtois, T.: An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition, 2010. In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling PATAT 2010* (2010)
13. Burke, E., Li, J., Qu, R.: A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research* **203**(2), 484–493 (2010)
14. Burke, E., Mareček, K., Parkes, A.J., Rudová, H.: A branch-and-cut procedure for the udine course timetabling problem. *Ann. Oper. Res.* pp. 1–17 (2011). DOI <http://dx.doi.org/10.1007/s10479-010-0828-5>. URL <http://cs.nott.ac.uk/~jxm/timetabling/patat2008-paper.pdf>
15. Burke, E.K., De Causmaecker, P., Berghe, G.V., Landeghem, H.V.: The state of the art of nurse rostering. *Journal of Scheduling* **7**, 441–499 (2004)

16. Cheang, B., Li, H., Lim, A., Rodrigues, B.: Nurse rostering problems—a bibliographic survey. *European Journal of Operational Research* **151**(3), 447–460 (2003)
17. Cornuéjols, G.: Revival of the gomory cuts in the 1990 's. *Annals of Operations Research* **149**(1), 63–66 (2007)
18. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve mip solutions. Tech. rep., ILOG (2003)
19. Danna, E., Rothberg, E., Le Pape, C.: Integrating mixed integer programming and local search: A case study on job-shop scheduling problems. In: *Proceedings CPAIOR'03* (2003)
20. Dash, S., Goycoolea, M., Gunluk, O.: Two step MIR inequalities for mixed-integer programs. *INFORMS Journal on Computing* (2009)
21. Eso, M.: Parallel branch-and-cut for set partitioning. Ph.D. thesis, Cornell University Ithaca, NY, USA (1999)
22. Fischetti, M., Lodi, A.: Local branching. *Mathematical Programming* **98**, 23–47 (2003)
23. Fischetti, M., Lodi, A.: Optimizing over the first Chvátal closure. *Mathematical Programming B* **110**(1), 3–20 (2007)
24. Forrest, J., Lougee-Heimer, R.: CBC user guide. *INFORMS Tutorials in Operations Research*. pp. 257–277 (2005). DOI 10.1287
25. Grotschel, M., Lovasz, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization*. Springer (1993)
26. Hansen, P., Mladenović, N.: Variable neighborhood search. *Computers and Operations Research* **24**(11), 1097–1100 (1997)
27. Hansen, P., Mladenović, N., Urošević, D.: Variable neighborhood search and local branching. *Comput. Oper. Res.* **33**(10), 3034–3045 (2006)
28. Haspeslagh, S., De Causmaecker, P., Stolevik, M., A., S.: First international nurse rostering competition 2010. Tech. rep., CODES, Department of Computer Science, KULeuven Campus Kortrijk. Belgium (2010)
29. Hoffman, K., Padberg, M.: Solving airline crew scheduling problems by branch-and-cut. *Management Science* **39**(6), 657–682 (1993)
30. IBM: CPLEX 12.2 User's Manual (2011)
31. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A., Heinz, S., Lodi, A., Mittelman, H., Ralphs, T., Salvagnin, D., Steffy, D., Wolter, K.: MIPLIB 2010. *Mathematical Programming Computation* **3**, 103–163 (2011). URL <http://dx.doi.org/10.1007/s12532-011-0025-9>. DOI 10.1007/s12532-011-0025-9
32. Linderoth, J.T., Ralphs, T.K.: Noncommercial software for mixed-integer linear programming. In: J. Karlof (ed.) *Integer Programming: Theory and Practice, Operations Research Series*, vol. 3 (2005)
33. Lougee-Heimer, R.: The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development* **47**(1), 57–66 (2003)
34. Martins, A.X., Souza, M.C., Souza, M.J., Toffolo, T.A.M.: GRASP with hybrid heuristic-subproblem optimization for the multi-level capacitated minimum spanning tree problem. *Journal of Heuristics* **15**, 133–151 (2009). DOI 10.1007/s10732-008-9079-x. URL <http://dl.acm.org/citation.cfm?id=1527562.1527566>
35. Méndez-Díaz, I., Zabala, P.: A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics* **156**, 159–179 (2008)
36. Mittelman, H.: Benchmarks for optimization software (2012). URL <http://plato.asu.edu/bench.html>
37. Nonobe, K.: Inrc2010: An approach using a general constraint optimization solver. *The First International Nurse Rostering Competition (INRC 2010)* (2010)
38. Nonobe, K., Ibaraki, T.: A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research* **106**(2-3), 599–623 (1998)
39. Padberg, M.: On the facial structure of set packing polyhedra. *Mathematical Programming* **5**(1), 199–215 (1973)
40. Poggi de Aragão, M., Uchoa, E.: Integer program reformulation for robust branch-and-cut-and-price. In: *Annals of Mathematical Programming in Rio*, pp. 56–61. Buzios, Brazil (2003)

41. Ralphs, T., Saltzman, M., Ladnyi, L.: The COIN-OR Open Solver Interface: Technology Overview (2004). URL <http://www.coin-or.org/Presentations/CORS2004-OSI.pdf>
42. Ralphs, T.K., Gzelsy, M.: The symphony callable library for mixed integer programming. In: B. Golden, S. Raghavan, E. Wasil, R. Sharda, S. Vo (eds.) *The Next Wave in Computing, Optimization, and Decision Technologies, Operations Research/Computer Science Interfaces Series*, vol. 29, pp. 61–76. Springer US (2005)
43. Uchoa, E., Toffolo, T.A.M., de Souza, M.C., Martins, A.X., Fukasawa, R.: Branch-and-cut and hybrid local search for the multi-level capacitated minimum spanning tree problem. *Networks* **59**(1), 148–160 (2012). DOI 10.1002/net.20485. URL <http://dx.doi.org/10.1002/net.20485>
44. Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., Housos, E.: A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research* (2012)

---

## A Survey on Workforce Scheduling and Routing Problems

J. Arturo Castillo-Salazar <sup>1</sup>  
Dario Landa-Silva  
Rong Qu

Received: date / Accepted: date

**Abstract** In the context of workforce scheduling, there are many scenarios in which personnel must carry out tasks at different locations hence requiring some form of transportation. Examples of these type of scenarios include nurses visiting patients at home, technicians carrying out repairs at customers' locations, security guards performing rounds at different premises, etc. We refer to these scenarios as *Workforce Scheduling and Routing Problems* (WSRP) as they usually involve the scheduling of personnel combined with some form of routing in order to ensure that employees arrive on time to the locations where tasks need to be performed. This kind of problems have been tackled in the literature for a number of years. This paper presents a survey which attempts to identify the common attributes of WSRP scenarios and the solution methods applied when tackling these problems. Our longer term aim is to achieve an in-depth understanding of how to model and solve workforce scheduling and routing problems and this survey represents the first step in this quest.

**Keywords** workforce scheduling, employee rostering, routing problems, mobile workforce

### 1 Introduction

In recent times, employees often need to be more flexible regarding the type of job performed and similarly, employers need to make compromises in order to retain their best employees (Eaton, 2003; Martínez-Sánchez et al, 2007). Moreover, in some cases workforce should perform tasks at different locations,

---

<sup>1</sup> The author acknowledges CONACyT for its financial support

J. Arturo Castillo-Salazar · Dario Landa-Silva · Rong Qu  
E-mail: jac@cs.nott.ac.uk · dariolandasilva@nottingham.ac.uk · rong.qu@nottingham.ac.uk  
ASAP Research Group, School of Computer Science  
University of Nottingham Jubilee Campus  
Wollanton Road, Nottingham, United Kingdom, NG8 1BB



e.g. nurses visiting patients at their home, technicians carrying out repairs at different companies, etc. Therefore, the scheduling of workforce with ‘flexible’ arrangements and ‘mobility’ is of great importance in many scenarios. Many types of personnel scheduling problems have been tackled in the literature (Baker, 1976; Miller, 1976; Golembiewski and Proehl Jr, 1978; Cheang et al, 2003; Ernst et al, 2004; Alfares, 2004). We are interested in those workforce scheduling problems in which personnel is considered *flexible* (in terms of tasks and working times) and *mobile* (travelling is required in order to do the job). By *mobility* we refer specifically to those cases in which moving from one location to another takes significant time and therefore reducing the travel time could potentially increase productivity. To some extent, this problem combines features from the general employee scheduling problem and also vehicle routing problems. The survey and discussion presented here represent the first step in our aim of formulating and tackling the problem of scheduling flexible and mobile workforce. In the rest of this paper, we refer to this as the workforce scheduling and routing problem (WSRP).

In section 2 we describe the WSRP and identify some of the main characteristics of this type of workforce scheduling problems. Section 3 outlines some workforce scheduling scenarios that have been investigated in the literature and that in our view present a case of WSRP. Examples include home care, scheduling of technicians, manpower allocation, etc. Subsection 3.6.3 is dedicated to the vehicle routing problem with time windows (VRPTW) (Desrochers et al, 1992; Kallehauge et al, 2005) since it is the base for the routing component of many of the problems discussed in this survey. Section 4 outlines different methods (optimisation, heuristics and hybrid approaches) used when tackling WSRP scenarios. Section 5 summarises our findings and outlines the next steps in our research into workforce scheduling and routing.

## 2 Workforce Scheduling and Routing Problems

### 2.1 Description of the problem

In this paper, we refer as Workforce Scheduling and Routing Problem (WSRP) to those scenarios involving the mobilisation of personnel in order to perform work related activities at different locations. In such scenarios, employees use diverse means of transportation, e.g. walking, car, public transport, bicycle, etc. Also, in these scenarios there are more than one activity to be performed in a day, e.g. nurses visiting patients at their homes to administer medication or provide treatment (Cheng and Rich, 1998), care workers aiding members of the community to perform difficult tasks (Eveborn et al, 2006), technicians carrying out repairs and installations (Cordeau et al, 2010), and security guards performing night rounds on several premises (Misir et al, 2011). The number of activities across the different locations is usually larger than the number of employees available, hence employees should travel between locations to perform the work. This results in a combination of employee scheduling and vehicle routing problems. The number of activities varies depending on the duration

of the working shift, but assuming that each activity needs to be performed at a different location, a routing problem also arises. A route is a sequence of locations that need to be visited (Raff, 1983) but we exclude problems in which workers need to move across work stations within the same factory for example. Work activities which need to be performed in a specified time (time window) require scheduling in addition to routing. Tackling WSRP scenarios could potentially involve many objectives like: reducing employees travel time, guaranteeing tasks to be performed by qualified people only, reducing the cost of hiring casual staff, ensuring contracted employees are used efficiently, etc.

We assume employees should rather spend more time doing work than travelling, particularly in settings in which travelling time is counted as working time, hence reducing travel time is valuable (Fosgerau and Engelson, 2011; Jara-Díaz, 2000). In WSRP scenarios is often beneficial that employees perform activities at customer premises more efficiently. Like in many workforce scheduling problems, the set of skills that an employee has for performing a task is of great importance (Cordeau et al, 2010). Many papers in the literature assume that the workforce is homogeneous regarding skills but in many scenarios, a diverse set of skills is the predominant environment. We should note that scenarios like the pick-up and delivery of goods (parcels) is not considered here as a WSRP because no significant ‘work’ (in terms of time) is carried out at customers’ premises. Although, one could argue that the action of delivering a parcel is a task, it does not take a significant amount of time once the worker gets to the destination. This type of pick-up and delivery problems are definitely routing problems but are not covered in our study of workforce scheduling and routing problems.

## 2.2 Main characteristics

In this subsection, we outline the main characteristics of any WSRP. Some of these characteristics are ‘obvious’ since they are in the nature of the problem while others were identified during our survey. We include the characteristics that appear the most in the literature and describe them in the subsections below. For a list of all the attributes considered and the papers included in this survey please refer to *Table 1*.

**Time windows** for performing a task/duty/job at a customer premises. It is assumed that employees can start the work as soon as they arrive to the location. Time windows can be very flexible or very tight and in accordance to contractual arrangements. In some cases, no time window is defined as employees work based on annualised hours. Also, in some cases employees can benefit from over-time payment, making compliance with the time window more of a soft constraint.

**Transportation modality** refers to employees using different means like: car, bicycle, walking or public transport. We assume that time and cost of transportation is not the same for each employee.

**Start and end locations** One location, when all employees start at the main office (Eveborn et al, 2006), up to many locations (perhaps as many as the

number of employees) assuming each employee may start from their home. In some cases the company's policy might be that employees should start their working time at the main office but then returning home directly after the last job performed.

**Skills and qualifications** act as restrictive filters on who can perform a task and there are two main cases. 1) In general, all workforce have the same ability (skills and qualifications) so anyone can perform the task, but this tends to be expensive for the organisation. 2) Workforce with diverse levels of abilities, this is common in industries such as consulting and healthcare. Matching employees' skills to the tasks assigned has been tackled for complex organisations (Cordeau et al, 2010).

**Service time** corresponds to the duration of the task and it varies depending on the employee who performs it and the type of task. Most models in the literature assume a fixed duration. If service times are long enough so that they restrict each worker to perform only one job, then the problem reduces to task allocation since every route would consider only one job per employee.

**Connected activities** refers to dependencies among the activities to be performed. *Sequential*, when one activity must be performed before/after another. Activities are said to be *simultaneous* when they happen at the same time and require two or more employees to be present. *Temporal dependencies: synchronisation, overlap, minimum difference, maximum difference, min+max difference*, as defined by Rasmussen et al (2012).

**Teaming** may be necessary due to the nature of the work to be carried out (Li et al, 2005). If members of the team remain unchanged then the team can be treated as a single person and synchronising the arrival of team members is necessary. If members of the team change frequently then skill matching according to the job is required (Cordeau et al, 2010).

**Clusterisation** may be necessary for several reasons. One is employee preferences when expecting not to travel more than a number of miles. Another reason is when companies assign employees to perform work only in certain geographical areas. Clusters may also be created just to reduce the size of the problem and solve many smaller instances.

### 3 Workforce Scheduling and Routing Problems in the Literature

In this section we describe some of the problems tackled in the literature that can be considered as a type of workforce scheduling and routing problem (WSRP). The intention is to illustrate the variety and importance of WSRP scenarios in the real-world.

#### 3.1 Home health care

Bertels and Fahle (2006) describe home health care (HHC) as visiting and nursing patients at their home. Patients preferences regarding the time of visit are respected as much as possible, as they cannot wait for the entire day.

Additionally, nurses have also *time window* limitations regarding the number of hours they work in a day or their starting and ending time. In HHC, *transportation modality* is present when nurses travel by car, public transport or even walking to visit more than one patient. The *start and end location* of nurses routes vary. They can depart from their homes or from a central health care office, and end their day once they return home or in some cases at the last patient visited. A diverse set of *skills and qualifications* is present in the set of nurses due to the large range of procedures required. Healthcare organisations often cannot afford to have nurses trained in all procedures. Then, the use of a highly qualified nurses should be restricted to tasks that demand those skills. Nursing activities vary in duration (*service time*) e.g. from a 10 min injection to a 45 min physical therapy. *Connected nursing activities* can be found when applying medicine e.g. the first dose is applied during the morning and 3 hours later another dose. Some activities require more than one nurse at the same time e.g. handling a person with epilepsy. In such cases, nurses can be *synchronised* to arrive at the location at the same time, or assign a *team of two nurses* who always performed these type of tasks. *Clusterisation* is used, by the organisation providing health care to avoid nurses having to travel too much.

Other characteristics of HHC which are not part of the main WSRP main ones include nurses preferences, shift types and other legal requirements. Also, it is desirable not to change much which nurses visit which patients. This is because patients and nurses develop a bond that is usually good to maintain. Cheng and Rich (1998) explore the use of casual nurses, i.e. those not in a contract with the organisation. Cheng's work does not consider different nurses' skills and qualifications but instead, proposes a matching method in which a pairing patient-nurse is either feasible or not for some reason. The objective in Cheng's work is to reduce the amount of overtime and part-time work employed.

### 3.2 Home care

Home care (also called domiciliary care) refers to the provision of community care service by local authorities to their constituents (Akjiratikar et al, 2007). The aim is to schedule care workers across a region in order to provide care tasks within a *time window* while reducing travel time. This problem is related to the home health care problem described by Bertels and Fahle (2006) and Cheng and Rich (1998). The difference is that home health care involves helping people for a relatively short period of time to recover after hospitalisation. Home care however usually refers to helping elderly and/or disabled people to perform their daily activities such as shopping, bathing, cleaning, cooking, etc. (Eveborn et al, 2009). Once a person starts receiving home care support it is likely to remain receiving such care for a long time. Home carers usually *start* travelling from their homes to deliver support at their predefined destinations using their own transport arrangements (mixed *transportation modality*) and *return home at the end of the day*. In some cases reported in the literature, care workers do not start from their home but from a *home care office* as last

minute changes to their schedules are possible and need to be agreed before starting the working day (Eveborn et al, 2009). In some cases, travel time is considered as work hours and hence the objective is to reduce the time used not providing care. Some assumptions are made such as given travel speed for a carer and travel distances to be euclidean. In other cases like the work by Dohn et al (2008), the objective is to maximise the quality level of care service provided. Reducing cost, although important, is not the main objective. Dohn et al (2008) study the problem as a variant of the VRP with time windows. Although not as much as in HHC, there are some *skills and qualification* required in home care when caring for others e.g. health and safety, handling people with dislexia, etc. *Service time* is standardised and it only varies due to the experience of the carer or difficulties with the person receiving care. *Connected activities* also exist in home care e.g. taking a shower before doing groceries. *Teaming* is not present since carers tend to be *synchronised* to perform difficult tasks e.g. assisting a heavy person. *Clusterisation* is based on municipalities borders to clearly defined which authority is responsible for part of the community e.g. council, borough, district etc.

Additional features of home care are: *prioritising visits*. Usually, there is not enough personnel to perform all the visits in a single day. Therefore, visits are rescheduled or even canceled in the worst case. Deciding who does not receive a visit is part of the problem. For example, it is more important to assist someone with his diabetes medication than to help another person doing groceries. The *shift patterns* are either given by contracts or expressed as preference by carers. Many organisations emphasise respecting *carers' preferences* to increase staff retention. Also, *tolerance on time windows* to perform care vary, e.g. critical medical activities having 5 minutes tolerance while support activities having 15 minutes to 2 hours tolerance.

### 3.3 Scheduling technicians

Some telecommunications companies require scheduling employees to perform a series of installation and maintenance jobs, e.g. Cordeau et al (2010). In the literature, this problem is referred to as technician and task scheduling problem (TTSP). In this sector, commitments on time to perform the jobs are enforced, resulting in strict *time windows*. Due to the equipment technicians carry, it is common to use *company vehicles* to travel from one customer location to the next one. Technicians *start and end locations* are the company premises, although in some cases technicians are allowed to take companies' cars home if the first job of the following day is at a location closer than the company's location. Technicians, depending on the sector, often are highly *skilled*. Nevertheless, their skills are related to their experience and training, as a result companies have levels of seniority among their workforce e.g. junior, senior, etc. Those seniority levels to some extent help estimating the *service time* required to complete the job e.g. although both junior and senior fibre optics technicians could recalibrate a connection, the later one does the job faster. Activities tend to be independent from each other with in the same

day, but in a wider time frame there are some *connection* between them. In this scenario, *teams* are often formed with the aim of having a balance set of personnel with as many skills as possible. *Teaming* also helps technicians to learn from each other, hence improving their performance. Companies with many branches across different regions use *clusterisation* to assign jobs to each branch when the scheduling is done centrally for all branches.

### 3.4 Security personnel routing and rostering

In this problem, round of visits are performed by security personnel to several customer premises distributed at different locations over a 24 hours period (Misir et al, 2011). Many organisations outsource security guards duties only for when premises are closed while in other cases, security is outsourced at all times. Round visits must be performed at the contracted time often given as a *time window*. Security personnel often uses a combination of *private vehicles* to go from one location to the next and *walking* once they get to the facility but require to check several buildings. Security guards *start and return* to their own homes. In this scenario, the author mentions 16 types of *skills* that the company records among its workforce and some visits require enforcing those skills. The duration (*service time*) of each visit can vary but it should be between a time framework in which the visit must finish. Visits are independent from each other. Customers are divided into regions (*clusterisation*), so that security guards living nearby are assigned to each region reducing travelling time. In this industry, contract terms vary considerably and this originates many different constraints being added to the problem. Although not mentioned in the scenario, it is not unreasonable to think that teams of two or more guards can be used.

### 3.5 Manpower allocation

Manpower allocation (Lim et al, 2004) refers to assigning servicemen to a set of customer locations to perform service activities. The objectives in this problem are to minimise the number of servicemen used, minimise the total travel distance, minimise the waiting time at service points, maximise the number of tasks assigned, etc. The manpower allocation cases when employees have to perform tasks at different locations and hence requiring transportation can qualify as a WSRP. Manpower allocation with *time windows* is particularly relevant since customers explicitly defined when the workforce is required. There is no mention of *transportation modality* so we assume all workforce uses the same type of transport. Every serviceman *starts and finishes* his working day at the control centre. *Skills* among the workforce are assumed to be the same, making no difference on who performs the service. Nevertheless there are restrictions on the number of hours each employee can work. Waiting time, the time that servicemen have to wait at a customer location before the start of the time window, is included within the *service time* making it vary accordingly. Li et al (2005) add job *teaming* constraints, a team is assembled

at every location and work cannot start unless all members of the team have arrived. More recently, a variation of the manpower allocation problem was used in the context of scheduling teams to do ground handling tasks in major airports (Dohn et al, 2009). In the work by Li et al (2005) teams are set at the beginning and do not change over the working day. Additional characteristics include teams having mandatory breaks within certain time windows, hence breaks are treated as just another visit.

### 3.6 Vehicle routing problem with time windows

The routing part in many of the problems considered here as examples of WSRP are based on the vehicle routing problem with time windows (VRPTW). In this problem the main objective is to minimise the total travel distance by a set of vehicles when performing visits to several customers spread across many locations. Every customer must be visited once by one vehicle. Each customer specifies a *time window* when the visit should take place. The delivery vehicle must arrive to the location within that specified time window. If the vehicle arrives before the time window starts, it must wait until the time window opens to perform the delivery (Desrochers et al, 1992; Kallehauge et al, 2005). Extensions of the VRPTW include other features such as multiple trips, multiple depots, capacity constraints, etc.

#### 3.6.1 VRPTW with multiple depots and waiting costs

Here, the fleet of vehicles is distributed across multiple depots allowing vehicles to return to the closest depot once all the deliveries by that vehicle have been completed. This VRPTW variant (Desaulniers et al, 1998) is relevant to our study because its formulation is applicable to workforce scheduling and routing. Many papers in the literature dealing with WSRP scenarios use this VRPTW variant and associate every employee's *starting and ending point* to a depot. It is also possible for every employee to start at the same location (depot) but then each employee to end their working day at a different location (home).

#### 3.6.2 Vehicle routing problems with multi-trips

This variant extends the classical vehicle routing problem to include multiple trips (Brandão and Mercer, 1998). It is important because it addresses the fact that an employee could perform more than one trip on a day to visit the same location. A trip in this context is a series of jobs before going back to the depot. In WSRP scenarios, an employee is assumed to have a mean of transportation either from the company or personal. Sometimes the employee might need to go back to the main office (depot) to replenish resources. The type of vehicles that can access a particular customer's location might be restricted as pointed by Brandão and Mercer (1997). Vehicles have different capacities which can be associated to model an heterogeneous workforce. Vehicles can also be hired for some time which is associated to hiring casual staff.

### 3.6.3 Synchronisation constraints in routing

Synchronisation, a type of *connected activity*, among workers when executing their tasks can be modelled in the same way as when vehicles need to arrive at the same customer location and at the same time (Bredström and Rönnqvist, 2007). Precedence constraints are another characteristic related to synchronisation (Bredström and Rönnqvist, 2008). Assuming a client can be visited more than once per day, it could be that the order of the visits matter. For example, before technicians install a satellite TV, it is important that the antenna is calibrated and then a demodulator set. These activities could be performed by different people at different times but the order matters and must be respected.

## 4 Solution Methods

In this section we summarise the range of solution approaches that have been used to tackle WSRP in the literature. We present them in three categories: optimisation techniques, heuristic algorithms and hybrid approaches. The purpose of this section is to identify the methods that have been applied to tackle different variants and components of WSRP scenarios. In *Table 1* row 44 associates each surveyed source with a domain problem mentioned in *Section 3* and row 45 presents the main technique used for its solution.

### 4.1 Optimisation techniques

*Begur et al (1997)* applied a mixed integer programming model combined with the nearest neighbor heuristic (Rosenkrantz et al, 1977) to solve a weekly nurse scheduling problem.

*De Angelis (1998)* used linear programming with clustering techniques. The scenario is split in two parts, a local one which addresses resource allocation within each district, and a global one focusing on all districts.

*Desaulniers et al (1998)* used an integer non-linear multi-commodity network flow model with time variables and solved it using column generation (Desrosiers and Lübbecke, 2005), embedded in a branch and bound algorithm. Minimum and exact waiting costs were taken into account.

*Borsani et al (2006)* used a mixed integer linear programming model based on assignment and scheduling models. The assignment model is used when new patients enter the system. The scheduling model is used to create weekly visits' plan taking as input the result of the assignment model.

*Bredström and Rönnqvist (2007)* used a set partitioning formulation. The model involves two types of variables, routing and scheduling variables. The formulation was tackled with a branch and price method (Barnhart et al, 1998).

*Dohn et al (2008)* also used a set partitioning problem with side constraints solved with branch and price. A series of shortest path problems are solved for the column generation and the master problem is solved with the set partitioning approach.



*Dohn et al (2009)* applied an integer programming formulation solved with branch and price. Dantzig-Wolfe decomposition was applied through feasible paths. The problem is divided into a generalised set covering problem and elementary shortest path.

*Kergosien et al (2009)* solved an integer programming model. After obtaining a first solution to the model, the second stage improves performance by adding cuts on the time windows. Activities requiring multiple people are split into several services.

*Rasmussen et al (2012)* also used a set partitioning problem with side constraints solved through a branch and price approach.

*Salani and Vaca (2011)* used a flow-based mixed integer program solved with branch and price.

From the above summary it can be noticed that a methodology that has been very useful to tackle WSRP is branch and price. Branch and price refers to using a branch and bound approach with column generation (Barnhart et al, 1998; Feillet, 2010). Column generation is not a recent technique it has been used successfully in other fields (Desaulniers et al, 2005). The advantage of using column generation is that the problem can be relaxed and solved with a reduced set of columns, which might not be an exhaustive enumeration of all possible routes for every employee, but at any time provides a solution if it exists. In the literature, the personnel scheduling constraints side of the problem is commonly solved by heuristics to generate columns. On the other hand, the routing component can be tackled via branching. Kallehauge et al (2005) showed that the problem formulation can be decomposed into a master problem and a pricing problem. The master problem is a set partitioning problem and the subproblem a series of shortest path problems with resources constraints (Irnich and Desaulniers, 2005; Feillet et al, 2004).

Models applied to VRPTW have also been applied to WSRP, in particular multi-commodity network flow models with time windows and capacity constraints. When using branch and price, many authors have modelled the master problem as either a set partitioning problem or as a set covering problem. There is not much difference between these two. In the first one, each customer is in one route only, whereas in the second one, more than one route could visit the same customer location.

#### 4.2 Heuristics algorithms

*Blais et al (2003)* employed a tabu search heuristic for the political district problem by Bozkaya et al (2003) which only uses two types of moves for the neighborhood single movements and swaps .

*Akjiratikarl et al (2006, 2007)* used an evolutionary approach, particle swarm optimisation (Kennedy and Eberhart, 1995), for a home care problem. An initial solution is generated using the earliest start time priority with minimum distance assignment formulation. Authors also used two local improvement procedures, a swap move to interchange activities among work-

ers and an insertion procedure to move activities from one route to another one.

*Lim et al (2004)* applied two different approaches, tabu-embedded simulated annealing and squeaky wheel optimisation (Joslin and Clements, 1999).

Three operators are used to generate local neighbourhoods shift, exchange and rearrange operators. The shift and exchange operators are similar to the insertion and swap procedures used by Akjiratikarl et al (2006).

*Itabashi et al (2006)* created a multi-agent system based on negotiation via text messages among agents representing employees. Both carers and patients have a personal device assistant. Patients submit requests to a centralised scheduling system which assigns them to a carer. The carer confirms the schedule which then is reflected in the overall schedule.

*Cordeau et al (2010)* utilised a constructive heuristic followed by an adaptive large neighbourhood search with five destroy and two repair heuristics (Ropke and Pisinger, 2006). The heuristic approach plans one day at a time in two stages. The first stage deals with team construction allocating single activities. In the second stage, the remaining activities are assigned to already defined teams.

*Misir et al (2010)* used three hyper-heuristics (Ross, 2005) with a simple learning mechanism that excludes the use of some heuristics for given phases of the search. Phases consists of a number of iterations and the best heuristics for each phase are stored in memory. Six low-level heuristics are used, two of them based on swap movements and four on removal and insertion in different routes. Three move acceptance mechanism are employed: improving or equal, improving and equal plus worsening subject to a threshold value and iteration limit, and an adaptive version by threshold changing.

*Misir et al (2011)* employed hyper-heuristics based on two different heuristics selection methods: simple random and adaptive dynamic heuristic set. First, visits are ordered and the assignment of these visits to available personnel is carried out. This is then followed by improvement heuristics that change visiting times. During the first stage, ten low-level heuristics are used. The moves to produce neighbouring solutions are swap, insertion and a move based on the idea of 'scramble' visit in the same route.

When employing heuristics including meta-heuristics and hyper-heuristics to solve the WSRP, there seems to be a tendency in the literature to use approaches based on swap (exchanges) and insertion operators. Depending on the method employ either memory is used to keep the best solutions so far or to remember which low-level heuristics are best applied in the stages of the search. Many solutions employ a constructive heuristic to generate a fast initial solution. There seems to be no solution method applied to different WSRP scenarios so far. Nevertheless, the operators used to generate neighbour solutions appear to be very similar in the different approaches.

### 4.3 Hybrid methods

*Li et al (2005)* combined the relaxation of an integer programming formulation of a network flow problem, to obtain lower bounds, with constructive heuristics embedded in a simulated annealing framework (Laarhoven and Aarts, 1987) for upper bounds. Two constructive heuristics were used, simple-append and block-insertion. Initial solutions are obtained with the first one. Neighbors are generated using block-transposition and block-reverse operators.

*Bertels and Fahle (2006)* created a hybrid model of a rostering and routing problem. It was solved using constraint programming and integer programming for sequencing visits. For improvements, simulated annealing, tabu search and combination of both, were used. The solution approach was to find a partition of jobs to nurses and to find an optimal sequencing of each partition.

*Eveborn et al (2006, 2009)* used a set partitioning model. Additionally, repeated matching to find suitable pairs of routes and workers and splitting techniques for when improvements are seek.

*Bredström and Rönnqvist (2008)* mixed, integer linear programming model and heuristics. During the first stage CPLEX is applied and in the second stage, heuristics similar to the ones by Fischetti et al (2004) are used to iteratively improve the best known solution.

*Landa-Silva et al (2011)* used a hybrid-approach combining a clustering algorithm, constructive and local search heuristics, and exact assignments based on integer programming to cluster shipments, create subgroups, build initial loads, carrier assignments and improve loads.

Most hybrid approaches try to combine the most appropriate algorithms depending on which part of the WSRP is being tackled (clustering, routing, matching skills, etc). For the routing part, it seems that the most used approaches are mathematical programming and constraint programming. This might be due to the significant advances in optimisation methods achieved recently for vehicle routing problems. Nevertheless, good heuristics methods, particularly those which provide fast initial solutions have also been employed. When matching employees to activities, the use of heuristics approaches appears to dominate.

## 5 Conclusion

A workforce scheduling and routing problem (WSRP) refers to any environment in which a skilled diverse workforce should be scheduled to perform a series of activities distributed over geographically different locations. Activities should be performed at specific times or within a given *time window*. The time window for each activity is usually determined by the customer or recipient of the job. This survey aimed to identify problems tackled in the literature that can be seen as a WSRP scenario. The problems identified in this survey include but are not limited to: *home health care, home care, scheduling*

*of technicians, security personnel routing and rostering, manpower allocation.* This survey also sought to identify the similarities between these problems in order to define the core characteristics that any WSRP would have. Those characteristics include: *time windows, transportation modalities, star and end locations, a diverse skilled workforce, activities' service time and relationship between them, and optionally, presence of teaming and clusterisation of locations* among perhaps others.

The second part of this survey sought to identify the solution methods that have been employed in the literature when tackling WSRP scenarios. Since the WSRP combines employee scheduling and vehicle routing with time windows, there seems to be a clear tendency for using exact approaches for the routing component of the problem, and using heuristics for the matching of employees to activities. Among other approaches for solving WSRP, we found: mixed integer linear programming (MILP), integer linear programming (ILP) and constraint programming (CP); using models such as set partitioning problem (SPP) and multi-commodity network flow problem; variety of meta-heuristics, tabu search (TS), particle swarm optimisation (PSO) and simulated annealing (SA).

The state of the art in WSRP, particularly in home health care and home care seems to be the use of a set partitioning problem with side constraints solved via branch and price. When clusterisation is used, it tends to be used in the initial stage of the solving procedure. The motivation for using clusterisation seems to be either to reduce the size of the problem (by creating many small problems) or to satisfy employee preferences regarding the geographic area for the location of their work. Although not present in all sources clusterisation is a current area of research, particularly in those problem which optimality could not be achieved (Rasmussen et al, 2012). In order to use the same approach for other WSRP domains, specific algorithms to generate columns based on the business rules and constraints of each domain seems to be a promising area. Our next steps are to develop such algorithms and aim to use them in a similar framework as Rasmussen et al.

As a result of this survey on workforce scheduling and routing problems, two issues seem to arise. The first one, it appears that authors who have used heuristics have not reused much previous work. We note this given the very diverse set of meta-heuristic and hyper-heuristics approaches applied. It seems not possible to identify heuristic solution methods that are popular and/or better in tackling WSRP scenarios. The second one, with the exception of the work by Akjiratikar et al it appears that no different papers have used the same data set. This is because most papers tackle different specialised variants of the problem. Nevertheless, this opens the opportunity to develop a data set that represents well various WSRP scenarios and that serves to investigate different solutions approaches.





Table 1 – continued from previous page

No	Characteristics in WSRP			
44	<p>Domains:</p> <p>HC Home care  HHC Home health care  ST Scheduling technicians  SP Security personnel  MA Manpower allocation  VRP Vehicle routing</p>			
45	<p>Solution method employed:</p> <p>O. Optimisation  1. MILP 2. ILP 3. INLP 4. SPP  5. MCNFP</p> <p>H. Heuristics  6. TS 7. PSO 8. SA 9. SWO  10. Agents based 11. LNS,  VLNS, ALNS 12. Hyper-  heuristics</p> <p>C. Hybrid</p>			
		Begur et al (1997)	HHC	O. MILP
		Brandão and Mercer (1997)	VRP	H. TS
		Brandão and Mercer (1998)	VRP	H. TS
		Cheng and Rich (1998)	HHC	C. MILP, Constructive
		De Angelis (1998)	HC	O. MILP
		Desautiers et al (1998)	VRP	O. INLP
		Blais et al (2003)	HC	H. TS
		Lim et al (2004)	MA	H. TS, SA, SWO
		Li et al (2005)	MA	C. ILP, Constructive, SA
		Akçiratikari et al (2006)	HC	H. PSO
		Bertels and Fahle (2006)	HHC	C. MILP, SA, TS
		Borsani et al (2006)	HC	O. MILP
		Eveborn et al (2006)	HC	C. ILP, SPP, Matching
		Eveborn et al (2006)	HC	H. Agents
		Itabashi et al (2006)	HC	H. PSO
		Akçiratikari et al (2007)	HC	H. PSO
		Bredström and Rönnqvist (2007)	VRP	O. ILP, SPP
		Dohn et al (2009)	MA	O. ILP, SPP
		Bredström and Rönnqvist (2008)	VRP	C. MILP, heuristics
		Dohn et al (2008)	HHC	O. ILP, SPP
		Eveborn et al (2009)	HC	C. SPP, Repeat matching
		Kergosien et al (2009)	HHC	O. ILP
		Cordeau et al (2010)	ST	H. LNS, VLNS, ALNS
		Misir et al (2010)	HC	H. Hyper-heuristics
		Rasmussen et al (2012)	HHC	O. ILP, SPP
		Misir et al (2011)	SP	H. Hyper-heuristics
		Landa-Silva et al (2011)	VRP	H. LNS, Clustering
		Salami and Vaca (2011)	VRP	O. MCNFP

MILP Mixed integer linear programming, ILP Integer linear programming, INLP Integer non-linear programming, SPP Set partitioning problem, MCNFP Multi-commodity network flow problem, TS Tabu search, PSO Particle swarm optimisation, SA Simulated annealing, SWO Squeaky wheel optimisation, (V/A)LNS (Very/Adaptive) Large neighbourhood search

## References

- Akjiratikarl C, Yenradee P, Drake PR (2006) An improved particle swarm optimization algorithm for care worker scheduling. In: Proceedings of the 7th Asia Pacific Industrial Engineering and Management Systems Conference 2006, pp 457–466
- Akjiratikarl C, Yenradee P, Drake PR (2007) Pso-based algorithm for home care worker scheduling in the uk. *Computers & Industrial Engineering* 53(4):559–583, DOI 10.1016/j.cie.2007.06.002
- Alfares HK (2004) Survey, categorization and comparison of recent tour scheduling literature. *Annals of Operations Research* 127(1–4):145–175, DOI 10.1023/B:ANOR.0000019088.98647.e2
- Baker KR (1976) Workforce allocation in cyclical scheduling problems: A survey. *Journal of the Operational Research Society* 27(1):155–167, DOI 10.1057/jors.1976.30
- Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MW, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Operations Research* 46(3):316–329, URL <http://www.jstor.org/stable/222825>
- Begur SV, Miller DM, Weaver JR (1997) An integrated spatial dss for scheduling and routing home-health-care nurses. *Interfaces* 27(4):35–48, DOI 10.1287/inte.27.4.35
- Bertels S, Fahle T (2006) A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. *Computers & Operations Research* 33(10):2866–2890, DOI 10.1016/j.cor.2005.01.015
- Blais M, Lapierre SD, Laporte G (2003) Solving a home-care districting problem in a urban setting. *Journal of the Operational Research Society* 54(11):1141–1147, DOI 10.1057/palgrave.jors.2601625
- Borsani V, Matta A, Sommaruga F, Beschi G (2006) A home care scheduling model for human resources. In: *Service Systems and Service Management, 2006 International Conference on*, vol 1, pp 449–454, DOI 10.1109/ICSSSM.2006.320504
- Bozkaya B, Erkut E, Laporte G (2003) A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research* 144(1):12–26, DOI 10.1016/S0377-2217(01)00380-0
- Brandão J, Mercer A (1997) A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research* 100(1):180–191, DOI 10.1016/S0377-2217(97)00010-6
- Brandão J, Mercer A (1998) The multi-trip vehicle routing problem. *Journal of the Operational Research Society* 49(8):799–805, URL <http://www.jstor.org/stable/3009960>
- Bredström D, Rönnqvist M (2007) A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. DOI 10.2139/ssrn.971726
- Bredström D, Rönnqvist M (2008) Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research* 191(1):19–31, DOI 10.1016/j.ejor.2007.07.033



- Cheang B, Li H, Lim A, Rodrigues B (2003) Nurse rostering problems—a bibliographic survey. *European Journal of Operational Research* 151(3):447–460, DOI 10.1016/S0377-2217(03)00021-3
- Cheng E, Rich JL (1998) A home health care routing and scheduling problem. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.1268>
- Cordeau JF, Laporte G, Pasin F, Ropke S (2010) Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling* 13(4):393–409, DOI 10.1007/s10951-010-0188-7
- De Angelis V (1998) Planning home assistance for aids patients in the city of rome, italy. *Interfaces* 28(3):75–83, URL <http://www.jstor.org/stable/25062377>
- Desaulniers G, Lavigne J, Soumis F (1998) Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research* 111(3):479–494, DOI 10.1016/S0377-2217(97)00363-9
- Desaulniers G, Desrosiers J, Solomon MM (eds) (2005) *Column Generation*. Springer
- Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. *Operations research* 40(2):342–354, DOI 10.1287/opre.40.2.342
- Desrosiers J, Lübbecke ME (2005) A primer in column generation. In: Desaulniers G, Desrosiers J, Solomon MM (eds) *Column Generation*, Springer, chap 1, pp 1–32, DOI 10.1007/0-387-25486-2\_1
- Dohn A, Rasmussen MS, Justesen T, Larsen J (2008) The home care crew scheduling problem. In: Sheibani K (ed) *Proceedings of the 1st International Conference on Applied Operational Research*, Institute for Operational Research, System Design and Financial Services, Tadbir, *Lecture Notes in Management Science*, vol 1, pp 1–8
- Dohn A, Kolind E, Clausen J (2009) The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research* 36(4):1145–1157, DOI 10.1016/j.cor.2007.12.011
- Eaton SC (2003) if you can use them: Flexibility policies, organizational commitment, and perceived performance. *Industrial Relations: A Journal of Economy and Society* 42(2):145–167, DOI 10.1111/1468-232X.00285
- Ernst A, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research* 153(1):3–27
- Eveborn P, Flisberg P, Rönnqvist M (2006) Laps care – an operational system for staff planning of home care. *European Journal of Operational Research* 171(3):962–976, DOI 10.1016/j.ejor.2005.01.011
- Eveborn P, Rönnqvist M, Einarsdóttir H, Eklund M, Lidén K, Almroth M (2009) Operations research improves quality and efficiency in home care. *Interfaces* 39(1):18–34, DOI 10.1287/inte.1080.0411
- Feillet D (2010) A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR: A Quarterly Journal of Operations Research* 8(4):407–424, DOI 10.1007/s10288-010-0130-z
- Feillet D, Dejax P, Gendreau M, Gueguen C (2004) An exact algorithm for the elementary shortest path problem with resource constraints: Appli-

- cations to some vehicle routing problems. *Networks* 44(3):216–229, DOI 10.1002/net.20033
- Fischetti M, Polo C, Scantamburlo M (2004) A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks* 44(2):61–72, DOI 10.1002/net.20017
- Fosgerau M, Engelson L (2011) The value of travel time variance. *Transportation Research Part B: Methodological* 45(1):1–8, DOI 10.1016/j.trb.2010.06.001
- Golembiewski R, Proehl Jr C (1978) A survey of the empirical literature on flexible workhours: Character and consequences of a major innovation. *Academy of Management Review* 3(4):837–853, DOI <http://www.jstor.org/stable/257938>
- Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. In: Desaulniers G, Desrosiers J, Solomon MM (eds) *Column Generation*, Springer, chap 2, pp 33–65, DOI 10.1007/0-387-25486-2\_2
- Itabashi G, Chiba M, Takahashi K, Kato Y (2006) A support system for home care service based on multi-agent system. In: *Information, Communications and Signal Processing 2005 Fifth International Conference on*, pp 1052–1056, DOI 10.1109/ICICS.2005.1689213
- Jara-Díaz S (2000) Allocation and valuation of travel time savings. In: Hensher DA, Button KJ (eds) *Handbook of transport modelling*, vol 1, Emerald Group Publishing, chap 18, pp 303–318
- Joslin DE, Clements DP (1999) Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373
- Kallehauge B, Larsen J, Madsen OBG, Solomon MM (2005) Vehicle routing problem with time windows. In: Desaulniers G, Desrosiers J, Solomon MM (eds) *Column Generation*, Springer, chap 3, pp 67–98, DOI 10.1007/0-387-25486-2\_3
- Kennedy J, Eberhart R (1995) Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*, IEEE, vol 4, pp 1942–1948, DOI 10.1109/ICNN.1995.488968
- Kergosien Y, Lente C, Billaut JC (2009) Home health care problem, an extended multiple travelling salesman problem. In: Blazewicz J, Drozdowski M, Kendall G, McCollum B (eds) *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009)*, pp 85–92, URL <http://www.mistaconference.org/2009/papers/085-092-110-P.pdf>
- Laarhoven PJM, Aarts EH (1987) *Simulated Annealing: Theory and applications*. Springer
- Landa-Silva D, Wang Y, Donovan P, Kendall G (2011) Hybrid heuristic for multi-carrier transportation plans. In: *Proceedings of the 9th Metaheuristics International Conference (MIC2011)*, pp 221–229
- Li Y, Lim A, Rodrigues B (2005) Manpower allocation with time windows and job-teaming constraints. *Naval Research Logistics* 52(4):302–311, DOI 10.1002/nav.20075
- Lim A, Rodrigues B, Song L (2004) Manpower allocation with time windows. *Journal of the Operational Research Society* 55:1178–1186, DOI

- 10.1057/palgrave.jors.2601782
- Martínez-Sánchez A, Pérez-Pérez M, De-Luis-Carnicer P, Vela-Jiménez MJ (2007) Telework, human resource flexibility and firm performance. *New Technology, Work and Employment* 22(3):208–223, DOI 10.1111/j.1468-005X.2007.00195.x
- Miller H (1976) Personnel scheduling in public systems: a survey. *Socio-economic planning sciences* 10(6):241–249
- Misir M, Verbeeck K, De Causmaecker P, Vanden Bergue G (2010) Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In: *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pp 18–23, DOI 10.1109/CEC.2010.5586348
- Misir M, Smet P, Verbeeck K, Vanden Bergue G (2011) Security personnel routing and rostering: a comparison of hyper-heuristic approaches. In: *The 3rd International Conference on Applied Operational Research, ICAOR11, Istanbul, Turkey, 2011*, pp 193–206
- Raff S (1983) Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research* 10(2):63–211, DOI 10.1016/0305-0548(83)90030-8
- Rasmussen MS, Justesen T, Dohn A, Larsen J (2012) The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research* 219(3):598–610, DOI 10.1016/j.ejor.2011.10.048
- Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40(4):455–472, DOI 10.1287/trsc.1050.0135
- Rosenkrantz DJ, Stearns RE, Lewis II PM (1977) An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing* 6(3):563–581, DOI 10.1137/0206041
- Ross P (2005) Hyper-heuristics. In: Burke EK, Kendall G (eds) *Search Methodologies*, Springer US, pp 529–556
- Salani M, Vaca I (2011) Branch and price for the vehicle routing problem with discrete split deliverables and time windows. *European Journal of Operational Research* 213(3):470–477, DOI 10.1016/j.ejor.2011.03.023

---

# A Constraint Programming Approach to the Traveling Tournament Problem with Predefined Venues

Gilles Pesant

the date of receipt and acceptance should be inserted later

**Abstract** The Traveling Tournament Problem with Predefined Venues (TTPPV) has been introduced as an abstraction of sports scheduling. Exact integer programming and heuristic approaches have been proposed so far. We investigate an exact constraint programming approach for this problem, discussing different models and search strategies. We report their respective performance on the standard set of benchmark instances and compare them to the current state of the art.

**Keywords** traveling tournament problem with predefined venues · constraint programming · sports scheduling

## 1 Introduction

The Traveling Tournament Problem with Predefined Venues (TTPPV) was introduced in [13] and consists of finding an optimal compact single round robin schedule for a sport tournament. Given a set of  $n$  teams, each team has to play once against every other team. In each game, a team is supposed to play either at home or away, however no team can play more than three consecutive times at home or away (in the rest of the paper, we will refer to this restriction as the *stretch* constraint). We seek to minimize the total distance traveled by all the teams. The main distinctive feature of this variant of the Traveling Tournament Problem (TTP) [4] is that the venue of each game is predefined, i.e. for the game in which team  $a$  plays against  $b$  it is already known whether it is going to be held at  $a$ 's home or at  $b$ 's home. A TTPPV instance is said to be balanced if the number of home games and the number of away games differ by at most one for each team; otherwise it is referred to

---

G. Pesant  
École Polytechnique de Montréal, Canada  
CIRRELT, Montreal, Canada  
E-mail: Gilles.Pesant@cirrelt.ca

as non-balanced or random. TTPPV benchmark instances were created in [13] from existing TTP instances (the Circle instances, see [4]) by adding a venue for each game. The number of teams goes from 4 to 20 and there are twenty instances (ten balanced; ten random) of each size. Instances on  $n$  teams will be denoted CIRC $n$ .

The integer programming models described in [13] solve to optimality instances with up to 8 teams but have great difficulty finding feasible solutions beyond 16-team instances. By removing the travel distance from the objective and replacing it with penalties associated with the (now relaxed) predefined venue and stretch constraints, they manage to generate feasible solutions for larger instances. More recently an iterated local search approach achieves much better solutions [2].

This paper's contribution is to show how to model and solve the TTPPV using constraint programming. This approach provides a concise formal model that can be used both in an exact or heuristic setting. It also offers the possibility to integrate side constraints easily. The rest of the paper is organized as follows: Section 2 gives a short introduction to constraint programming, Section 3 gradually describes CP models and search heuristics for the TTPPV, Section 4 presents search space exploration strategies and empirical results on instances of realistic size.

## 2 Constraint Programming

Constraint Programming (CP) is a powerful technique to solve combinatorial problems. It applies sophisticated inference to reduce the search space and a combination of variable- and value-selection heuristics to guide the exploration of that search space. The problem to solve is described through a formal model expressed using constraints from a rich set of modeling primitives. Each type of constraint encapsulates its own specialized inference algorithm.

### 2.1 CP Inference

To every variable of a CP model is associated a finite set called its *domain*: each value in that domain represents a possible value for the variable. Constraints on the variables forbid certain combinations of values. Picturing the model as a network whose vertices are the variables and whose (hyper)edges are the constraints provides insight into the basic algorithm used in CP. A vertex is labeled with the set of values in the domain of the corresponding variable and an edge is incident to those vertices representing the variables appearing in the associated constraint. Looking locally at a particular edge (constraint), the algorithm attempts to modify the label (reduce the domain) of the incident vertices (variables) by removing values which cannot be part of any solution because they would violate that individual constraint; this *local consistency* step can be performed efficiently. If every violating variable-value

pair is identified and removed, we achieve *domain consistency* which is the best we can do locally; sometimes achieving that level of consistency is computationally too costly and we will only remove values at both ends of a domain, achieving *bounds consistency* (typically for domains from a totally ordered set such as the integers).

The modification of a vertex's label triggers the inspection of all incident edges, which in turn may modify other labels. This recursive process stops when either all label modifications have been dealt with or the empty label is obtained, in which case no solution exists. The overall behavior is called *constraint propagation*.

## 2.2 CP Search

Since constraint propagation may stop with indeterminate variables (i.e. whose domain still contains several values) the solution process requires search, which can potentially take exponential time. It usually takes the form of a tree search in which branching corresponds to fixing a variable to a value in its domain, thus triggering more constraint propagation. We call *variable-selection heuristic* and *value-selection heuristic* the way one decides which variable to branch on and which value to try first, respectively. For combinatorial optimization problems, the tree search evolves into a branch-and-bound search in which branching is the same as before and lower bounds at tree nodes are obtained by various means.

## 2.3 CP for Sports Scheduling

The area of sports scheduling has already been quite successful for CP. For example it plays an important role in scheduling Major League Baseball in North America [5], it has been used to schedule the National Football League in the US [12], and has been shown to perform well for College Basketball [9]. In particular a CP model for the TTP is proposed in [10].

# 3 Modeling the TTPPV

In this section we present and evaluate empirically several models and search heuristics for the TTPPV. All tests were performed on a AMD Opteron 2.2GHz with 1GB of RAM and used the Ilog Solver 6.6 constraint programming language.

## 3.1 Initial Model

A CP model for the TTP was presented in [10] and can be partly transposed to the TTPPV. We describe an adaptation of it as our first model. For a

tournament with  $n$  teams, we will have  $n - 1$  rounds since it is built as a single round robin, as opposed to the TTP. We define *opponent* variables  $o_{ij}$ ,  $1 \leq i \leq n$ ,  $1 \leq j \leq n - 1$  to represent the opponent of team  $i$  in round  $j$ . We also define *home* variables  $h_{ij}$ ,  $1 \leq i \leq n$ ,  $1 \leq j \leq n - 1$  which are equal to 1 if team  $i$  plays at home in round  $j$  and zero otherwise.<sup>1</sup> The model is partly expressed as

$$\text{alldifferent}((o_{ij})_{1 \leq j \leq n-1}) \quad 1 \leq i \leq n \quad (1)$$

$$o_{o_{ij},j} = i \quad 1 \leq i \leq n, 1 \leq j \leq n - 1 \quad (2)$$

$$o_{ij} \in \{1, \dots, i - 1, i + 1, \dots, n\} \quad 1 \leq i \leq n, 1 \leq j \leq n - 1 \quad (3)$$

$$h_{ij} \in \{0, 1\} \quad 1 \leq i \leq n, 1 \leq j \leq n - 1 \quad (4)$$

The **alldifferent** constraint, defined on a set of variables, enforces that these variables take on distinct values and a few different consistency levels (with corresponding filtering algorithms) can be achieved for it [11]. We will use domain consistency, the strongest possible for filtering variable domains. Constraints (1) state that each team plays exactly once against every other team (single round robin): all opponents must be different and there are as many opponents as there are rounds.<sup>2</sup> By definition, Constraints (3) guarantee that each team plays exactly one game in every round (compact tournament). However one must ensure that the schedule is consistent: Constraints (2) state that in any given round, the opponent of team  $i$  has its opponent variable set to  $i$ . This is an instance of the **element** constraint, which allows array indexing by finite-domain variables and maintains domain consistency [7]. Finally Constraints (4) express the choice of the venue for the game team  $i$  plays in round  $j$ .

To take into account the predefined venue of each game, we again use **element** with the  $n \times n$  matrix  $V$  giving the venue of each game ( $V[i, k] = 1$  if the game is hosted by  $i$  and 0 if it is hosted by  $k$ ):

$$h_{ij} = V[i, o_{ij}] \quad 1 \leq i \leq n, 1 \leq j \leq n - 1 \quad (5)$$

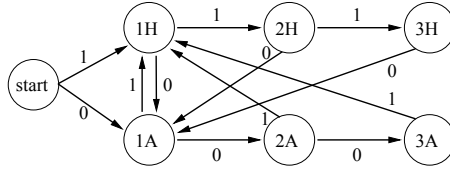
Reference [10] is not very clear on how it handles the stretch constraint and there is only mention of “some inequalities”. Since its publication there has been significant progress in modeling restrictive patterns on sequences of variables, notably the regular language membership (**regular**) constraint that takes as input an automaton describing the allowed patterns and achieves domain consistency [14]. We use it here:

$$\text{regular}((h_{ij})_{1 \leq j \leq n-1}, \mathcal{A}) \quad 1 \leq i \leq n \quad (6)$$

The small automaton  $\mathcal{A}$  for this constraint is depicted at Figure 1.

<sup>1</sup> *away* variables are also introduced in [10], but these are unnecessary since they are simply the opposite of the *home* variables.

<sup>2</sup> [10] uses the more general **cardinality** constraint since each opposing team is met twice in the TTP.



**Fig. 1** Automaton for the “maximum 3 consecutive home or away games” restriction

Expressing the cost of a schedule is a bit tedious with this model because individual travel distances depend on pairs of consecutive variables. Let  $D$  represent the  $n \times n$  travel distance matrix and variable  $d_{ij}$  the travel distance for team  $i$  to go play its game in round  $j$ . We follow [10] by considering four cases: two consecutive games for a team are either both played home, home then away, away then home, or both played away:

$$(h_{ij} = 1 \wedge h_{i,j+1} = 1) \Rightarrow d_{i,j+1} = 0 \quad 1 \leq i \leq n, 1 \leq j < n-1 \quad (7)$$

$$(h_{ij} = 1 \wedge h_{i,j+1} = 0) \Rightarrow d_{i,j+1} = D[i, o_{i,j+1}] \quad 1 \leq i \leq n, 1 \leq j < n-1 \quad (8)$$

$$(h_{ij} = 0 \wedge h_{i,j+1} = 1) \Rightarrow d_{i,j+1} = D[o_{ij}, i] \quad 1 \leq i \leq n, 1 \leq j < n-1 \quad (9)$$

$$(h_{ij} = 0 \wedge h_{i,j+1} = 0) \Rightarrow d_{i,j+1} = D[o_{ij}, o_{i,j+1}] \quad 1 \leq i \leq n, 1 \leq j < n-1 \quad (10)$$

$$(h_{i1} = 1) \Rightarrow d_{i1} = 0 \quad 1 \leq i \leq n \quad (11)$$

$$(h_{i1} = 0) \Rightarrow d_{i1} = D[i, o_{i1}] \quad 1 \leq i \leq n \quad (12)$$

$$(h_{i,n-1} = 1) \Rightarrow d_{in} = 0 \quad 1 \leq i \leq n \quad (13)$$

$$(h_{i,n-1} = 0) \Rightarrow d_{in} = D[o_{i,n-1}, i] \quad 1 \leq i \leq n \quad (14)$$

$$d_{ij} \in \{0\} \cup \{\min\{D\}, \dots, \max\{D\}\} \quad 1 \leq i \leq n, 1 \leq j \leq n \quad (15)$$

$$z = \sum_{i=1}^n \sum_{j=1}^n d_{ij} \quad (16)$$

Constraints (7)-(10) state the four cases in terms of the *home* variables and correspondingly define the travel distance variables through indexing the travel distance matrix by the *opponent* variables. Constraints (11)-(14) handle the special cases of the first and last rounds. Note that these “ $p \Rightarrow q$ ” constraints propagate in both directions: when  $p$  is satisfied then  $q$  is enforced; when  $q$  is violated then  $\neg p$  is enforced. Constraint (16) sums the individual travel distances into  $z$ , the cost objective to be minimized.

Finally we must specify a search heuristic. [10] guides search by selecting uniformly at random the next variable to branch on (value selection is not mentioned). To be more precise, a team is first selected at random, then all rounds for that team are fixed in random order, where the *home* variable is fixed before the *opponent* variable. We initially do something similar, selecting uniformly at random the next *opponent* variable among those of smallest current domain size (a proven simple generic heuristic criterion) and selecting values randomly as well. We also report on the even simpler static heuristic selecting both variables and values lexicographically.



**Table 1** Average computation time for Model (1)-(16) on the CIRC6 and CIRC8 instances

instance		time (sec)	
size	type	randomMinDom	lexico
6	random	0.07	0.10
	balanced	0.15	0.16
8	random	395.50	298.60
	balanced	–	–

**Table 2** Average computation time for Model (1)-(17)

instance		time (sec)	
size	type	randomMinDom	lexico
8	random	–	567.8
	balanced	–	2926.4

On the TTP, [10] reported that he could solve the  $n = 4$  and  $n = 6$  instances but the latter required more than 10 minutes and 100 000 backtracks. Table 1 shows our results on the TTPPV  $n = 6$  and  $n = 8$  instances. The 6-team instances are easily solved whereas only the random 8-team instances are within reach with this model: neither search heuristic could solve a majority of the CIRC8 balanced instances. Note that one CIRC8 random instance could not be solved within the one-hour time limit and was therefore excluded from the average. The lexicographic search heuristic appears to do a little better than randomized smallest domain.

### 3.2 Adding Redundant Constraints

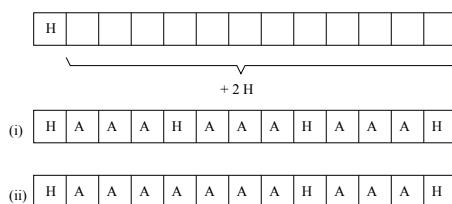
Even though our model accurately describes an optimal solution to the TTPPV, adding some constraints can help us to solve it faster. They will be redundant from a declarative point of view but the inference algorithms they encapsulate may filter out more values from the domains of variables and thus reduce the search space further, at the expense of extra computation.

As already pointed out in [10], the *opponent* variables in a given round must take distinct values:

$$\text{alldifferent}((o_{ij})_{1 \leq i \leq n}) \quad 1 \leq j \leq n - 1 \quad (17)$$

Note that these do not replace Constraints (2), which are still needed.

Table 2 presents new results once redundant Constraints (17) are added. This time every 8-team instance is solved to optimality by the lexicographic heuristic, but not by the randomized-smallest-domain one. Henceforth we will use the former. There is still a sharp difference in performance between the random and balanced instances, probably due to the fact that the former are more constrained and therefore have a smaller search space.



**Fig. 2** 14-team individual schedule with 3 predefined home games and starting with a home game. Both (i) and (ii) completed schedules are invalid.

**Table 3** Average computation time for Model (1)-(5), (7)-(18) with the lexico search heuristic on the CIRC8 instances

instance		time (sec)
size	type	
8	random	413.2
	balanced	2666.4

### 3.3 Catching Infeasible Instances

In addition to proving optimality, another advantage of exact methods is identifying infeasibility. Arguably a successful tree search algorithm needs to be good at pruning infeasible subtrees. Among the CIRC instances, 14 are known to be infeasible. Using the previous model and search heuristic, five of these are proven without any search (the 6- and 8-team infeasible instances), one requires 367 backtracks, and another 2595142 backtracks in about 12 minutes. The other seven could not be proven within one hour.

One source of infeasibility is a team not having enough home (resp. away) games to separate long stretches of away (resp. home) games. A necessary condition is given in [13] and states that at least  $\lfloor \frac{n-1}{4} \rfloor$  of each are needed. The inability of our model to detect this stems from Constraints (6) and (1) being handled separately: the former would need to know that games cannot be repeated. Alternatively we could make sure that we have the correct number of home games in each team's schedule. The `cost_regular` constraint [3], a variant of `regular` that additionally assigns an individual cost to each value taken by a variable and constrains their sum, can do just that:

$$\text{cost\_regular}((h_{ij})_{1 \leq j \leq n-1}, \mathcal{A}, \sum_{1 \leq k \leq n} V[i, k], [0, 1]) \quad 1 \leq i \leq n \quad (18)$$

The last argument attributes a cost to each value possibly taken by the  $h_{ij}$  variables: here an away game for  $i$  costs nothing whereas a home game costs 1, effectively counting the latter. The third argument counts the number of home games for team  $i$  according to matrix  $V$  and constrains the sum of individual costs to that value. This way the constraint simultaneously enforces the correct number of home/away games and the stretch requirement — it is also strictly stronger than the  $\lfloor \frac{n-1}{4} \rfloor$  condition. Consider Figure 2: thirteen games are to be planned, three of them home games, and the first one being

**Table 4** Average computation time for Model (1)-(5), (7)-(19) with simple static symmetry breaking and the lexico search heuristic on the CIRC8 instances

instance size	instance		time (sec)
	type		
8	random		249.1
	balanced		1372.5

played home. Despite the  $\lfloor \frac{n-1}{4} \rfloor$  condition being satisfied, there is no valid way to complete that schedule. For example, schedule (i) respects the stretch constraint but not the number of predefined home games, whereas schedule (ii) respects the latter but not the former. With this more powerful constraint replacing (6), the 14 instances are shown to be infeasible without any search (0 backtrack). This confirms that the source of infeasibility here is an insufficient number of home (or away) games for a team to respect the stretch restriction, now captured by a single constraint. Table 3 also shows that the additional inference improves the overall performance on feasible instances as well by pruning more infeasible subtrees.

### 3.4 Symmetry Breaking

The presence of symmetry in models can considerably slow down tree search approaches because the same infeasible subtree will be met repeatedly. This is especially true when we are solving an optimization problem and the whole tree must be traversed (even if only implicitly by pruning infeasible or provably suboptimal subtrees). Identifying and removing all symmetries is generally a very difficult task. Here there is one symmetry that is easy to see and remove: the mirror image of a schedule, going from the last round to the first. Such a transformation preserves the compact single round robin structure of the schedule and the stretch restriction. Games are still played at the same venues. And because travel distances are symmetric, its cost will be identical. We break that symmetry by selecting the first team, arbitrarily, and requiring that its first opponent be smaller than its last (according to team identifiers):

$$o_{11} < o_{1,n-1} \quad (19)$$

Table 4 shows breaking that symmetry improves the overall performance further, cutting the computation time almost by half. However solving the 10-team instances to optimality remains out of reach within one hour of computation time.

## 4 Exploring the Search Space

Given a CP model and search heuristics to dictate how we branch at a search tree node, there are still many ways we can explore the search space, even if it is all based on tree search. This section explores two possible avenues.

**Table 5** Average solution value for the CIRC 10-team random instances with and without limited discrepancy search

LDS	1 sec.	1 min.	1 hour
no	168.9	163.4	158.9
yes	171.1	164.0	158.6

**Table 6** Best solution value for the feasible CIRC 18-team random instances using different approaches

instance	CP without LDS			CP with LDS			IP	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour	2 hours	1 sec.	2 hours
A	1040	1028	998	1054	986	962	1124	940	806
D	1052	1014	1006	–	1020	972	1060	914	800
E	1018	994	994	1038	1018	972	1092	922	804
F	986	970	962	1006	974	936	1098	956	802
G	1002	988	976	–	990	962	1098	924	782
H	996	972	972	–	1004	956	1110	944	804
I	1000	990	968	–	958	936	1104	932	818
J	944	928	918	–	970	896	1102	898	782

#### 4.1 Adding Robustness to the Search Heuristic

It is well known for tree search that regardless of a search heuristic's quality, a depth-first traversal may take a very long time to undo a bad branching decision made early on. Our simple static search heuristic certainly is no exception. There are a few devices commonly used to add robustness to such search heuristics, e.g. randomized restarts and limited discrepancy search (LDS) [6]. The latter modifies the order in which the leaves of a search tree are visited according to how often the corresponding path deviates from the search heuristic's recommendation: first the leaf with 0 deviation, then those with  $1k$  deviations, followed by those with  $2k$  deviations, and so forth, for a given parameter  $k$ . This has the effect of more quickly changing decisions close to the root. We add LDS to the lexicographic search heuristic as its tree traversal strategy.

Table 5 compares the average value of solutions found for the 10-team random instances after 1 second, 1 minute, and one hour, with and without LDS (the behavior on the balanced instances is similar). Note that the quality starts out worse with LDS but it eventually catches up to and exceeds the performance of the heuristic without LDS. The improvement observed here is small but we next confirm it on instances of realistic size, comparing it at the same time to the state of the art.

The two previous papers on the TTPPV both use the CIRC 18- and 20-team instances. Table 6 and 7 report our results for  $n = 18$  (Column 2 to 7) as well as the best ones for [13] (Column 8) and [2] (Column 9 and 10). Our solutions are considerably better than the ones obtained with IP or their polishing/enumerative heuristics, even after only 1 second of computation.

**Table 7** Best solution value for the feasible CIRC 18-team balanced instances using different approaches

instance	CP without LDS			CP with LDS			IP 2 hours	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour		1 sec.	2 hours
A	998	972	960	1036	980	932	1106	912	776
B	1012	1006	990	1036	972	924	1100	896	796
C	1068	1040	1022	1050	980	978	1038	892	794
D	1020	1002	986	–	1022	988	1096	882	788
E	1018	1012	1006	–	954	948	1074	892	784
F	1044	1030	1016	1034	1014	976	1060	910	792
G	972	948	942	1018	972	918	1100	894	784
H	986	964	956	–	984	918	1094	880	780
I	1004	994	978	1074	982	968	1102	894	778
J	1022	994	966	994	968	930	1078	878	780

**Table 8** Best solution value for the feasible CIRC 20-team random instances using different approaches

instance	CP without LDS			CP with LDS			IP 2 hours	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour		1 sec.	2 hours
A	1422	1410	1380	–	1418	1340	1502	1270	1106
B	1456	1454	1446	–	1436	1358	1522	1258	1082
C	–	–	1440	–	1424	1324	1488	1318	1096
D	1386	1376	1360	–	1368	1334	1510	1294	1136
E	1432	1410	1404	–	1432	1346	1574	1250	1100
G	1400	1380	1370	–	1386	1362	1540	1278	1078
I	1388	1366	1348	–	1360	1304	1516	1236	1082
J	1376	1360	1356	–	1344	1272	1516	1220	1070

For every instance but one, using LDS yields noticeably improved solutions after one hour. Our best solutions after one hour are comparable to those obtained by ILS after one second on the random instances but are inferior on the balanced instances. This may be explained by the larger search space of the latter for our exact approach. On none of the instances are we competitive with ILS given a reasonable amount of time (two hours).

Table 8 and 9 report our results for  $n = 20$  (Column 2 to 7) as well as the best ones for [13] (Column 8) and [2] (Column 9 and 10). Again our solutions are much better than the ones from [13] and LDS systematically improves our exact algorithm. Unfortunately the gap between our performance and that of ILS widens.

#### 4.2 Applying Large Neighborhood Search to the CP Model

Local search is often the method of choice to solve large combinatorial optimization problems and as we saw it is currently the best method for this problem. Several ways to combine CP and local search have been proposed in the literature, e.g. [15], [16], [8]. Large Neighborhood Search (LNS) iteratively

**Table 9** Best solution value for the feasible CIRC 20-team balanced instances using different approaches

instance	CP without LDS			CP with LDS			IP	ILS	
	1 sec.	1 min.	1 hour	1 sec.	1 min.	1 hour	2 hours	1 sec.	2 hours
A	1380	1368	1360	1362	1356	1302	1520	1236	1094
B	1412	1386	1382	1432	1394	1346	1530	1252	1082
C	1408	1388	1370	1366	1362	1338	1470	1234	1072
D	1404	1400	1398	–	1430	1354	1464	1238	1100
E	1426	1416	1402	–	1414	1356	1526	1214	1076
F	1390	1372	1356	1440	1404	1340	1546	1236	1072
G	1348	1334	1316	–	1368	1296	1536	1210	1068
H	1446	1430	1410	1444	1358	1336	1516	1268	1094
I	1378	1356	1352	1422	1362	1310	1544	1238	1078
J	1410	1400	1368	–	1340	1308	1484	1222	1086

**Table 10** Compared best solution value for the feasible CIRC 18-team random (left) and balanced (right) instances using LNS

	LDS		LNS		ILS	
	1 hour	1 hour	1 hour	1 hour	1 sec.	2 hours
A	962	894	940	806	912	776
D	972	902	914	800	896	796
E	972	882	922	804	910	794
F	936	902	956	802	900	788
G	962	896	924	782	866	784
H	956	868	944	804	918	792
I	936	888	932	818	882	784
J	896	870	898	782	894	780
A	932	890	912	776	918	784
B	924	896	896	796	918	780
C	978	910	892	794	968	778
D	988	900	882	788	930	780
E	948	866	892	784		
F	976	918	910	792		
G	918	882	894	784		
H	918	856	880	780		
I	968	914	894	778		
J	930	874	878	780		

freezes part of the current solution and explores the remaining search space (its potentially large neighborhood) by applying a (typically incomplete) CP tree search, benefiting from the usual inference and search heuristics. It is thus easy to transform an exact CP approach to one using LNS.

We tried a simple implementation of this idea. We run our exact CP algorithm for a few seconds in order to get a fair initial solution. We then freeze the schedule of a randomly selected small subset of the teams (here, 6 teams). We explore the neighborhood with the same exact CP algorithm, stopping at the first improving solution or until a time limit of 30 seconds is reached. We stop after 100 consecutive unsuccessful iterations or one hour. Admittedly this is a bare-bones representative of local search methods.

Table 10 reports empirical results on the  $n = 18$  instances. It reiterates some of the results from previous tables and adds a column for LNS. On every instance LNS significantly improves our LDS results. On random instances LNS solution values now lie somewhere between the one-second and two-hours results of ILS. On balanced instances they are now comparable to the one-second results. Table 11 reports empirical results on the  $n = 20$  instances.

**Table 11** Compared best solution value for the feasible CIRC 20-team random (left) and balanced (right) instances using LNS

	LDS		LNS		ILS				
	1 hour	1 hour	1 sec.	2 hours	1 hour	1 hour	1 sec.	2 hours	
A	1340	1206	1270	1106	1302	1270	1236	1094	
B	1358	1294	1258	1082	1346	1270	1252	1082	
C	1324	1286	1318	1096	1338	1264	1234	1072	
D	1334	1250	1294	1136	1354	1268	1238	1100	
E	1346	1278	1250	1100	1356	1244	1214	1076	
G	1362	1280	1278	1078	1340	1218	1236	1072	
I	1304	1258	1236	1082	1296	1248	1210	1068	
J	1272	1234	1220	1070	1336	1266	1268	1094	
					I	1310	1268	1238	1078
					J	1308	1242	1222	1086

Again LNS dominates LDS. On both random and balanced instances it is often comparable to the one-second results of ILS.

## 5 Conclusion

We presented a constraint programming approach to the traveling tournament problem with predefined venues. A model was gradually refined and a few search heuristics and strategies were considered. On standard benchmark instances of realistic size, this approach outperforms a previous integer programming exact approach and its related heuristic variants but falls short of competing with the current best local search approach to this problem.

Much has yet to be tried with this CP approach and we believe there is real potential for improvement. This is especially true of our search heuristic which is currently static: it has performed well but a dynamic search heuristic tailored to the problem should perform better. There is definitely a lot of room for improvement in the local search approach with LNS which is currently very simple. As an exact algorithm it could be brought to solve the 10-team instances in reasonable time. Finally the easy integration of side constraints is also an asset for a CP approach in the real world of sports scheduling.

## References

1. Burke, E.K., Trick, M.A. (eds.): Practice and Theory of Automated Timetabling V, 5th International Conference, PATAT 2004, Pittsburgh, PA, USA, August 18-20, 2004, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 3616. Springer (2005)
2. Costa, F., Urrutia, S., Ribeiro, C.: An ILS heuristic for the traveling tournament problem with predefined venues. *Annals of Operations Research* **194**(1), 137–150 (2012)
3. Demassey, S., Pesant, G., Rousseau, L.M.: A Cost-Regular Based Hybrid Column Generation Approach. *Constraints* **11**(4), 315–333 (2006)
4. Easton, K., Nemhauser, G.L., Trick, M.A.: The Traveling Tournament Problem Description and Benchmarks. In: T. Walsh (ed.) CP, *Lecture Notes in Computer Science*, vol. 2239, pp. 580–584. Springer (2001)

5. Easton, K., Nemhauser, G.L., Trick, M.A.: Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach. In: E.K. Burke, P.D. Causmaecker (eds.) PATAT, *Lecture Notes in Computer Science*, vol. 2740, pp. 100–112. Springer (2002)
6. Harvey, W.D., Ginsberg, M.L.: Limited Discrepancy Search. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI- 95, pp. 607–615. Morgan Kaufmann (1995)
7. Hentenryck, P.V., Carillon, J.P.: Generality versus Specificity: An Experience with AI and OR Techniques. In: H.E. Shrobe, T.M. Mitchell, R.G. Smith (eds.) AAAI, pp. 660–664. AAAI Press / The MIT Press (1988)
8. Hentenryck, P.V., Michel, L.: Constraint-based local search. MIT Press (2005)
9. Henz, M.: Scheduling a Major College Basketball Conference—Revisited. *Operations Research* **49**(1), 163–168 (2001)
10. Henz, M.: Playing with Constraint Programming and Large Neighborhood Search for Traveling Tournaments. In: Burke and Trick [1]
11. van Hove, W.J.: The alldifferent Constraint: A Survey. CoRR **cs.PL/0105015** (2001)
12. Lustig, I.: Scheduling the NFL with Constraint Programming. In: Burke and Trick [1]
13. Melo, R., Urrutia, S., Ribeiro, C.: The traveling tournament problem with predefined venues. *Journal of Scheduling* **12**(6), 607–622 (2009)
14. Pesant, G.: A Regular Language Membership Constraint for Finite Sequences of Variables. In: M. Wallace (ed.) CP, *Lecture Notes in Computer Science*, vol. 3258, pp. 482–495. Springer (2004)
15. Pesant, G., Gendreau, M.: A constraint programming framework for local search methods. *J. Heuristics* **5**(3), 255–279 (1999)
16. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: M.J. Maher, J.F. Puget (eds.) CP, *Lecture Notes in Computer Science*, vol. 1520, pp. 417–431. Springer (1998)



---

# Hyper-Heuristics for Educational Timetabling

Nelishia Pillay

*School of Mathematics, Statistics and Computer Science*

+27 33 2605644

+27 33 2605648

[pillayn32@ukzn.ac.za](mailto:pillayn32@ukzn.ac.za)

**Abstract:** Hyper-heuristics aim at providing generalized solutions to combinatorial optimization problems. Educational timetabling encompasses university examination timetabling, university course timetabling and school timetabling. This paper provides an overview of the use of hyper-heuristics to solve educational timetabling problems. The paper then proposes future research directions focusing on using hyper-heuristics to provide a generalized solution over the domain of educational timetabling instead of for a specific timetabling problem.

**Keywords:** *hyper-heuristics, educational timetabling, university examination timetabling, university course timetabling, school timetabling*

## 1. Introduction

Whereas research into solving combinatorial optimization problems have generally focused on producing the best results for one or more problems, hyper-heuristics aim at generalizing well over a set of problems (Burke et al. 2003). Based on the classification presented by Burke et al. (2010a) hyper-heuristics can be selective or generative. Selection hyper-heuristics choose low-level heuristics to construct or improve a potential solution timetable while generation hyper-heuristics induce new low-level heuristics for a particular domain. Hyper-heuristics can also be categorized as being constructive or perturbative. Constructive hyper-heuristics either select or generate construction low-level heuristics to create a solution. Perturbative hyper-heuristics either choose or generate low-level heuristics to improve an initial candidate solution. A perturbation hyper-heuristic consists of two components one for heuristic selection and another for move acceptance. Thus, the four main categories of hyper-heuristics can be described as selection constructive, selection perturbative, generation constructive and generation perturbative.

There are three main areas of educational timetabling, namely, university examination timetabling, university course timetabling and school timetabling. All

three types of timetabling involve the allocation of events to timetable periods while at the same time satisfying a set of hard constraints and minimizing a set of soft constraints (Qu et al. 2009c; McCollum et al. 2008; Pillay 2010a). These events are exams for exam timetabling, meetings between groups of students and lecturers in a specific venue for university course timetabling and meetings between classes and teachers for school timetabling. Hard constraints of the problem must be met in order to obtain an operable timetable. A timetable meeting the hard constraints is described as feasible. Examples of hard constraints include students not being scheduled to sit for two or more examinations during the same period; classes, teachers and venues not being scheduled more than once in the same period. Soft constraints define characteristics that we would like a timetable to possess, e.g. certain events to be scheduled at a particular time of the day, examinations with large numbers to be scheduled early in the timetable to facilitate marking. The number of soft constraints violated is minimized as these constraints are often contradictory and thus a soft constraint cost of zero is not attainable. There are two types of university course timetabling problems namely, curriculum-based and post enrolment. In the curriculum-based version student enrolment is not known at the time of timetabling construction while in the post enrolment version this is known (McCollum et al.2008).

The paper firstly provides an overview of hyper-heuristics to solve educational timetabling problems. Section 2 focuses on university examination timetabling, section 3 on university course timetabling and section 4 on school timetabling. Section 5 presents an analysis of the use of hyper-heuristics to solve educational timetabling problems and section 6 proposes future research directions.

## **2. Hyper-heuristics for University Examination Timetabling**

A majority of the research into the use of hyper-heuristics for educational timetabling has been for the domain of examination timetabling. The hyper-heuristics employed to solve this problem have been either selection constructive or selection perturbative hyper-heuristics.

## 2.1 Selection Constructive Hyper-Heuristics

Burke et al. (2002) present a case-based hyper-heuristic to solve examination timetabling problems. The hyper-heuristic maintains a case base of previously solved problems and the low-level construction heuristic that was most appropriate to use at each stage of the timetable construction process. A timetable for a new problem is constructed by using the same low-level construction heuristic as that used in a previous case most similar to the current point of construction. A similarity measure was used for this purpose. Each problem in the case base is defined in terms of the problem characteristics and partial solutions, including the heuristic used at each stage of the timetable construction process. The low-level construction heuristics used include largest degree, largest degree using tournament selection, colour degree and saturation degree. The system was evaluated on generated timetabling problems. Tabu search was employed to determine the best list of problem characteristics for case comparisons. In later work (Burke et al. 2006) an additional low-level heuristic, namely, hill-climber which improves an initial solution created randomly using hill-climbing, was added to the heuristic set.

Yang and Petrovic (2004) present a hybrid approach combining a case-based hyper-heuristic and the great deluge algorithm to solve the examination timetabling problem. The great deluge algorithm improves a candidate solution timetable created using a low-level construction heuristic such as largest degree, largest enrollment, largest colour degree, largest weighted degree and saturation degree. Yang et al. implement a case-based hyper-heuristic to choose which construction heuristic to use to create the initial solution. The case base stores previously solved examination timetabling problems and the construction heuristic used. When solving a new examination timetabling problem the hyper-heuristic uses a fuzzy similarity measure to match the problem to problems in the case base and so identify which construction heuristic to apply to create an initial solution which is then improved by the great deluge algorithm. The case base was created using generated examination timetabling problems. The approach produced feasible good quality timetables for problems from the Carter benchmark set.

Burke et al. (2005) compare the performance of a tabu search and a hybrid hyper-heuristic in solving the examination timetabling problem. The former

employs a tabu search to explore a space of combinations of the two low-level construction heuristics, namely, largest degree and saturation degree. The hybrid approach combines case-based reasoning and tabu search. Case-based reasoning is used to determine the percentage of largest degree and saturation degree in each combination. The characteristics of the problem being solved are compared to previous cases. The same hybridization of largest degree and saturation degree is used as that in the case that is the closest match. Tabu search was used to determine the most appropriate list of characteristics to use for comparison to previous cases. Both the hyper-heuristics were used to solve six generated examination timetabling problems and four problems from the Carter benchmark set. The tabu search hyper-heuristic outperformed the hybrid hyper-heuristic.

Burke et al. (2007) investigate the performance of the tabu search hyper-heuristic further by extending the set of low-level heuristics used to include largest colour degree, largest enrollment, largest weighted degree and random ordering. The revised tabu search hyper-heuristic was used to solve eleven of the Carter benchmark problems. In Qu et al. (2009b) the heuristic combinations performing well are studied to identify any patterns with the respect to the positions of the low-level heuristics in the combinations. This revealed that the best performing combination contained the saturation degree and largest weighted degree heuristics however the best percentage of each low-heuristic and the best position of these occurrences in the heuristic combination is problem dependent. Based on this an adaptive mechanism was built into the hyper-heuristic to hybridize the amount of saturation degree and largest weighted degree in a heuristic combination. The hyper-heuristic was used to solve eleven problems from the Carter benchmark set.

Qu and Burke (2005) investigate the use of a selection constructive hyper-heuristic to solve the examination timetabling problem. The hyper-heuristic employs variable neighbourhood search to explore a space of heuristic combinations consisting of two or more graph colouring heuristics, namely, color degree, largest degree, largest enrollment, largest weighted degree, saturation degree or random ordering. Each heuristic is applied in order to allocate an exam to a minimum penalty period. The hyper-heuristic was used to solve the Carter benchmark set of problems.

Pillay (2008, 2010b, 2012) implement an evolutionary algorithm hyper-heuristic to search a space of heuristic combinations of low-level construction heuristics chosen from a set containing the largest degree, largest weighted degree, largest enrollment, saturation degree and highest cost heuristics. The hyper-heuristic was able to produce good quality timetables for both the Carter set of benchmark problems and the benchmark set for the second international timetabling competition (McCollum et al. 2008). This research also examined the effect of the representation used for heuristic combinations on the performance of the evolutionary algorithm hyper-heuristic. Three representations, namely, fixed length, variable length and n-times and a combination of all three representations were tested. The latter option produced the best results.

The hyper-heuristic implemented by Burke et al. (2009b) employed the greedy random adaptive search procedure (GRASP) to hybridize the use of two low-level construction heuristics, namely, saturation degree and largest weighted degree, in choosing the next examination to schedule during the timetable construction process. An improvement phase is also conducted to improve the candidate solution constructed. Steepest descent is used for this purpose. The hyper-heuristic was used to solve problems in the Carter benchmark set.

Qu and Burke (2009a) compare the performance of various hyper-heuristics, each employing a different search to explore the heuristic space, to solve the examination timetabling problem. These hyper-heuristics search a space of heuristics combinations comprised of low-level construction heuristics. The combinations are constructed by selecting heuristics from a set containing the largest degree, largest weighted degree, largest colour degree, largest enrollment, saturation degree and random ordering heuristics. The hyper-heuristics were tested on eleven problems from the Carter benchmark set. The iterated local search hyper-heuristic was found to produce the best results. The performance of the hyper-heuristic was improved by searching the solution space, using iterative local search, at different intervals during timetable construction.

Saber et al. (2011) have used a selection constructive hyper-heuristic to solve this problem. In this study four low-level heuristics are combined to decide which examination to schedule next. The latter three heuristics in the combination are used to deal with ties. Roulette wheel selection is used to decide which period to

allocate the examination to. The hyper-heuristic was tested on the benchmark set for the second international timetabling competition.

## 2.2 Selection perturbative hyper-heuristics

Kendall and Hussin (2004) use a tabu search hyper-heuristic to solve the examination timetabling problem for MARA University. The tabu search hyper-heuristic is used to improve an initial solution created using either the largest degree or saturation degree construction low-level heuristic. Two variations of the standard tabu search hyper-heuristic, namely, tabu search hyper-heuristics with hill-climbing and tabu search hyper-heuristics with great deluge were also tested. Low-level heuristics include five move heuristics that reschedule examinations, two swap heuristics that swap the periods of two exams, a heuristic that un schedules an examination, and five construction heuristics (largest enrolment, largest degree, largest weighted degree, largest colour degree, and saturation degree) to reschedule unscheduled exams. The timetable produced by the hyper-heuristic was an improvement on the manually created timetable used by the university. In later work Kendall and Hussin (2005) applied the tabu search hyper-heuristic to eight problems from the Carter benchmark set.

Biligin et al. (2006) test seven approaches for heuristic selection and five for move acceptance. The heuristic selection methods include simple random, random descent, random permutation, random permutation descent, choice function, tabu search, and a greedy method. The three move acceptance approaches evaluated are accept all moves, accept improving moves only, great deluge and Monte Carlo. Low-level heuristics used in the study include three hill-climbing operators (next ascent hill-climbing, Davis' bit hill climber, random mutation hill climber) and three mutation operators (swap dimension, dimensional mutation and hypermutation). All six operators are applied to binary operands. The hyper-heuristic was used to solve the Carter benchmark set of problems and the examination timetabling for the Faculty of Architecture and Engineering at Yeditepe University. The hyper-heuristic combining the use of a choice function and Monte Carlo for move acceptance produced the best results.

In Ersoy et al. (2007) a hyper-heuristic is embedded in a memetic algorithm used to solve the examination timetabling problem. The hyper-heuristic is used to select one of three hill-climbers to be used by the memetic algorithm. The

memetic algorithm using various hyper-heuristics was tested on six of the Carter benchmark problems. Self-adaptive hyper-heuristics using either a choice function for heuristic selection and great deluge for move acceptance or simple random combined with improving and equal move acceptance, were found to perform well.

Burke et al. (2008) study the use of simulated annealing selection perturbative hyper-heuristics. Simulated annealing is used for move acceptance. Three methods, namely, simple random, a greedy method and a choice function are evaluated for heuristic selection. Four low-level heuristics which reschedule examinations are used. Three of the heuristics attempt to reschedule exams so as to remove constraint violations. The last heuristic attempts to reschedule all the allocated exams. The hyper-heuristic using a choice function for heuristic selection with simulated annealing for move acceptance was found to outperform the other hyper-heuristic combinations.

Ozcan et al. (2009) also implement a perturbation hyper-heuristic to solve the examination timetabling problem. The move acceptance component employs a late acceptance strategy. Instead of comparing the current candidate solution to that obtained on the previous iteration, the move acceptance component compares it to a solution from  $n$  previous iterations. Heuristic selection methods tested include simple random, greedy, reinforcement learning, reinforcement learning with tabu search, and a choice function. Four low-level heuristics are implemented. The first is a mutation operator which attempts to reschedule all exams. The remaining three heuristics reschedule exams so as to reduce constraint violations. Tournament selection is used to select an exam and to select a slot to reschedule the examination in. The hyper-heuristic using simple random for heuristic selection and late acceptance strategy for move acceptance produced the best results.

Burke et al. (2010b) have implemented a Monte Carlo selection perturbative hyper-heuristic to solve the capacitated version of the Carter benchmark set (Qu et al. 2009) of examination timetabling problems. This set consists of data collected from thirteen different institutions. Methods tested for heuristic selection include simple random, a greedy method, a choice function and reinforcement learning. Similarly, different methods were made available for move acceptance, namely, simulated annealing, simulated annealing with reheating and an exponential

Monte Carlo method. Three low-level perturbative heuristics are used. The first reschedules an exam based on the number of conflicts the examination is involved in. The second reschedules exams so as to meet the capacity constraint while the third reschedules an examination randomly. The hyper-heuristic producing the best results for the benchmark set used the choice function for heuristic selection and simulated annealing for move acceptance.

Burke et al. (2010c) employ a hyper-heuristic to improve the quality of an initial feasible solution created using the largest degree construction heuristic. The hyper-heuristic uses four low-level perturbative heuristics, namely, move exam, swap exam, Kempe chain move and swap timeslot. All four heuristics aim at producing the least penalty timetable. Preliminary studies indicated that Kempe chain in combination with swap timeslot performed the best over problems of differing characteristics. The best hybridization (i.e. percentage occurrence and position) of these two heuristics in an optimal heuristic combination is problem dependent. An adaptive component is built into the hyper-heuristic to perform the hybridization of these two heuristics. The saturation degree is used to choose an examination, causing a soft constraint violation, which the move operator is applied to. The hyper-heuristic was used to find solutions to problems from the Carter benchmark set and the benchmark set for the second international timetabling competition.

Ozcan et al. (2012) have implemented a selection perturbative hyper-heuristic employing reinforcement learning for heuristic selection and great deluge for move acceptance. The hyper-heuristic was used to improve an initial solution. Three types of low-level heuristics were used. The first type aims at rescheduling the examination causing the most constraint violations in a set of  $n$  examinations. The second reschedules the examination that has the highest impact on the capacity violation for a particular period from a set of  $n$  periods with capacity violations. The last type of low-level heuristic attempts to reschedule all allocated examinations probabilistically. The hyper-heuristic was used to induce timetables for Yeditepe University and the Carter benchmark set.

In the study conducted by Sin and Kham (2012) reinforcement learning is used for heuristic selection and great deluge for move acceptance. Three variants of great deluge were tested, namely, flex deluge, non-linear great deluge, and extended great deluge. Low-level heuristics focused on changing timeslots of



examinations or swapping subsets of examinations between two timeslots. The hyper-heuristic was used to improve an initial solution created using the largest enrollment construction heuristic. Evaluation on the Carter benchmark set revealed that the hyper-heuristic using the extended great deluge for move acceptance was the most effective.

### **2.3 Selection generative hyper-heuristics**

Asmuni et al. (2005; 2007; 2009) combine low-level graph heuristics, namely, largest degree, saturation degree and largest enrollment, using a fuzzy logic function. The fuzzy function combines two to three heuristics and the single value produced is used to sort examinations to be scheduled according to difficulty. The hyper-heuristic was used to solve the Carter benchmark set of problems.

Pillay and Banzhaf (2009b) proposed that low-level heuristics be combined hierarchically allowing them to be applied simultaneously instead of combining them linearly and applying them sequentially. The use of conditional and logically operators have facilitated the hierarchical combination and simultaneous application of low-level construction heuristics chosen from largest degree, largest weighted degree, largest enrollment, saturation degree and highest cost heuristics. Four such combinations were created and tested on the Carter benchmark set of problems. These combinations produced results competitive to other hyper-heuristics tested on the same benchmark set of problems. Pillay (2009a) automates the process of creating the hierarchical heuristic combinations. In this study genetic programming is used to evolve these combinations comprised of conditional and logical operators and the low-level heuristics.

Pais and Burke (2010) use a Choquet integral to combine five low-level construction heuristics, namely, largest degree, colour degree, largest weighted degree, largest enrollment and saturation degree. The single value produced by the Choquet integral estimates the difficulty of scheduling an examination. The examinations are sorted in decreasing order according to this value and allocated accordingly. The performance of the Choquet integral is compared to that of each of the low-level heuristics applied individually to sort the examinations. The low-level heuristics and the Choquet integral were evaluated on the Carter benchmark problems and the benchmark set for the second international timetabling competition. The Choquet integral produced the best results for eleven of the

thirteen Carter problems and for five of the eight timetabling competition problems. Burke and Pais (2011) extend this work and evaluate differential evolution to induce fuzzy measures to estimate examination difficulty. This improved the performance of the hyper-heuristic.

## 2.4 Summary of Hyper-Heuristic Performance

This section summarizes the performance of the different types of hyper-heuristics in solving the examination timetabling problem. There are essentially two benchmark problem sets that these hyper-heuristics have been applied to, namely, the Carter (also known as the Toronto) benchmark set (Qu et al. 2009c) and the benchmark set used for the second international timetabling competition ITC' 2007 (McCollum et al. 2008). A majority of the hyper-heuristics have been evaluated on the Carter benchmark set. The characteristics of the problems included in this benchmark set are listed in Table 1 in Appendix A. This benchmark set has been constructed by collecting data from real-world educational institutions. The density of the clash matrix is a ratio of the number students involved in clashes to the total number of students and is a measure of the difficulty of the problem. The hard constraint for this set of problems is that there must be no clashes, i.e. a student must not be scheduled to write more than one examination at a time. The soft constraint is that the examinations must well-distributed over the examination period for any one student. A distance formula is used to calculate the soft constraint cost (Qu et al. 2009c). Performance of selection constructive, selection perturbative and generation constructive hyper-heuristics applied to the Carter benchmarks are tabulated in Appendix B, Appendix C and Appendix D respectively. Note that only those studies that have reported these results are included.

From the results presented in Appendix B the hybrid approach combining case-based reasoning with the great deluge algorithm appears to have performed the best. The evolutionary algorithm hyper-heuristic, using a combination of three different representations for individuals, has also produced fairly good results. Selection perturbative hyper-heuristics have only been applied to subsets of Carter benchmark problems and need to be evaluated further. The best performing selection perturbative hyper-heuristic is the adaptive selection perturbative hyper-heuristic implemented by Burke et al. (2012c). However, the

results produced by this hyper-heuristics is not as good as that of the selection constructive hyper-heuristics listed in Appendix B. Similarly, the generation constructive hyper-heuristics presented in Appendix D do not perform as well as the selection constructive hyper-heuristics on the Carter benchmark set.

The comparison in this section has been restricted to examination timetabling as there has not been sufficient research into university course timetabling and school timetabling to do the same.

### **3. Hyper-Heuristics for University Course Timetabling**

The use of hyper-heuristics to solve the university course timetabling problem is not as well researched as for the university examination timetabling problem. Most of the research in this area has focused on the use of selection constructive hyper-heuristics to find solutions to this problem.

#### **3.1 Selection constructive hyper-heuristics**

Rossi-Doria and Paechter (2003) implement an evolutionary algorithm selection constructive hyper-heuristic. Each chromosome is a comprised of two rows of integers representing heuristics. The first row represents heuristics to choose which event to schedule next and are chosen from largest degree, largest colour degree, least saturation degree, maximum weighted number of event correlations, maximum number of students, maximum number of features by events, minimum number of possible rooms, event with room suitable for most events, least saturation degree with room consideration. The second row represents heuristics used to select room and timeslots, e.g. smallest possible room, least room suitable, least used room, latest or earliest timeslot in the day. The evolutionary algorithm is steady-state and uses binary tournament selection. One point crossover and mutation is used to produce offspring. The hyper-heuristic was tested on five generated problems of medium difficulty and produced competitive results for two of these problems.

A case-based hyper-heuristic is proposed in Burke et al. (2006) to solve the university course timetabling problem. The case base stores previously solved problems in terms of problem features and steps of the construction process and the low-level construction heuristic used to schedule each event. Each new

problem is solved by finding a match to stored cases at each stage of the timetable construction process. The hyper-heuristic was used to solve generated university course timetabling problems.

Burke et al. (2007) implement a tabu search to explore a space of heuristic combinations of low-level construction heuristics. These heuristics include, random ordering, largest degree, saturation degree, largest colour degree, largest enrollment and largest weighted degree. The hyper-heuristic was used to solve eleven benchmark course timetabling problems (Socha et al. 2002).

Qu et al. (2009a) evaluate various search methods for use by a selection constructive hyper-heuristic. The hyper-heuristic, using different search techniques, was tested on eleven benchmarks problem made available by Socha et al. (2002). The hyper-heuristic employing variable neighbour search to explore the space of heuristic combinations comprised of low-level construction heuristics produced the best results for the benchmark set. The low-level construction heuristics used include largest degree, largest weighted degree, largest colour degree, largest enrollment, saturation and random ordering heuristics. A variation of the hyper-heuristic employing iterative local search to explore the solution space at various stages during the timetable construction process was found to improve the performance of the hyper-heuristic.

### **3.2 Selection perturbative hyper-heuristics**

The selection perturbative hyper-heuristic implemented by Bai et al. (2007a, 2007b) uses simulated annealing for move acceptance. Heuristic selection is initially random until a heuristic performance history has been developed and is then based on the performance of the low-level heuristics in the previous iterations. Three low-level perturbative heuristics are available for use by the hyper-heuristic. The first reschedules a randomly selected event. The second swaps the periods of two randomly chosen events. The third swaps the events of two randomly selected periods. The hyper-heuristic is used to improve an initial feasible solution. The hyper-heuristic was tested on two benchmark problem sets. The first set contained five small, five medium and one large problem and the second twenty problems. The hyper-heuristic performed better than two other hyper-heuristics and meta-heuristics applied to the problems in the first benchmark set and to one of the problems in the second benchmark set.

### **3.3 Generation Perturbative Hyper-Heuristic**

In the study conducted by Rattadilok (2010) an initial solution is created using a random or greedy approach and is improved using a generation perturbative hyper-heuristic. A choice function is used for heuristic selection. This function selects low-level heuristics based on their previous performance. Swap operators are used as low-level heuristics. Each operator is created by making configuration decisions, namely, a number of candidates involved in the swap, swap candidate sets and acceptance criteria for termination. Sub-controllers are used to formulate configuration decisions. Ten low-level swap heuristics are used. The hyper-heuristic was applied to data sets from the first international university course timetabling competition.

## **4. School Timetabling**

There has not been much research conducted into the use of hyper-heuristics for solving the school timetabling problem. There have basically been two studies, one investigating the use of a selection constructive hyper-heuristic and the second evaluating a generation constructive hyper-heuristic in solving the school timetabling problem.

### **4.1 Selection constructive hyper-heuristics**

Pillay (2010c) implements an evolutionary algorithm hyper-heuristic to solve the school timetabling problem. The evolutionary algorithm explores a space of heuristic combinations of low-level construction heuristics. Construction heuristics used include random ordering, largest degree, saturation degree, class degree, teacher degree and class-teacher degree. The hyper-heuristic was tested with different subsets of low-level heuristics from which the elements of each heuristic combination are chosen. The subset consisting of largest degree and saturation degree produced the best results. The incorporation of hill-climbing in the genetic operators was found to improve the performance of the EA hyper-heuristic. The EA hyper-heuristic produced competitive results in solving a difficult generated problem and outperformed a neural work and greedy search applied to the same problem. Pillay (2011a) applied this EA hyper-heuristic to solving the school timetabling problem for a South African primary school. In this study the low-level construction heuristic set included largest degree,

saturation degree, double degree and period preference degree. A Pareto function of the hard and soft constraint costs was found to be the most effective option to evaluate the fitness of each heuristic combination. The EA hyper-heuristic produced a solution of better quality than that currently being used by the school.

#### **4.2. Generation constructive hyper-heuristics**

Pillay (2011b) employed genetic programming to evolve heuristics for the school timetabling problem. The function set was composed of arithmetic operators, arithmetic logical operators and conditional operators. The terminal set contains variables to represent the characteristics of the problem, namely, the number of class-teacher meetings a class is involved in and the number of class-teacher meetings a teacher is involved in as well as the heuristics that are traditionally used in solving the school timetabling problem, namely, largest degree and saturation degree. The hyper-heuristic was tested on a difficult generated problem and performed better than the saturation degree and largest degree applied to the same problem. The generation hyper-heuristic also performed better than a tabu search, the evolutionary algorithm hyper-heuristic described in section 4.1, a Hopfield neural network and greedy search in solving the same problem.

### **5. Discussion and Future Research Directions**

As is evident from the above discussion most of the research into using hyper-heuristics for solving educational timetabling problems has been on examination timetabling and on selection constructive hyper-heuristics for this domain. Furthermore selection constructive hyper-heuristics appear to perform the best for examination timetabling. However, this may not be a fair comparison at this stage as the other types of hyper-heuristics have not been as well researched. Hyper-heuristic research for the three different types of educational timetabling have been conducted in isolation of each other. Future research should aim at investigating the effectiveness of the different types of hyper-heuristics for educational timetabling as a whole. The hyper-heuristics also need to be more widely tested. The two benchmark sets available for examination timetabling are the Carter benchmark set (Qu et al. 2009c) and the benchmark set of problems used for the examination timetabling track of the second international timetabling competition (McCollum et al. 2008). There are three benchmark sets available for

the university course timetabling problem, namely, that provided by Socha et al. (2002), the problem set used for the first international timetabling competition (Paechter et al. 2003) and the data sets for curriculum-based and post enrolment university course timetabling tracks of the second international timetabling competition. The third international timetabling competition (Post 2011) is focused on school timetabling and various real-world school timetabling data sets have been made available for the competition.

A fair amount of research has been conducted into using selection constructive hyper-heuristics to solve the different educational timetabling problems. In a majority of these studies a metaheuristic has been employed to explore the space of heuristic combinations. These heuristic combinations are comprised of low-level construction heuristics. The low-level heuristics that have been used for this purpose have generally been the graph colouring heuristics, namely, largest degree, largest weighted degree, largest colour degree, largest enrollment and saturation degree. Some studies have introduced other low-level heuristics, namely, highest cost which estimates the soft constraint cost (Pillay and Banzhaf 2009b) and period and room heuristics introduced by Rossi-Doria and Paechter (2003). Metaheuristics used to search such a heuristic space include tabu search, iterated local search, variable neighbourhood search and evolutionary algorithms. Case-based reasoning has also been used for low-level heuristic selection. The best performing hyper-heuristic was a hybrid combining case-based reasoning and greatest deluge. Processes that automate the hybridization of low-level heuristics that perform well have also been studied for selection constructive hyper-heuristics. The effectiveness of searching both the solution space and heuristic space during the construction of a timetable has also been illustrated. This needs to be investigated further for educational timetabling in general.

The use of selection perturbative hyper-heuristics for solving educational timetabling problems, have also been researched. Low-level heuristics commonly used by these hyper-heuristics include hill-climbing operators, mutation operators, rescheduling events with high constraint violation costs, swapping events or subsets of events, swapping timetable periods, unscheduling and rescheduling events. Techniques commonly used for heuristic selection include simple random, greedy, a choice function, reinforcement learning and tabu search. Methods evaluated for move acceptance are the late acceptance strategy,

simulated annealing, Monte Carlo and the great deluge algorithm. Selection perturbative hyper-heuristic heuristic selection and move acceptance pairs that have performed well for different timetabling problems include a choice function with either simulated annealing, Monte Carlo or the great deluge algorithm and simple random with the late acceptance strategy. Further research into the effectiveness of different methods for driving this category of hyper-heuristics as well as a more expansive evaluation of these hyper-heuristics needs to be conducted.

There has not been much research into the use of generation hyper-heuristics in solving educational timetabling problems. Fuzzy logic and genetic programming are the most popular methods used to induce constructive low-level heuristics. There has only been one study into generation perturbative hyper-heuristics for solving timetabling problems, namely, that conducted by Rattadilok (Rattadilok 2010) to configure swap operators. The generation of low-level construction heuristics for timetabling needs to be researched further. Generally, graph colouring heuristic have been used as low-level heuristics for timetable construction. The induction of heuristics based on problem characteristics need to be studied. Given its success in other domains (Burke et al. 2009a), genetic programming can be investigated for this purpose.

Most selection constructive hyper-heuristics have focused on constructive heuristics for selecting which event to schedule next. There is a need for investigations into developing selection and generation hyper-heuristics that cater for construction heuristics for choosing timetable periods and rooms in addition to heuristics for event selection.

An area that has not been investigated is that of hybrid hyper-heuristics that combine different types of hyper-heuristics, e.g. combining selection constructive and perturbative hyper-heuristics. Furthermore, should such combinations be sequential, i.e. apply one type of hyper-heuristic followed by another, or should there be an alternating application of the different hyper-heuristics.

## 7. References

- Asmuni, H., Burke, E.K. & Garibaldi, J.M. (2005) Fuzzy Multiple Ordering Criteria for Examination Timetabling. In: Burke, E.K. and Trick, M. (Eds.), selected papers from the *5th International Conference on the Theory and Practice of Automated Timetabling (PATAT*



2004) - *The Theory and Practice of Automated Timetabling V, Lecture Notes in Computer Science*, 3616, 147–160.

- Asmuni, H., Burke, E.K., Garibaldi, J.M. & McCollum, B. (2007) Determining Rules in Fuzzy Multiple Heuristic Orderings for Constructing Examination Timetables. In: Bapiste, P., Munier, A. Kendall, G. & Sourd, F. (Eds.), *proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications Conference, MISTA 2007* (pp. 59-66).
- Asmuni, H., Burke, E.K., Garibaldi, J.M., McCollum, B. & Parkes, A.J. (2009). An Investigation of Fuzzy Multiple Heuristic Orderings in the Construction of University Examination Timetables. *Computers and Operations Research*, 36(4), 981-1001.
- Bai, R., Blazewicz, J., Burke, E.K., Kendall, G. & McCollum, B. (2007a) *A Simulated Annealing Hyper-Heuristic Methodology for Flexible Decision Support* (Technical Report No. NOTTCS-TR-2007-8). School of Computer Science and Information Technology, University of Nottingham, Nottingham.
- Bai, R., Burke, E.K., Gendreau, M., Kendall, G. & McCollum, B. (2007b) Memory Length Hyper-Heuristics: An Empirical Study. In *proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, CI-Sched 2007* (pp. 173-178).
- Bilgin, B., Ozcan, E. & Korkmaz, E.E. (2006) An Experimental Study on Hyper-Heuristics and Exam Timetabling. In Burke, E.K. & Rudova, H. *proceedings of the international conference on the Practice and Theory of Automated Timetabling, PATAT 2006* (pp. 123-140).
- Burke, E.K. , Dror, M. , Petrovic, S. & Qu, R. (2005) Hybrid Graph Heuristics with a Hyper-Heuristic Approach to Exam Timetabling Problems. In: Golden, B., Raghavan, S. & Wasil, E.A. (Eds.), *the Next Wave in Computing, Optimization, and Decision Technologies – Conference Volume of the 9th Inform Computing Society Conference* (pp. 79 -91).
- Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P. & Schulenburg, S. (2003) Hyper-Heuristics: An Emerging Direction in Modern Research. In the *Handbook of Metaheuristics*, Chapter 16, 457– 474.
- Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. & Woodard, J. (2009a) Exploring Hyper-Heuristic Methodologies with Genetic Programming. *Computational Intelligence*, 6, 177-201.
- Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. & Woodard, J. (2010a) A Classification of Hyper-Heuristic Approaches. In the *Handbook of Metaheuristics*, International Series in Operations Research and Management Science, Volume 146, 449-468.
- Burke, E.K., Kendall, G., Misir, M. & Ozcan, E. (2008) A Study of Simulated Annealing Hyper-Heuristics. In the *proceedings of the international conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, <http://www.asap.cs.nott.ac.uk/patat/patat08/Papers/Ozcan-HD3a.pdf> . Accessed 12 February 2012.
- Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. & Qu, R. (2007) A Graph-Based Hyper-Heuristic for Educational Timetabling Problems. *European Journal of Operational Research*, 176, 177-192.

- Burke, E. K., Kendall, G., Misir, M. & Ozcan, E. (2010b) Monte Carlo Hyper-Heuristics for Examination Timetabling. *Annals of Operations Research*, doi 10.1007/s10479-010-0782-2.
- Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. & Qu, R. (2007). A Graph-Based Hyper-Heuristic for Educational Timetabling Problems. *European Journal of Operational Research*, 176, 177 – 192.
- Burke, E. K. & Pais, T. C. (2011) Using Differential Evolution to Identify Fuzzy Measures for the Exam Timetabling Problem. In proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2011 (pp. 335-351).
- Burke, E.K., Petrovic, S. & Qu, R. (2002) Case Based Heuristic Selection for Examination Timetabling. In *proceedings of SEAL '02* (pp. 277-281).
- Burke, E.K., Petrovic, S. & Qu, R. (2006) Cased-Based Heuristic Selection for Timetabling Problems. *Journal of Scheduling*, 9(2), 115-132.
- Burke, E.K., Qu, R. & Soghier, A. (2009b) Adaptive Selection of Heuristics within a GRASP for Exam Timetabling. In *proceedings of the Multidisciplinary Conference on Scheduling: Theory and Application, MISTA 2009* (pp. 409-423).
- Burke, E.K., Qu, R. & Soghier, A. (2010c) Adaptive Selection of Heuristics for Improving Constructed Exam Timetables. In proceedings of the 8<sup>th</sup> International Conference on the Practice and Theory of Automated Timetabling, PATAT 2010 (pp. 136 -151).
- Ersoy, E., Ozcan, E. & Uyar, S. (2007). Memetic algorithms and hillclimbers. In: Baptiste, P., Kendall, G., Kordon, A.M. & Sourd, F. (Eds.), *proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications Conference, MISTA 2007* (pp. 159–166).
- Kendall, G. & Hussin, M.H. (2004) Tabu Search Hyper-Heuristic Approach to the Examination Timetabling Problem at University Technology MARA. In the *proceedings of the international conference on the Practice and Theory of Automated Timetabling, PATAT 2004* (pp. 270-295).
- Kendall, G. & Hussin, N.M. (2005). An investigation of a tabu search based on hyper-heuristics for examination timetabling. In: Kendall G., Burke E.K. & Petrovic S. (Eds.) *proceedings of the 2nd Multidisciplinary Scheduling: Theory and Applications Conference, MISTA 2005* (pp. 309–328).
- McCollum, B., McMullan, P., Paechter, B., Lewis, R., Schaerf, A., DiGaspero, L., Parkes, A.J., Qu, R. & Burke, E.K. (2008). Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal of Computing*, 22(1), 120–130.
- Ozcan, E., Bykov, Y., Birben, M. & Burke, E.K. (2009) Examination Timetabling Using Late Acceptance Hyper-Heuristics. In *proceedings of the IEEE Congress on Evolutionary Computing, CEC '09* (pp. 997-1004).
- Ozcan, E., Misir, M., Ochoa, G. & Burke, E.K. (2012) A Reinforcement Learning – Great-Deluge Hyper-Heuristic for Examination Timetabling. *Modeling, Analysis, and Applications in Metaheuristic Computing*, 34-55.

- Pais, T.C. & Burke, E. K. (2010) Choquet Integral for Combining Heuristic Values for Exam Timetabling Problem. In *proceedings of the 8<sup>th</sup> International Conference on the Practice and Theory of Automated Timetabling, PATAT 2010* (pp. 305 -320).
- Paechter, B., Gambardella, L. M., Rossi-Doria, O. (2003) International Timetabling Competition, <http://www.idsia.ch/Files/ttcomp2002/oldindex.html>. Accessed 1 July 2012.
- Pillay, N. (2008) An Analysis of Representations for Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem in a Genetic Programming System. In Cilliers, C., Barnard, L. & Botha R. (Eds.), *proceedings of SAICSIT 2008* (pp. 188-192).
- Pillay, N. (2010a) An Overview of School Timetabling Research. In *proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, PATAT '10* (pp. 321-335).
- Pillay, N. (2010b) Evolving Hyper-Heuristics for a Highly Constrained Examination Timetabling Problem. In *proceedings of the 8th international conference on the Practice and Theory of Automated Timetabling, PATAT 2010* (pp. 336-346).
- Pillay, N. (2010c) A Study into the Use of Hyper-Heuristics to Solve the School Timetabling Problem. In *proceedings of SAICSIT 2010* (pp. 258-264).
- Pillay, N. (2011a) A Hyper-Heuristic Approach to Solving School Timetabling Problems. In *proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2011* (pp.628-632).
- Pillay, N. (2011b) Evolving Heuristics for the School Timetabling Problem. In *proceedings of the 2011 IEEE Conference on Intelligent Computing and Intelligent Systems (ICIS011)*, Vol. 3 (pp. 281-286).
- Pillay, N. (2012) Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem. *Journal of the Operational Research Society*, 63, 47-58.
- Pillay, N. (2009a) Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem. In *proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2009* (pp. 409-422).
- Pillay, N. & Banzhaf, W. (2009b) A Study of Heuristic Combinations for Hyper-Heuristic Systems for the Uncapacitated Examination Timetabling Problem. *European Journal of Operational Research*, 197, 482-491.
- Post, G. (2011) Third International Timetabling Competition (ITC 2011), <http://www.utwente.nl/ctit/itc2011/>. Accessed 1 July 2012.
- Qu, R. & Burke, E.K. (2005). Hybrid Variable Neighbourhood HyperHeuristics for Exam Timetabling Problems. In: *Proceedings of the MIC2005: The Sixth Metaheuristics International Conference*, Vienna, Austria. <http://www.cs.nott.ac.uk/~rxq/files/MIC05.pdf>, accessed 28 June 2008.
- Qu, R. and Burke, E.K. (2009a). Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60, 1273–1285.

- Qu, R., Burke, E.K. & McCollum, B. (2009b). Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal Operational Research*, 198(2), 392–404.
- Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G. & Lee, S.Y. (2009c) A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1), 55–89.
- Rattadilok, P. (2010) An Investigation and Extension of a Hyper-Heuristic Framework. *Informatica*, 34, 523-534.
- Rossi-Doria, O. & Paechter, B. (2003) A Hyperheuristic Approach to Course Timetabling Problems Using an Evolutionary Algorithm, <http://www.metaheuristics.net/media/documents/hyperEA.pdf>. Accessed 12 February 2012.
- Saber, N.R., Ayob, M., Qu, R. & Kendall, G. (2011) A Graph Colouring Constructive Hyper-Heuristic for Examination Timetabling Problems. *Applied Intelligence*, doi: 10.1007/s10489-011-0309-9.
- Sin, E.S. & Kham, N.S.M. (2012) Hyper Heuristic Based on Great Deluge and its Variants for Exam Timetabling Problem. Cornell University Library, <http://arxiv.org/abs/1202.1891>. Accessed 12 February 2012.
- Socha, K., Knowles, J. & Sampels, M. (2002) A Max-Min Ant System for the University Course Timetabling Problem. In *proceedings of the 3<sup>rd</sup> International Workshop on Ant Algorithms. Lecture Notes in Computer Science*, 2463, 1-13.
- Yang, Y. & Petrovic, S. (2004) A Novel Similarity Measure for Heuristic Selection in Examination Timetabling. In the *proceedings of the international conference on the Practice and Theory of Automated Timetabling, PATAT 2004* (pp. 247-269).

## Appendix A – Carter Benchmark Data Set

Table 1. Characteristics of problems in the Carter benchmark set

<b>Data</b>	<b>Institution</b>	<b>Periods</b>	<b>No. of Exams</b>	<b>No. of Students</b>	<b>No. Enrolments</b>	<b>Density of Conflict Matrix</b>
car-f-92 I	Carleton University, Ottawa	32	543	18419	55522	0.14
car-s-91 I	Carleton University, Ottawa	35	682	16925	56877	0.13
ear-f-83 I	Earl Haig Collegiate Institute, Toronto	24	190	1125	8109	0.27
hec-s-92 I	Ecole des Hautes Etudes Commerciales, Montreal	18	81	2823	10632	0.42
kfu-s-93	King Fahd University of Petroleum and Minerals, Dharan	20	461	5349	25113	0.06
lse-f-91	London School of Economics	18	381	2726	10918	0.06
pur-s-93 I	Purdue University, Indiana	43	2419	30029	120681	0.03
rye-s-93	Ryerson University, Toronto	23	486	11483	45051	0.08
sta-f-83 I	St Andrew's Junior High School, Toronto	13	139	611	5751	0.14

tre-s-92	Trent University, Peterborough, Ontario	23	261	4360	14901	0.18
uta-s-92 I	Faculty of Arts and Sciences, University of Toronto	35	622	21266	58979	0.13
ute-s-92	Faculty of Engineering, University of Toronto	10	184	2749	11793	0.08
yor-f-83 I	York Mills Collegiate Institute, Toronto	21	181	941	6034	0.29

## Appendix B – Performance of Selection Constructive Hyper-Heuristics

Table 2. Comparison of selection constructive hyper-heuristic performance

<b>Data</b>	<b>Yang &amp; Petrovic (2004)</b>	<b>Burke et al. (2007)</b>	<b>Burke et al. (2009)</b>	<b>Qu &amp; Burke (2005)</b>	<b>Pillay (2012)</b>	<b>Qu &amp; Burke (2009a)</b>	<b>Qu &amp; Burke (2009b)</b>	<b>Sabar et al. (2011)</b>
car-f-92 I	3.93	4.84	4.74	4.7	4.22	4.77	4.32	4.70
car-s-91 I	4.50	5.41	5.48	5.4	4.95	5.3	5.11	5.14
ear-f-83 I	33.71	38.19	37.71	37.29	35.95	38.39	35.56	37.86
hec-s-92 I	10.83	12.72	12.41	12.23	11.27	12.72	11.62	11.90
kfu-s-93	13.82	15.76	16.84	15.11	14.12	15.09	15.18	15.30

lse-f-91	10.35	13.15	12.29	12.27	10.76	12.72	11.32	12.33
pur-s-93 I	-	-	-	-	-	-	-	5.37
rye-s-93	8.53	-	-	-	9.23	-	-	10.71
sta-f-83 I	151.52	141.08	163.63	159.1	157.69	159.2	158.88	160.12
tre-s-92	7.92	8.85	9	8.67	8.43	8.74	8.52	8.32
uta-s-92 I	3.14	3.54	3.62	3.56	3.33	3.32	3.21	3.88
ute-s-92	25.39	32.01	30.01	30.23	26.95	30.32	28	32.67
yor-f-83 I	36.53	40.13	42.54	43	39.63	40.24	40.71	40.53

## Appendix C – Performance of Selection Perturbative Hyper-Heuristics

Table 3. Comparison of selection perturbative hyper-heuristic performance

<b>Data</b>	<b>Kendall &amp; Hussin (2005)</b>	<b>Ersoy et al. (2007)</b>	<b>Burke et al. (2010c)</b>
car-f-92 I	4.67	-	4.31
car-s-91 I	5.37	-	5.19
ear-f-83 I	40.18	-	35.79
hec-s-92 I	11.86	11.6	11.19
kfu-s-93	15.84	15.8	14.51

lse-f-91	-	13.2	10.92
pur-s-93 I	-	-	-
rye-s-93	-	-	-
sta-f-83 I	157.38	157.7	157.18
tre-s-92	8.39	-	8.49
uta-s-92 I	-	-	3.44
ute-s-92	27.6	26.3	26.7
yor-f-83 I	-	40.7	39.47

## Appendix D – Performance of Generation Constructive Hyper-Heuristics

Table 4. Comparison of generation constructive hyper-heuristic performance

<b>Data</b>	<b>Asumni et al. (2005)</b>	<b>Asumni et al. (2007)</b>	<b>Asumni et al. (2009)</b>	<b>Pillay &amp; Banzhaf (2009b)</b>
car-f-92 I	4.56	4.51	4.54	4.28
car-s-91 I	5.29	5.19	5.29	4.97
ear-f-83 I	37.02	36.16	37.02	36.86
hec-s-92 I	11.78	11.61	11.78	11.85
kfu-s-93	15.81	15.34	15.81	14.62



lse-f-91	12.09	11.35	12.09	11.14
pur-s-93 I	-	-	-	4.73
rye-s-93	10.35	10.02	10.38	9.65
sta-f-83 I	160.75	159.09	160.75	158.33
tre-s-92	8.67	8.62	8.67	8.48
uta-s-92 I	3.57	3.52	3.57	3.4
ute-s-92	27.78	27.64	28.07	28.88
yor-f-83 I	40.66	39.25	39.80	40.74

# Abstracts

---

## Scheduling the Brazilian Football Tournament in Practice

Celso C. Ribeiro · Sebastian Urrutia

Received: January 26, 2012 / Accepted: May 16, 2012

Professional sports leagues face challenging optimization problems. Devising good tournament schedules is of utmost importance for players, teams, fans, sponsors, hosting cities, and the media. Fair and balanced schedules are a major issue for ensuring attractiveness and confidence in the tournament outcome. The annual Brazilian football tournament is a compact, mirrored, double round-robin tournament played by 20 teams. We describe the integer programming approach that has been proposed for solving the scheduling problem associated with this tournament and report on the successful practical experience after running this system for three years.

### 1 Introduction

Professional football teams do not want to waste their investments in players and structure due to poor game playing schedules. National and international competitions played in parallel require strong coordination of travel and game schedules. Professional leagues face challenging optimization problems and efficient schedules are of major interest for players, teams, fans, sponsors, and the media; see recent literature surveys in [1, 4, 6].

The annual football tournament organized by the Brazilian Football Confederation (CBF) is Brazil's most important sporting event. Its major sponsor is TV Globo, the largest media group and television network in Brazil.

Nurmi et al [3] have noticed that few professional leagues have adopted optimization models to date. This seems to be due both to the hardness of

---

Celso C. Ribeiro

Universidade Federal Fluminense, Department of Computer Science, Niterói, RJ 24210-240, Brazil. E-mail: celso@ic.uff.br

Sebastian Urrutia

Universidade Federal de Minas Gerais, Department of Computer Science, Belo Horizonte, MG 31270-010, Brazil. E-mail: surrutia@dcc.ufmg.br

the problem and to some fuzzy preference restrictions and criteria that can be hard to describe, and also to the resistance of teams and leagues to using new tools that introduce modern techniques in sports management.

This work summarizes the formulation and the implementation of the optimization software developed by the authors in partnership with CBF to determine good schedules for the two divisions of the Brazilian football tournament, extending the developments reported in [5]. We also report on the practical experience resulting from using this interactive software to schedule the 2009, 2010, and 2011 editions of the tournament.

## 2 Schedule Requirements

The annual Brazilian soccer tournament lasts for seven months, from May to December. Each division (Series A and Series B) is structured as a compact, mirrored, double round-robin tournament played by  $n = 20$  teams over  $2(n - 1) = 38$  rounds. *Weekend rounds* are played on Saturdays and Sundays, while *midweek rounds* are played on Wednesdays and Thursdays. The dates available for game playing change from year to year and must be coordinated with other competitions, such as Brazil's Cup, South America's Cup, and Libertadores Cup. Some games are required to be played on weekends.

Twelve teams form the so-called *Group of Twelve* (G12), which are the strongest founding teams of the league and have greater broadcast rights. The teams are organized by pairs with complementary home-away patterns of game playing. Teams in the same pair are usually based in the same home city.

*Regional games* involve two opposing teams whose home cities are located in the same state. *Classic* games (or derbies) are those that involve two opponents based in the same home city and with a long tradition of rivalry. They are usually the most important games and attract the largest attendances.

The tournament schedule should satisfy a number of constraints, ranging from fairness to security issues, and from technical to broadcasting criteria. Most of them reflect strategies for maximizing revenues and tournament attractiveness, while others attempt to avoid unfair game sequences that could benefit some team. These requirements fall in five classes: round-robin constraints, home-away patterns of game playing, classic and regional games, geographical and G12 constraints, and perfect matching of paired teams.

The maximization of gate attendance and TV audience is the major issue at stake. Major revenues earned by the teams come from broadcast and merchandising rights paid by the sponsors, who request good schedules that draw large audiences. Fair and balanced schedules are also a major issue for the attractiveness of the tournament and for the confidence in its outcome.

## 3 Solution Approach

The problem summarized in the previous section has been formulated as an integer programming model, see Ribeiro and Urrutia [7] for details.

We developed a three-phase solution approach based on a “first-break, then-schedule” decomposition scheme similar to that proposed in [2] to schedule a basketball league. In the first phase, we create feasible home-away patterns. In the second phase, we assign a different feasible home-away pattern to each team. Finally, in the third phase, we seek an optimal schedule by solving a simpler version of the integer programming model, obtained by variable fixations.

#### 4 Practical Experience

The optimization model and the software system have been developed, tuned, and validated over the last four years. CBF and TV Globo staff participated actively in this effort. The system was validated with data of the 2005 and 2006 A editions of the tournament.

The system was used for the first time in 2009 as the official scheduler of the Brazilian football tournament. A number of schedules have been provided to the users, who selected their preferred choice. New criteria have been added in the model along the decision process based on successive refinements of the solution, as the decision makers evaluated and filtered the different schedules. The organizers checked each proposed schedule and imposed additional constraints (or removed existing constraints) to handle specific situations that might be desired to fine-tune the schedule. This tournament was the most attractive until that year, with four teams still in contention for the title when all games in the last round simultaneously started. The title changed hands several times, as the scores of the ten games underway changed. The goal that decided the tournament for Flamengo was scored only 20 minutes before the end of the tournament. The champion was not known until the last game ended, contrary to what had happened in previous years when the winners were known many rounds before the end of the tournament, making the games of the last rounds very uninteresting.

The optimization system was used for the second time in 2010. Once again, the decision makers were happy with the schedules the system computed. This was a particularly difficult tournament to schedule. Since it had to be interrupted in June and July during the 2010 World Cup, there were few dates available for game playing. As a consequence, there were many midweek rounds and few weekend rounds, making it impossible to schedule all classic games in double weekend rounds. The system sought a schedule with a maximum number of classic games played at double weekend rounds. Once again, the title was decided in the last round, with three teams still in contention for the title when their matches started. The goal that decided the tournament for Fluminense was scored 25 minutes before the end of the tournament.

In order to increase the interest for the games played in the last rounds, it was decided to schedule all derbies in the last rounds of the 2011 edition of the tournament. Due to old local rivalries, teams give their best when playing against rivals from the same city. Adding these new constraints lead to a

harder optimization problem, for which not even feasible solutions could have been obtained without an automated system. The schedule produced by the optimizer lead to the tightest tournament of all time. Five teams were still in contention a few rounds before the last. Two draws in the most important games played in the last round decided the title for Corinthians. Any additional goal in any of these two matches could have changed the outcome of the tournament. The press was unanimous in crediting the success of the tournament to the new constraints.

## References

1. G. Kendall, S. Knust, C. C. Ribeiro, and S. Urrutia. Scheduling in sports: An annotated bibliography. *Computers & Operations Research*, 37:1–19, 2010.
2. G. L. Nemhauser and M. A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46:1–8, 1998.
3. K. Nurmi, D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Duran, J. Kyngäs, J. Marenco, C. C. Ribeiro, F. Spieksma, S. Urrutia, and R. Wolf. A framework for a highly constrained sports scheduling problem. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume III, pages 1991–1997, Hong-Kong, 2010.
4. R. V. Rasmussen and M. A. Trick. Round robin scheduling – A survey. *European Journal of Operational Research*, 188:617–636, 2008.
5. C. C. Ribeiro and S. Urrutia. Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. In *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 147–157. Springer, Berlin, 2007.
6. C.C. Ribeiro. Sports scheduling: Problems and applications. *International Transactions in Operational Research*, 19:201–226, 2012.
7. C.C. Ribeiro and S. Urrutia. Scheduling the Brazilian soccer tournament: Solution approach and practice. *Interfaces*, to appear.

---

Sej gf wlpj 'Et lengv'Wo rlt gu'Wlpj 'P gli j dqwt j qgf 'Ugct ej 'Vj g'F tco cve  
 Kō rcev'qht'Ulo rrg'Ej cpi g'lp'P gli j dqwt j qgf 'F ght'pklqp

O ctmY tk j v'

One widespread application of employee timetabling is scheduling officials for sports leagues. This can be complex, as there are frequently many objectives, preferences and constraints (hard and soft) to accommodate. While one can sometimes use optimising software for such problems, frequently heuristics are necessary. This often involves neighbourhood search.

In a cricket league, two umpires must be appointed to each match during the cricket season, which typically lasts between 16 and 20 weeks. There may be several divisions at different levels.

A computer system was developed to schedule umpires for the Devon League in south-west England, with four hierarchically organised divisions of ten teams each.

The hard constraints are:

- every match requires two umpires
- no umpire may be used on a date for which he has declared himself unavailable
- no umpire may be used for two matches on the same date
- some specific rules: for example, some umpire-team combinations are considered infeasible

Soft constraints concern the *target* number of matches for each umpire in each division over the season – it is usually possible to satisfy all targets exactly if they add up correctly for each division, but the system accepts, during the search, solutions which break soft constraints, using penalty costs.

Objectives concern travel distances and spread of umpires between teams, home grounds and each other. Further adaptations weight travel more heavily at certain times of year and encourage situations where two umpires can travel together, thus reducing costs.

The original neighbourhood definition was simple: replace one umpire by another for a specific match (a *replacement* perturbation), or exchange two umpires between two matches (a *swap* perturbation). The search technique was the variant of Simulated Annealing described in Wright (2001), using subcost information, and considering perturbations in a predefined order rather than at random. For precise details, see Wright (2007).

Recently, the system was adapted for the Home Counties League in South-East England. The structure is slightly different, with one top division and two others of equal status, organised geographically. The computer system was amended so that two divisions could be considered at an equal *level*; thus soft constraints concerned the targets for each umpire at each level. This was then implemented in practice.

While the client was happy with the results and implemented them, they contained disappointing features. In particular, on several occasions an umpire/team combination occurred twice within three weekends, which rarely happened for the Devon League, and there were other unexpectedly high costs.

Further analysis uncovered a possible reason, concerning the umpires' targets. The Devon League deliberately allows some slack for most umpires, enabling flexibility when schedules must be changed, e.g. because of illness. Thus, for example, an umpire available for 17 of the 18 weekends might be given an overall target of 15 matches. Thus there was plenty of scope for swapping umpires between dates without breaking any hard constraints.

However, mainly because of a shortage of suitable umpires, the Home Counties League uses most umpires as often as possible. Thus, once the umpires' targets have been met, swaps involving such umpires are feasible only between two matches taking place on the same date. This greatly restricts the search; moreover, if an umpire currently meets his targets there are few perturbations available which do not break these, i.e. only those between matches at the same divisional level and on the same date.

Variable Neighbourhood Search (Hansen & Mladenovic, 2001) was considered, but not implemented since it can increase neighbourhood size enormously. Instead, a new type of perturbation was defined, involving two umpires, two dates and four matches, such that the umpires swap matches on both dates (a *double* perturbation). This allows umpires to change division levels on each date without breaking any soft constraint.

Tests on Home Counties data showed that including such perturbations reduced the total cost by about 22%, averaged over 100 runs. Surprisingly, given that it had not been apparent that there was a problem, almost as large an effect was demonstrated for the Devon League, where the total cost was reduced by about 20%.

Other interesting features were apparent. About 73% of perturbations were doubles, compared with 11% for replacements and 16% for swaps. Replacement perturbations were accepted 0.27% of the time; swaps 1.06% of the time; and doubles only 0.02% of the time, which is interesting given the great improvements seen from including such perturbations. Overall, only 0.21% of perturbations were accepted, compared with 0.64% for the original system without double perturbations.

Moreover, 18.3% of accepted replacements were accepted in the first 0.05% of the run, compared with 0.61% of swaps and 0.65% of doubles; and no replacements were ever accepted in the last 1% of the run, compared with 0.053% for swaps and 0.016% for doubles. (These figures are for the Home Counties League – similar patterns were evident for the Devon League.)

This suggests that the new system's success may owe much to the presence of three distinct "meta-types" of perturbation:



1. "Get rich quick", making strong early gains, eliminating most unnecessary major costs such as soft constraint violation penalties, but of little value when fine-tuning later in the search;
2. "Steady as she goes", valuable throughout the process, especially when fine-tuning at the end;
3. "Occasional gem", only rarely useful, but capable of producing dramatic improvements.

It is hypothesised that any neighbourhood search process is more likely to produce high-quality solutions to complex problems if it contains perturbations of all three meta-types.

### References

- Hansen, P. and Mladenovic, N. (2001) "Variable Neighborhood Search: principles and applications". *European Journal of Operational Research* **130(3)**: 449-467.
- Wright, M.B. (2001) "Subcost-guided simulated annealing", in: "Essays and surveys in metaheuristics", eds. C.C. Ribeiro and P. Hansen, chapter 28, 631-639 (Kluwer Academic Publishers, Boston).
- Wright, M.B. (2007) "Case study: problem formulation and solution for a real-world sports scheduling problem", *Journal of the Operational Research Society* **58 (6)**, 439-445.

## A Column Generation Approach for Solving the Patient Admission Scheduling Problem

Troels Martin Range · Richard Martin  
Lusby · Jesper Larsen

the date of receipt and acceptance should be inserted later

**Abstract** The Patient Admission Scheduling Problem (PAS) is the problem of assigning patients to rooms during their stays in the hospital over a predefined planning period. Each room has a number of beds, which is the capacity of the room. A set of hard constraints determine whether or not a patient can stay in a room. For each patient-room combination a penalty is given for having the patient in the room. The penalty measures the inconvenience of being in the room for the given patient. Transferring patients between rooms during their stays is allowed, but this is penalized, as it is both inconvenient for the patient and takes time for the staff to organize. Finally the rooms are gender segregated, i.e. only patients of the same gender can stay in any room in the same time period. This constraint is considered a hard constraint. The problem is then to assign patients to rooms in their admission periods such that the total penalty is minimized and the hard constraints are satisfied.

PAS was proposed by Demeester et al. (2010) who solved it by a hybrid tabu search heuristic. Ceschia and Schaerf (2011) use simulated annealing to identify feasible upper bound solutions and they provide several lower bounds based on the assignment problem as well as linear programming relaxations of an integer programming model. Finally, a hyper-heuristic approach is described by Bilgin et al. (2011)

We present a Dantzig-Wolfe decomposition of PAS into a set-partitioning problem as the master problem and a set of room scheduling problems as the pricing problems. The set-partitioning problem has columns corresponding to feasible schedules for the rooms and it has two types of rows: a row for each patient-time combination stating that the patient has to be in a room in the time period and a

---

Troels Martin Range  
Department of Business and Economics, COHERE, University of Southern Denmark,  
Campusvej 55, 5230 Odense, Denmark,  
E-mail: tra@sam.sdu.dk

Richard Martin Lusby · Jesper Larsen  
Department of Management Engineering, Technical University of Denmark,  
Produktionstorvet, Building 426, 2800 Kgs. Lyngby, Denmark,  
E-mail: {rmlu,jesla}@dtu.dk

row for each room type stating that we cannot use more than the available number of rooms of that type. The room scheduling problem for a specific room has to select a number of patients in each time period such that the number of patients is not greater than the capacity of the room and the genders of the patients chosen in each time period are identical. The room schedules have to be constructed such that the reduced cost coefficients of the schedules are minimized.

We solve the pricing problems by a series of greedy heuristics as well as a new exact dynamic programming based algorithm identifying the most negative reduced cost column for a specific room type. The latter has a stage for each time period and states composed of the cost of entering the state as well as the patients present in the room. Each stage has at most  $\binom{n_t+Q}{Q}$  states, where  $n_t$  is the number of patients in period  $t$  and  $Q$  is the capacity of the room. This is exponential in  $Q$  but in our case  $Q$  never becomes more than 4. To reduce the number of states we apply several sufficient dominance criteria eliminating states which will never yield an optimal negative reduced cost column. We especially introduce pairwise patient dominance, which is used to prove state dominance. To further reduce the computation time we introduce preprocessing for the pricing problem based on shortest path calculations in two different acyclic graphs.

The master problem is severely degenerate, which makes it hard to solve the linear-programming relaxation directly by column generation. As a consequence, we apply dynamic constraint aggregation as proposed by Elhallaoui et al. (2005). This improves the performance of the column generation significantly.

We introduce several branching strategies to integerize the lower bound solution. The first is to branch on room types where a fractional number of rooms of the type is used. Next, if a room in a period has a fractional number of patients with one gender, then we branch on the gender, i.e. requiring the room to be either of this gender or not. Finally, if a patient is used fractionally in a time period for a room type, then we branch by either forcing the patient to be in the room at that time or prohibiting the patient from being in the room at that time.

The method is tested on benchmark instances described by Demeester et al. (2008) where we derive tighter lower bounds for several of the instances than previously reported. The computation times for identifying these lower bounds are in most cases significantly less than those presented by Ceschia and Schaerf (2011).

**Keywords** Patient Admission Scheduling · Decomposition · Dynamic Constraint Aggregation

## References

- Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, W., and Vanden Berghe, G. (2011). One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics*. Published on-line.
- Ceschia, S. and Schaerf, A. (2011). Local search and lower bounds for the patient admission scheduling problem. *Computers & Operations Research*, 38:1452–1463.
- Demeester, P., Bilgin, B., and Vanden Berghe, G. (2008). Patient admission scheduling benchmark instances. <http://allserv.kahosl.be/~peter/pas/>.

- Demeester, P., Souffriau, W., De Causmaecker, P., and Vanden Berghe, G. (2010). A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine*, 48:61–70.
- Elhallaoui, I., Villeneuve, D., Soumis, F., and Desaulniers, G. (2005). Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4):632–645.

# Timetabling and field assignment for training youth football teams in amateur leagues

Renatha O. Capua · Simone L. Martins ·  
Celso C. Ribeiro

Received: date / Accepted: May 16, 2012

## 1 Introduction

Professional sport leagues involve millions of fans and significant investments in players, broadcast rights, and advertising. Although amateur leagues usually do not have access to the same amounts of resources, the number of tournaments and competitors can be very large, also requiring coordination and logistical efforts [2, 4, 6, 8]. Amateur leagues of sports such as baseball and football have hundreds of games every weekend in different divisions. In a single league in California there might be up to 500 soccer games in a weekend. In the MOSA (Monmouth & Ocean Counties Soccer Association) league, New Jersey, boys and girls of ages 8 to 18 make up six divisions per age and gender group with six teams per division, totalizing 396 games every Sunday.

Amateur leagues face the problem of assigning fields and practice time to youth football teams. Players in these teams are young and are not free for training at any time of the day. They can only practice at off-school time. Low age children cannot train in the evening. Some coaches are hired by several teams, which must have compatible times and places for training. We present next the problem definition, including its constraints and objective function. This is followed by the description of a three-phase heuristic developed to find high-quality feasible solutions. Preliminary computational results are reported.

## 2 Problem definition

The problem of timetabling and field assignment for training youth football teams involves different constraints and several objectives. In this section, we describe the specific scheduling requirements addressed in this work.

---

R.O. Capua E-mail: rcapua@ic.uff.br · S.L. Martins E-mail: simone@ic.uff.br · C.C. Ribeiro E-mail: celso@ic.uff.br  
Universidade Federal Fluminense, Institute of Computing, Niterói, RJ 24210-240, Brazil.

We first describe the input data and the main assumptions. We consider a set  $T$  of teams that are coached by a set  $C$  of trainers and require training time at any of the fields in a set  $F$  of sports facilities available in a given region. Each facility has a specific number of fields available for training every weekday. The daily period of time available for training at each facility is partitioned in timeslots with the same duration (say, one hour or 90 minutes each). At the beginning of the season, each team requires some specific weekly training time. Each team may be assigned to one or more timeslots per week according to its training requirements, but at no more than one timeslot a day.

Each coach may train one or more teams in different divisions (or even in different leagues). In case two teams share the same coach, then they must be assigned to different timeslots. Due to mobility constraints, teams with the same coach cannot be scheduled for training at consecutive timeslots in different facilities. Teams and coaches may express their preferences about timeslots and facilities, and may be unavailable for training in some specific timeslots. Therefore, the main scheduling requirements are:

1. Every team must be assigned to a number of timeslots that fulfills its weekly training time.
2. No team can be assigned to a timeslot for which it is not available.
3. Teams sharing the same coach cannot be assigned to the same timeslot.
4. Teams sharing the same coach cannot be assigned to consecutive timeslots in different facilities.
5. The number of teams assigned to the same timeslot at any facility must not exceed its number of fields.
6. Each team can be assigned to at most one timeslot per day.
7. Each team must train always at the same facility and time of the day.

There are a number of objectives to be optimized. In this work, we seek to maximize coach and team preferences, assigning them as much as possible to their preferred timeslots and facilities. A second relevant objective consisting of minimizing coaches' idle time will be handled by a biobjective extension of this problem.

### 3 Solution approach

This combined timetabling [1,7] and facility assignment problem was formulated as an integer programming problem that could not be solved by standard codes in reasonable times.

Due to the hardness of the problem, we developed a three-phase heuristic to find high-quality feasible solutions. In its first phase, a constructive randomized heuristic builds an initial solution. If this solution is not feasible, then a repair procedure is applied to make it feasible. If no feasible solution is obtained, then a new attempt is made and another initial solution is built. Otherwise, an improvement heuristic is applied to the current solution. Both the repair and improvement heuristics are based on the principles of the Iterated Local Search metaheuristic [5] and follow a similar approach to that described in [2].

Each iteration of the construction phase starts by randomly selecting a coach among those with more teams. The heuristic attempts to assign timeslots and training facilities to all teams of this coach. Teams are ranked by predefined weight preferences. The next team to be handled is randomly selected from a candidate list formed by those with higher rankings. The heuristic makes an initial attempt to assign this team to its preferred timeslots. If this cannot be done, then other timeslots are considered. If this team cannot be assigned to any timeslot after some attempts, then it is discarded and the algorithm moves to another team trained by the same coach. Finally, all unassigned teams are randomly assigned to some facility and timeslot.

If this solution is not feasible, then an ILS repair procedure is applied to minimize the number of constraint violations in the incumbent.

If no feasible solution is found, then the algorithm stops. Otherwise, an ILS improvement heuristic is applied to this feasible solution. Each iteration of this phase starts by a VND [3] local search, based on three different neighborhoods: reassignment of teams to new timeslots, swap of the timeslots of teams with the same weekly requirements, and exchange of the timeslots assigned to teams with the same starting time at the same facility. Next, a perturbation consisting of reassigning half of the teams to empty timeslots is applied to the current local optimum. A new iteration resumes and the heuristic stops after a given number of iterations is performed without updating the best solution.

This approach was applied to ten test instances with up to 150 teams, eight facilities, 24 training fields, and 25 coaches. We performed ten runs for each instance. The constructive heuristic found feasible solutions for 64 out of the 100 runs. The repair heuristic obtained feasible solutions in 24 additional runs. On average, the improvement procedure increased the solution values by 9,6%. Detailed numerical results will be reported in the final version of this paper, together with additional results regarding its bicriteria version.

## References

1. Colorni, A., Dorigo, M., Maniezzo, V.: Metaheuristics for high school timetabling. *Computational Optimization and Applications* **9**, 275–298 (1998)
2. Duarte, A.R., Ribeiro, C.C., Urrutia, S., Haeusler, E.H.: Referee assignment in sports leagues. In: E.K. Burke, H. Rudová (eds.) *Practice and Theory of Automated Timetabling VI, Lecture Notes in Computer Science*, vol. 3867, pp. 158–173. Springer, Berlin (2007)
3. Hansen, P., Mladenović, N.: Variable neighbourhood search. In: F. Glover, G. Kochenberger (eds.) *Handbook of Metaheuristics*, pp. 145–184. Kluwer (2001)
4. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography. *Computers & Operations Research* **37**, 1–19 (2010)
5. Lourenço, H.R., Martin, O.C., Stutzle, T.: Iterated local search. In: F. Glover, G. Kochenberger (eds.) *Handbook of Metaheuristics*, pp. 320–353. Kluwer (2001)
6. Melo, R.A., Urrutia, S., Ribeiro, C.C.: The traveling tournament problem with predefined venues. *Journal of Scheduling* **12**, 607–622 (2009)
7. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., Lee, S.Y.: A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling* **12**, 55–89 (2009)
8. Ribeiro, C.: Sports scheduling: Problems and applications. *International Transactions in Operational Research* **19**, 201–226 (2012)

---

# Rostering RAF Air Traffic Control Personnel

R.Conmiss · T.Curtois · S.Petrovic ·  
E.Burke

Received: date / Accepted: date

**Keywords** Personnel Scheduling, Rostering, Heuristics

## 1 Introduction

Research in personnel scheduling has largely focussed on specific industries or professions. The available literature includes many examples of work done on nurse rostering [1] and aircrew scheduling [2], with a great variety of both mathematical optimisation and heuristic approaches explored [3] [4]. An interesting addition to this area is the rostering of military Air Traffic Control (ATC) personnel, mainly due to the complicated nature of the set of constraints and goals to be met. Rostering ATC personnel has some similarities with both nurse and aircrew scheduling, all are subject to shift patterns, appropriate qualifications, crew duty/working hours restrictions and rest break planning. The differences however, make this a unique problem, with some novel, challenging and interesting issues that require further exploration. Despite the military roots of operations research, very little work on military rostering appears in the literature [5], although this could be due to security restrictions. Currently, the rostering of ATC personnel is produced manually, usually by an air traffic controller as a secondary or additional duty. It is

---

R.Conmiss

A.S.A.P. Research Group, School of Computer Science, University of Nottingham, UK  
E-mail: psxrc@nottingham.ac.uk

T.Curtois

A.S.A.P. Research Group, School of Computer Science, University of Nottingham, UK  
E-mail: tim.curtois@nottingham.ac.uk

S.Petrovic

A.S.A.P. Research Group, School of Computer Science, University of Nottingham, UK  
E-mail: sxp@cs.nott.ac.uk

E.Burke

University of Stirling E-mail: e.k.burke@stir.ac.uk



obviously undesirable to use a highly qualified specialist to do what is an administrative task, and the removal of a controller from ATC duties adds additional pressure to already over committed ATC operations. As such a system which can automatically generate solutions of comparable or superior quality is highly desirable.

## 2 Problem Definition

Each ATC unit is made up of a set of control positions and controllers. Each controller holds a set of qualifications that allow them to work in one or more positions to achieve a specific ATC task. As an example, a controller can work in the Aerodrome control position, controlling all aircraft landing and departing from the runway, or operating close to the airfield and coordinating their movement with RADAR controllers in another part of the building. Each position requires at least one controller to operate it, and they must be relieved by another controller that holds a qualification for that position. This means each shift requires enough qualified controllers to operate all positions, and spare controllers that can offer breaks to their colleagues. In practice this often means that a controller will take a break from one position and return to another position to give another colleague their break. Any roster produced must offer this flexibility to give rest breaks, as controller fatigue is a flight safety issue. To obtain a qualification, a controller is designated as under training (UT) in a position, and then must be supervised by another controller who holds the qualification for the position and an additional unit instructor (UI) rating in that position. UI ratings are specific to a particular position, and not all controllers will hold UI ratings in all of their qualified positions. Also, the UT may well hold qualifications in other positions and whilst under training cannot be used by the unit to staff other positions. This burden of training is further compounded by the nature of military service, as personnel are only likely to be at a particular unit for 3-5 years. When new controllers arrive, they are not qualified in any position so must join the training queue. When experienced controllers leave, all their qualifications for that unit are cancelled, and they arrive at their next unit as unqualified controllers. The main military ATC training school produces new controllers every 6-8 weeks, and these personnel are distributed to ATC units to undertake on the job training for their first qualification controlling real aircraft. The rostering problem then becomes one of not only having to schedule suitable controllers to positions, and allowing those controllers to take reasonable rest breaks, but to do it in such a way that new controllers can be trained to maintain the effectiveness of the unit. Controllers also require leave and can fall ill, just like in nurse/aircrew scheduling, but because they are military personnel they also have additional requirements to deploy overseas, undertake duties outside of ATC and undertake regular training courses for core military skills.

### 3 Solution Approach

Some potential approaches from the literature that could be used for parts of the solution include those demonstrated in [6] and [7]. Due to the size and complexity of the overall problem, the authors elected to decompose the problem into the following four parts. Each part is then solved using a heuristic based method.

1. Shift allocation: Using heuristic techniques from Nurse Rostering, a feasible allocation of controllers to shifts is produced. An additional requirement is that a minimum number of qualifications for each position must exist.
2. Task allocation: From the shift allocation, the controllers are allocated particular tasks or positions for the shift. If this allocation is infeasible, a new shift is generated.
3. Break assignment/feasibility: Once the tasks can all be covered, the system then needs to check if breaks for all controllers on shift can be given. Some softer constraints are used here for example certain positions may not be staffed at all times and this will allow flexibility to give rest breaks. Any failure here causes a reallocation of tasks.
4. Training: Finally, training controllers are scheduled. The first simple check is the existence of a UI in a required position. Although training is a critical task for any unit, it is also the first requirement to be relaxed as the ATC task always takes priority. Similarly, during periods of low traffic levels, training is unlikely to be productive and during periods of low staff availability training is invariably cancelled. A choice can be made at this stage if training is desired to return to the task allocation stage and attempt to swap controllers to place UIs in the appropriate positions.

### 4 Planned Research

Although of significant theoretical interest, the aim of the research is to provide a robust system of rostering to RAF ATC units. Close cooperation between the authors and RAF units is required to benchmark the system and evaluate its usefulness in an operational environment <sup>1</sup>. The next stage of the research is to take data provided by partner units, and produce experimental problem instances. These can then be used to evaluate the developed heuristic-based algorithms and to fine tune the system to a point where it is deemed ready for use in a live environment. The system will then be used to generate rosters, and these can then be evaluated against the current manual approach by experienced RAF ATC personnel. With the permission of the Squadron Commander at an ATC unit, the rosters produced will then be used operationally for a fixed period of time, and their effectiveness rated by the unit command staff.

---

<sup>1</sup> The motivation for this research stems from one of the authors previous career as a RAF ATC Officer

## References

1. Burke, Edmund K., De Causmaecker, Patrick., Vanden Berghe, Greet., Van Landeghem, Hendrik., *The State of the Art of Nurse Rostering*, Journal of Scheduling, 7, 441-499 (2004)
2. Butchers, E., Day, P.R., Goldie, A.P., Miller, S., Meyer, J.A., Ryan, D.M., Scott, A.C., Wallace, C.A., *Optimized crew scheduling at air new zealand*, Interfaces, 31, 30-56 (2001)
3. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D., *An Annotated Bibliography of Personnel Scheduling and Rostering*, 127, 21-144 (2004)
4. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D., *Staff scheduling and rostering: A review of applications, methods and models*, European Journal of Operational Research, 153, 3-27 (2004)
5. Wang, Jun., *A Review of Operations Research Applications in Workforce Planning and Potential Modelling of Military Training*, Defence Science and Technology Organisation, 1-25 (2005)
6. Rekik, Monia., Cordeau, Jean-François., Soumis, François., *Implicit shift scheduling with multiple breaks and work stretch duration restrictions*, Journal of Scheduling, 13, 49-75 (2010)
7. Li, Jingpeng., Burke, Edmund K., Curtois, Tim., Petrovic, Sanja., Qu, Rong., *The falling tide algorithm: A new multi-objective approach for complex workforce scheduling*, Omega, 40, 283-293 (2012)

---

# High School Timetabling: Modeling and solving a large number of cases in Denmark

Matias Sørensen · Thomas R. Stidsen

**Keywords** High School Timetabling · Modeling · Adaptive Large Neighborhood Search

## 1 Introduction

A general model for the timetabling problem of high schools in Denmark is introduced, as seen from the perspective of the commercial system Lectio<sup>1</sup>, and an Adaptive Large Neighborhood Search (ALNS) algorithm is proposed for producing solutions. Lectio is a general-purpose cloud-based system for high school administration (available only for Danish high schools), which includes an embedded application for creating a weekly timetable. Currently, 230 high schools are customers of Lectio, and 191 have bought access to the timetabling software. This constitutes the majority of high schools in Denmark.

This large customer base entails a need for a model of the problem which is general enough to suit many different requirements, while still remain tractable by computer aided solution methods. This supports the recent trend of developing general models for timetabling problems (see Burke et al (1998); Asratian and de Werra (2002); Özcan (2005); Causmaecker and Berghe (2010); Bonutti et al (2010); Post et al (2011, 2012)). Furthermore, the timetabling problem of Danish high schools has not been formally described in the literature before. Some recent formulations of related problems from other countries include Wright (1996); Wood and Whitaker (1998); Bufé et al (2001); Melício et al (2005); Avella et al (2007); Nurmi and Kyngas (2007); Boland et al (2008); Santos et al (2010); Minh et al (2010).

---

Matias Sørensen  
MaCom A/S, DK  
Tel.: +45 3379 7900  
E-mail: ms@macom.dk

Thomas R. Stidsen  
Department of Management Engineering  
Technical University Of Denmark

<sup>1</sup> <http://www.lectio.dk> [lectio@macom.dk], developed by MaCom A/S, Vesterbrogade 48 1., DK-1620 Copenhagen V.

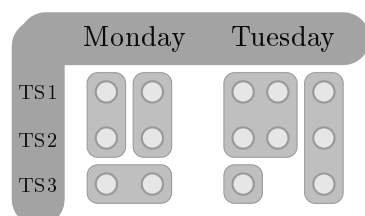
## 2 Model formulation

The following sets are given:

- Timeslots, usually made up of 5 days and 4-8 daily timeslots.
- Entities, the combined set of students and teachers.
- Classes, a subset of entities which is taught/teaching a specific topic.
- Rooms
- Events (corresponding to lectures), the basic timetabling-unit which must be assigned exactly one timeslot in the timetable.

For each class, a number of events is given (usually between 2 and 5). The basic timetabling problem concerns the assigning of each event to a timeslot, and a room to each event, such that no clashes among entities or rooms occur.

Furthermore we introduce the concept of EventChains, which separates this problem from related problems described in the literature. An EventChain consists of a subset of events, and each of these events are assigned an *offset*. At least one event must have offset 0, corresponding to the start of the EventChain. All events of an EventChain with the same offset must be placed in the same timeslot. All events of offset 1 must be assigned the timeslot following the timeslot assigned to events of offset 0, and so forth. See also Fig. 1. No restrictions are posed on which events can be included in the same EventChain, and the offsets of events in an EventChain are completely for the user to decide, providing a lot of flexibility. E.g. a double lecture can be set up by creating an EventChain consisting of two events for the same class, with offsets 0 and 1, respectively. EventChains also allows for parallel double lectures for several classes, triple lectures, grouping of elective classes in the same timeslot, etc. Many of such features have been requested by the users of Lectio, and EventChains are a pretty generic way of solving them.

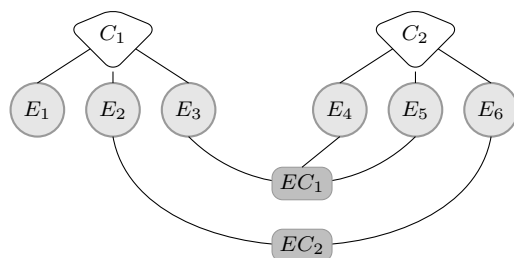


**Fig. 1** Six EventChains placed in a partial timetable.

Fig. 2 illustrates the data-model by an example with two classes, each assigned three events, and two EventChains.

Besides the no-clashes constraints, the following hard constraints are included in the model:

- An event cannot be assigned to any of its forbidden timeslots
- Each event must be assigned an admissible room



**Fig. 2** Two classes and their respective events, and two EventChains.

- Only one event of each class per day (unless otherwise specified in the given EventChain)
- For each teacher, a required number of days-off is given

The following weighted objectives (soft constraints) are used:

- Maximize the number of events assigned to a timeslot (very high priority)
- Maximize the number of events which are assigned a room (high priority)
- Maximize the number of days-off for teachers (low priority)
- Minimize idle slots for entities (medium priority)
- Minimize the amount of different rooms which are assigned to events of the same class (room stability constraint, low priority)
- Minimize the number of occurrences of two events of the same class being assigned two consecutive days (neighbor-day constraint, low priority)

### 3 Adaptive Large Neighborhood Search

ALNS is a recent extension of the Large Neighborhood Search (LNS) paradigm, often credited to Ropke and Pisinger (2006). As in the LNS framework, first a destruct (ruin/remove) operator is applied to the solution at hand, and then a construct (recreate/insert) operator is used to repair the solution. In an ALNS framework, multiple destruct and construct operators are used, and the adaptive layer keeps track of their individual performance, and increases the probability of selecting operators which have previously performed 'good'. ALNS has mainly been applied to variants of the Vehicle Routing Problem (VRP) (Azi et al (2010); Hemmelmayr et al (2011); Salazar-Aguilar et al (2011); Ribeiro and Laporte (2012)), but lately also other problem-domains (Muller and Spoorendonk (2011); Muller (2010); Kristiansen et al (2011); Kristiansen and Stidsen (2012); Sørensen et al (2012))

We propose here an ALNS heuristic for solving the described timetabling problem. The following remove- and insertion-operators are used:

- InsertGreedy: Insert EventChains in greedy way based on contribution to objective. At each insertion, also attempt to assign rooms to inserted events.

- **InsertRegretN**: This is similar to the Regret-N neighborhood applied to variants of the VRP (Tillman and Cain (1972); Martello and Toth (1981); Potvin and Rousseau (1993)). The regret-measure is specified in terms of best and second-best insert-move for a given EventChain.
- **RemoveRandom**: Select  $N$  random EventChains and unassign them.
- **RemoveRelated**: Related to Shaw operator (Shaw (1997, 1998)). Select EventChains to remove based on their similarity between classes, feasible rooms and entities.
- **RemoveTime**: Select a random timeslot, and remove EventChains assigned to it. Repeat until  $N$  EventChains has been removed.
- **RemoveClass**: Select a random class, and remove EventChains which contains it. Repeat until  $N$  EventChains has been removed.

Furthermore we apply a special set of operators, RoomRemove/Insert, which are coupled. In a coupled set of operators, the choice of remove operator implies the choice of repair operator. In RoomRemove,  $N$  random room-assignment are removed from the solution. In RoomInsert, rooms are assigned to events in a greedy way.

#### 4 Preliminary results

The Lectio database contains about 4000 datasets from 94 different high schools. Grouping these by school and year entails about 200 'unique' datasets. The authors plan to make at least some of these datasets public, most likely using the XHSTT format (Post et al (2012)).

Table 1 shows statistics and preliminary computational results for six datasets. These results show that the heuristic finds solutions where many events are unassigned, but if they are assigned a timeslot, a suitable room is also assigned. In the full paper, comprehensive computational studies will be made. This will include comparison of the found solutions with a bound provided by an integer programming model.

Name	#E.	#E.C.	#R.	#C.	#Ent.	#T.	#E./w pos.	#E./w room
Sorø2011	424	424	70	132	230	50	360.6 (13.6)	360.6 (13.6)
Skive2010	2955	1749	58	331	304	90	2358.2 (13.9)	2358.2 (13.9)
Fjerritslev2009	530	387	51	152	122	40	487.9 (2.9)	487.9 (2.9)
ViborgT2010	536	447	21	116	85	50	515.2 (1.8)	515.2 (1.8)
Herning2010	1671	1616	86	355	213	60	1545.2 (2.5)	1545.2 (2.5)
Hassersis2011	1343	1080	79	404	578	50	1331.8 (3.2)	1331.8 (3.2)

**Table 1** Preliminary results for 10 runs of ALNS heuristic on 6 datasets. Columns 2-9 shows the number of Events, EventChains, Rooms, Classes, Entities, Timeslots, Avg. events assigned to a timeslot, Avg. events assigned to a room, respectively. For columns 8 and 9, also the standard deviation is shown.

## References

- Asratian A, de Werra D (2002) A generalized class-teacher model for some timetabling problems. *European Journal of Operational Research* 143(3):531 – 542, DOI 10.1016/S0377-2217(01)00342-3
- Avella P, DAuria B, Salerno S, Vasilàev I (2007) A computational study of local search algorithms for italian high-school timetabling. *Journal of Heuristics* 13:543–556, 10.1007/s10732-007-9025-3
- Azi N, Gendreau M, Potvin JY (2010) An Adaptive Large Neighborhood Search for a Vehicle Routing Problem with Multiple Trips. CIRRELT
- Boland N, Hughes B, Merlot L, Stuckey P (2008) New integer linear programming approaches for course timetabling. *Computers & Operations Research* 35(7):2209 – 2233, DOI DOI: 10.1016/j.cor.2006.10.016, part Special Issue: Includes selected papers presented at the ECCO'04 European Conference on combinatorial Optimization
- Bonutti A, De Cesco F, Di Gaspero L, Schaerf A (2010) Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research* pp 1–12, 10.1007/s10479-010-0707-0
- Bufé M, Fischer T, Gubbels H, Häcker C, Hasprich O, Scheibel C, Weicker K, Weicker N, Wenig M, Wolfangel C (2001) Automated solution of a highly constrained school timetabling problem - preliminary results. In: Boers E (ed) *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, vol 2037, Springer Berlin / Heidelberg, pp 431–440
- Burke E, Kingston J, Pepper P (1998) A standard data format for timetabling instances. In: Burke E, Carter M (eds) *Practice and Theory of Automated Timetabling II*, Lecture Notes in Computer Science, vol 1408, Springer Berlin / Heidelberg, pp 213–222, 10.1007/BFb0055891
- Causmaecker PD, Berghe G (2010) Towards a reference model for timetabling and rostering. *Annals of Operations Research* pp 1–10, 10.1007/s10479-010-0721-2
- Hemmelmayr VC, Cordeau JF, Crainic TG (2011) An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. Tech. Rep. CIRRELT-2011-42, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation
- Kristiansen S, Stidsen TR (2012) Adaptive large neighborhood search for student sectioning at danish high schools. In: *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*
- Kristiansen S, Sørensen M, Stidsen T, Herold M (2011) Adaptive large neighborhood search for the consultation timetabling problem, to appear
- Martello S, Toth P (1981) An algorithm for the generalized assignment problem. *Operational research* 81:589–603
- Melício F, Caldeira P, Rosa A (2005) Solving real school timetabling problems with meta-heuristics. In: *Proceedings of the 4th WSEAS International Conference on Applied Mathematics and Computer Science*, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, pp 4:1–4:8
- Minh K, Thanh N, Trang K, Hue N (2010) Using tabu search for solving a high school timetabling problem. In: Nguyen N, Katarzyniak R, Chen SM (eds) *Advances in Intelligent Information and Database Systems*, Studies in Computational Intelligence, vol 283, Springer Berlin / Heidelberg, pp 305–313
- Muller L (2010) An adaptive large neighborhood search algorithm for the multi-mode resource-constrained project scheduling problem. Tech. rep., Department of Management Engineering, Technical University of Denmark Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
- Muller L, Spoorendonk S (2011) A hybrid adaptive large neighborhood search algorithm applied to a lot-sizing problem. *European Journal of Operational Research* Volume 218(Issue 3):614–623
- Nurmi K, Kyngas J (2007) A framework for school timetabling problem. In: *Proceedings of the 3rd multidisciplinary international scheduling conference: theory and applications*, pp 386–393



- Post G, Kingston JH, di Gaspero L, McCollum B, Schaerf A (2011) The third international timetabling competition (itc2011), <http://www.utwente.nl/ctit/hstt/itc2011/>.
- Post G, Ahmadi S, Daskalaki S, Kingston J, Kyngas J, Nurmi C, Ranson D (2012) An xml format for benchmarks in high school timetabling. *Annals of Operations Research* 194:385–397
- Potvin JY, Rousseau JM (1993) A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research* 66(3):331 – 340
- Ribeiro GM, Laporte G (2012) An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research* 39(3):728 – 735, DOI 10.1016/j.cor.2011.05.005
- Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40:455–472
- Salazar-Aguilar M, Langevin A, Laporte G (2011) An adaptive large neighborhood search heuristic for a snow plowing problem with synchronized routes. In: Pahl J, Reinert T, Voss S (eds) *Network Optimization*, Lecture Notes in Computer Science, vol 6701, Springer Berlin / Heidelberg, pp 406–411
- Santos H, Uchoa E, Ochi L, Maculan N (2010) Strong bounds with cut and column generation for class-teacher timetabling. *Annals of Operations Research* pp 1–14, 10.1007/s10479-010-0709-y
- Shaw P (1997) A new local search algorithm providing high quality solutions to vehicle routing problems
- Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget JF (eds) *Principles and Practice of Constraint Programming — CP98*, Lecture Notes in Computer Science, vol 1520, Springer Berlin / Heidelberg, pp 417–431
- Sørensen M, Kristiansen S, Stidsen TR (2012) International timetabling competition 2011: An adaptive large neighborhood search algorithm. In: *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*
- Tillman FA, Cain TM (1972) An upperbound algorithm for the single and multiple terminal delivery problem. *Management Science* 18(11):664 – 682
- Wood J, Whitaker D (1998) Student centred school timetabling. *The Journal of the Operational Research Society* 49(11):1146–1152
- Wright M (1996) School timetabling using heuristic search. *The Journal of the Operational Research Society* 47(3):347–357
- Özcan E (2005) Towards an xml-based standard for timetabling problems: Ttml. In: Kendall G, Burke EK, Petrovic S, Gendreau M (eds) *Multidisciplinary Scheduling: Theory and Applications*, Springer US, pp 163–185

## Adaptive Large Neighborhood Search for Student Sectioning at Danish high schools

Simon Kristiansen · Thomas R. Stidsen

**Keywords** Student Sectioning · Elective Course Planning · Adaptive Large Neighborhood Search · Educational Timetabling · High School Planning · Metaheuristics

**Mathematics Subject Classification (2000)** 90-08 · 90B35 · 90C10 · 90C59

### 1 Introduction

Student Sectioning is one of the less studied subjects within Educational Timetabling (Pillay (2010)). Student Sectioning is normally used at universities (e.g. Müller and Murray (2010) and Cheng et al (2003)), whereas this paper is concerned high schools in Denmark. I.e. a more generalized model is needed since this problem should be applicable for approximately 200 different high schools. The concerned problem is also known as Elective Course Planning Problem (ECP), described in Kristiansen et al (2011b). However in this paper we will also try to pack the students more convenient in the classes. I.e. minimize the number of common classes in each class and to have a more even distribution between classes of same course. The ECP is a preceding planning problem of the actual High School Timetabling in Denmark. The International Timetabling Competition 2011 was devoted to High School Timetabling (see e.g. Post et al (2012) and Sørensen et al (2012)).

The students request some elective courses, and the problem is then to assign course classes to time slots and then assign students to the classes given their requests. We want to maximize the number of fulfilled requests and to minimize the number of classes. The problem is to both please the students and to insure good economic. It cost approximately €27.000 p.a. to create a class, and as the high schools are self-governing, ECP is a crucial problem. If the requests are not granted, the students might change school and as the high school are payed upon the number of students, they will lose revenue.

	Monday	Tuesday	Wednesday	Thursday	Friday
8:15 9:45	Block1			Block3	
10:00 11:30					
Lunch break					
12:00 13:30					
13:45 15:15		Block2		Block4	Block5

**Fig. 1:** An example of a weekly schedule with four modules each day and a total of five time slots for elective courses. The white time slots are used for mandatory courses.

As mentioned it is also desired to have an even distribution on the number of students in classes of same course. E.g. 40 students requests the same given course, and with 28 as upper bound on the class size, we aim at having a distribution of 20-20.

## 2 Integer programming model

The ECPP is formulated as an IP model. A high school has a set of students  $s \in \mathcal{S}$ , a set of offered courses  $e \in \mathcal{E}$ , a set of classes  $c \in \mathcal{C}$  and a set of time slots  $b \in \mathcal{B}$ . Each students is assigned to a common class,  $q \in \mathcal{Q}$ , this is denoted by  $I_{q,s} \in \{0, 1\}$ . Each course belongs to one of the course subjects given by the set  $f \in \mathcal{F}$ . The maximum number of classes of each subject in a time slot is given by  $M_f \in \mathbb{R}^+$ .  $K_{c,f} \in \{0, 1\}$  denotes whether course  $c$  is teaching subject  $f$  or not. For each course class there exist an upper bound on the class sizes,  $U_c \in \mathbb{R}^+$ .  $A_{c,s} \in \mathbb{R}^+$  indicates whether student  $s$  is locked to course class  $c$ . I.e. the student must be assigned to the given course class.  $R_{e,s} \in \{0, 1\}$  indicates whether student  $s$  has requested course  $e$  or not.  $D_{c,e} \in \{0, 1\}$  denotes whether class  $c$  is teaching course  $e$ , or not. The total maximum number of classes which can be created is given by  $P \in \mathbb{R}^+$ . The parameters  $J_{c,c'} \in \{0, 1\}$  and  $H_{c,c'} \in \{0, 1\}$  indicates whether two classes cannot be placed in the same time slot or should be placed in the same time slot, respectively. The decision whether student  $s$  is assigned to class  $c$  in block  $b$  is defined by  $x_{c,b,s} \in \{0, 1\}$ , while the decision whether course class  $c$  is assigned to time slot  $b$  is given by  $y_{c,b} \in \{0, 1\}$ . The binary variable  $z_{c,q} \in \{0, 1\}$  takes value 1 if common class  $q$  is present in class  $c$ . The variables  $w_{c,c'}^+ \in \mathbb{N}_0$  and  $w_{c,c'}^- \in \mathbb{N}_0$  counts the difference of the number of students between classes of same course. The objectives are weighted in respect to each other, given by  $\alpha_{c,s}$ ,  $\beta_c$ ,  $\gamma$  and  $\delta$

$$\max \sum_{c,b,s} \alpha_{c,s} \cdot x_{c,b,s} - \sum_{c,b} \beta_c \cdot y_{c,b} - \gamma \cdot \sum_{c,q} z_{c,q} - \delta \cdot \sum_{c,c'} (w_{c,c'}^+ + w_{c,c'}^-) / 2 \quad (1)$$

$$\text{s.t.} \quad \sum_{c,b,s} x_{c,b,s} \leq 1 \quad \forall b, s \quad (2)$$

$$\sum_{c,b} y_{c,b} \leq 1 \quad \forall c \quad (3)$$

$$\sum_c \sum_b x_{c,b,s} \cdot D_{c,e} \leq R_{e,s} \quad \forall e, s \quad (4)$$

$$\sum_{c,b} x_{c,b,s} \leq U_c \quad \forall c, b \quad (5)$$

$$\sum_b \sum_s I_{q,s} \cdot x_{c,b,s} \leq z_{c,q} \quad \forall c, q, \sum_s A_{c,s} = 0 \quad (6)$$

$$\sum_s x_{c,b,s} - \sum_s x_{c',b,s} = w_{c,c'}^+ - w_{c,c'}^- \quad \forall c, c', b, e, D_{c,e} = D_{c',e} = 1, \sum_s A_{c,s} = \sum_s A_{c',s} = 0 \quad (7)$$

$$x_{c,b,s} \leq y_{c,b} \quad \forall c, b, s, A_{c,s} = 0 \quad (8)$$

$$x_{c,b,s} = y_{c,b} \quad \forall c, b, s, A_{c,s} = 1 \quad (9)$$

$$\sum_{c,b} y_{c,b} \leq P \quad (10)$$

$$y_{c,b} + y_{c',b} \leq 1 \quad \forall c, c', b, s, J_{c,c'} = 1 \quad (11)$$

$$y_{c,b} = y_{c',b} \quad \forall c, c', b, s, H_{c,c'} = 1 \quad (12)$$

$$\sum_c K_{c,f} \cdot y_{c,b} \leq M_f \quad \forall b, f, M_f > 0 \quad (13)$$

$$x_{c,b,s} \in \{0, 1\} \quad (14)$$

$$y_{c,b} \in \{0, 1\} \quad (15)$$

$$z_{c,q} \in \{0, 1\} \quad (16)$$

$$w_{c,c'}^+ \in \mathbb{N}_0 \quad (17)$$

$$w_{c,c'}^- \in \mathbb{N}_0$$

The constraints (2) ensure that no students is assigned more than one course class in each time slot, while constraints (3) ensure that a class cannot be assigned to more than one time slot. Constraints (4) ensure that students can only be assigned course classes which they requested, and that each requests is only granted once. Constraints (5) sets the upper bound on the number

of students in a course class. Constraints (6) are counting the number of common classes used while constraints (7) are used for equal distribution of the students in classes of same course. Constraints (8) is the connection between the two variables  $x_{c,b,s}$  and  $y_{c,b}$  and make sure that a student cannot be assigned a class which is not yet assigned to a time slot. Constraints (6) and (7) are soft constraints. Constraints (9) shall ensure that if students are locked to a course class and the class is assigned a timeslot, the students should be assigned to the class. Constraints (10) ensures that the total number of created classes does not exceed maximum. Constraints (11) ensure that classes which cannot be placed in same time slot are not done so and constraints (12) ensure that classes which should be placed in same time slot are satisfied. Finally, constraints (13) make sure that the resource limit on subjects  $f$  are respected.

### 3 Solution method

It has been chosen to attempt Adaptive Large Neighborhood Search (ALNS) to establish solutions to the ECPP. ALNS was first applied and is still mainly used on Vehicle Routing Problems (Azi et al (2010); Laporte et al (2010); Salazar-Aguilar et al (2011); Ribeiro and Laporte (2012)). It has however also been applied on a few other problems such as Lot-sizing (Muller and Spoorendonk (2011)) and Scheduling problems (Muller (2010); Kristiansen et al (2011a); Sørensen and Stidsen (2012)). Pisinger and Ropke (2010) is recommended for additional reading on ALNS. The pseudo code is given in Algorithm 1.

---

#### Algorithm 1: Adaptive Large Neighborhood Search

---

**Input:** a feasible solution  $x_{g,b}^s$ ,  $q \in \mathbb{N}$

- 1 solution  $x_{best} = x$ ;  $\pi = (1, \dots, 1)$
- 2 **repeat**
- 3      $x' = x$
- 4     select destroy and repair methods  $d \in \Omega^-$  and  $r \in \Omega^+$  using  $\pi$
- 5     remove  $q$  requests from  $x'$  using  $d$
- 6     reinsert removed requests into  $x'$  using  $r$
- 7     **if**  $c(x') > c(x_{best})$  **then**
- 8          $x_{best} = x'$
- 9     **if**  $accept(x', x)$  **then**
- 10          $x = x'$
- 11     update  $\pi$
- 12 **until** *stop-criterion met*
- 13 **return**  $x_{best}$

---

The neighborhoods are implicitly defined by several destroy and repair methods. In each iteration, a destroy and a repair method is chosen upon some performance indicators which is updated after each iteration.

For this implementation of ALNS two different types of moves are used: Assign/unassign a class with students to/from a time slot and assign/unassign a student to/from an assigned class. Based on these moves a total of 8 different destroy methods and 4 repair methods are implemented. The destroy methods are simple random removal heuristics and Shaw heuristics (Shaw (1997)), where less or more related classes are removed from the solution. The repair methods are basic greedy algorithms and regret heuristics (Potvin and Rousseau (1993)), which aims at inserting the request which we will regret most if not inserted immediately.

## 4 Results

The algorithm is implemented in Lectio <sup>1</sup>, and is hence available for use for approximately 200 different high schools in Denmark. This gives the possibilities for a huge amount of data for further testing and research. Table 1 shows the size and the computational results from 7 different datasets

	No.of student	No.of requests	No.of courses	No.of blocks	Assigned classes	Assigned requests
Vejen	382	586	29	3	36	586
Silkeborg	927	1789	65	5	77	1786
Falkoner	421	1080	49	4	66	1080
Vordingborg	415	1462	61	5	68	1462
Alssund	385	650	31	5	34	645
Holstebro	345	567	18	5	29	567
Frederikssund	159	273	18	4	18	273

**Table 1:** Results for a given set of real-life problems at Danish high schools.

## References

- Azi N, Gendreau M, Potvin JY (2010) An Adaptive Large Neighborhood Search for a Vehicle Routing Problem with Multiple Trips. CIRRELT
- Cheng E, Kruk S, Lipman M (2003) Flow formulations for the student scheduling problem. In: Burke E, De Causmaecker P (eds) Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science, vol 2740, Springer Berlin / Heidelberg, pp 299–309
- Kristiansen S, Sørensen M, Stidsen T, Herold M (2011a) Adaptive large neighborhood search for the consultation timetabling problem, to appear
- Kristiansen S, Sørensen M, Stidsen TR (2011b) Elective course planning. European Journal of Operational Research 215(3):713 – 720, DOI 10.1016/j.ejor.2011.06.039
- Laporte G, Musmanno R, Vocaturo F (2010) An adaptive large neighbourhood search heuristic for the capacitated arc-routing problem with stochastic demands. Transportation Science 44(1):125–135
- Müller T, Murray K (2010) Comprehensive approach to student sectioning. Annals of Operations Research 181:249–269
- Muller L (2010) An adaptive large neighborhood search algorithm for the multi-mode resource-constrained project scheduling problem. Tech. rep., Department of Management Engineering, Technical University of Denmark Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
- Muller L, Spoorendonk S (2011) A hybrid adaptive large neighborhood search algorithm applied to a lot-sizing problem. European Journal of Operational Research Volume 218(Issue 3):614–623
- Pillay N (2010) An overview of school timetabling research. In: Proceedings of the International Conference on the Theory and Practice of Automated Timetabling, Belfast, United Kingdom, pp 321–335
- Pisinger D, Ropke S (2010) Large neighborhood search. In: Gendreau M, Potvin JY (eds) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol 146, Springer US, pp 399–419
- Post G, Gaspero LD, Kingston JH, McCollum B, Schaerf A (2012) The third international timetabling competition. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), Son, Norway
- Potvin JY, Rousseau JM (1993) A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research 66(3):331 – 340
- Ribeiro GM, Laporte G (2012) An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research 39(3):728 – 735, DOI 10.1016/j.cor.2011.05.005
- Salazar-Aguilar M, Langevin A, Laporte G (2011) An adaptive large neighborhood search heuristic for a snow plowing problem with synchronized routes. In: Pahl J, Reiners T, Voss S (eds) Network Optimization, Lecture Notes in Computer Science, vol 6701, Springer Berlin / Heidelberg, pp 406–411
- Shaw P (1997) A new local search algorithm providing high quality solutions to vehicle routing problems
- Sørensen M, Stidsen TR (2012) High school timetabling: Modeling and solving a large number of cases in denmark. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)
- Sørensen M, Kristiansen S, Stidsen TR (2012) International timetabling competition 2011: An adaptive large neighborhood search algorithm. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)

<sup>1</sup> Lectio is developed by MaCom A/S and is a cloud-based high school administration, which handles all sorts of administrative tasks for the high schools, including a GUI and a heuristic-based solver for the ECPP.

---

## Investigation of fairness measures for nurse rostering

Pieter Smet · Simon Martin · Djamila  
Ouelhadj · Ender Özcan · Greet Vanden  
Berghe

Received: date / Accepted: date

**Abstract** The nurse rostering problem presents a combinatorial optimisation problem in which shifts must be assigned to nurses who are subject to a large number of workforce related constraints. In the literature on nurse rostering, the fairness of the constructed rosters has often been neglected. Solutions are typically evaluated by means of a weighted sum objective function which does not explicitly account for the fair distribution of individual high quality rosters. The present contribution aims at incorporating fairness measures in existing solution methods for the nurse rostering problem. Preliminary experimental results show that fairer solutions are obtained when applying the new fairness measures.

**Keywords** Nurse rostering · Fairness · Evaluation

### 1 Introduction

Several studies have shown that high quality work rosters contribute to the job satisfaction of nurses. This is an important result, because by increasing the job satisfaction of nurses, their retention rate is also likely to increase. Factors that influence the quality of a roster include the working hours and whether

---

P. Smet and G. Vanden Berghe  
CODeS, KAHO St.-Lieven,  
Gebr. De Smetstraat 1, 9000 Gent, Belgium  
E-mail: {Pieter.Smet, Greet.VandenBerghe}@kahosl.be

S. Martin and D. Ouelhadj  
Logistics and Mathematics Management Group,  
University of Portsmouth, Department of Mathematics, UK  
E-mail: {Simon.Martin, Djamila.Ouelhadj}@port.ac.uk

E. Özcan  
Automated Scheduling, Optimisation and Planning Research Group,  
University of Nottingham, Department of Computer Science, UK  
E-mail: Ender.Ozcan@nottingham.ac.uk

or not nurses can work a minimum number of consecutive days. Furthermore, we assume that a fair distribution of work also attributes to a higher rate of satisfaction with regard to the roster. We present a novel approach that tries to guarantee a fair distribution of individual rosters among nurses in automatically generated solutions.

In the literature, several automated approaches have been proposed as decision support systems for nurse rostering. [2] describe an auction based self rostering system in which nurses have a number of points they can use to bid for shifts or days off. After the bidding phase, an algorithm determines the winner of each auction and tries to find a feasible schedule which includes the winning bids. However, fairness is not guaranteed since experienced nurses can easily misuse the auction system such that they always make the winning bids. [5] present an automated preference scheduling approach in which nurses can request particular shifts or days off. Linked to each request is a grade indicating its importance. All requests are passed on to an algorithm that produces a feasible schedule, while respecting the preferences as much as possible. The system explicitly attends to fairness by means of an extra variable in the objective function. This variable is used to maximise the number of requests of the least favoured nurse, thus taking into account the worst individual roster. Other automated approaches often model fairness as balance constraints on working time [1].

Nurse rostering problems can be represented as constraint optimisation problems using 5-tuples  $\langle N, D, S, K, C \rangle$  with  $N$  the set of nurses,  $D$  the set of days in the planning period and all relevant days in the previous and next planning periods,  $S$  the set of shift types,  $K$  the set of skill types and  $C$  the set of constraints. Solutions are typically evaluated using a weighted sum of soft constraint violations (Equation 1). This objective function has the advantage that it is both easy to understand and to implement. However, algorithms optimising  $WO$  can generate unfair solutions in which bad individual rosters are compensated by other high quality rosters. These solutions will have a high overall quality but also an unfair distribution of individual rosters. We propose an alternative for  $WO$  which better guarantees fair solutions and can be easily added to models for nurse rostering.

$$WO = \sum_{\forall n \in N} \sum_{\forall c \in C} \#violations_{n,c} \times weight_c \quad (1)$$

## 2 Fairness objective

Inspired by the approach of [5], we propose the objective function  $FO$  (Equation 2), whereby the quality of the worst individual roster determines the overall solution quality. In doing so, nurses' rosters will not be improved at the expense of the worst individual roster.

$$FO = \max_{\forall n \in N} \sum_{\forall c \in C} \#violations_{n,c} \times weight_c \quad (2)$$

When optimising  $FO$ , the nurse rostering problem becomes a min-max problem. For algorithms that performed well when optimising  $WO$ , it will be more difficult to find good solutions. Algorithms must deal with a trade-off between a fair distribution of individual rosters and the overall solution quality. In the next section we present preliminary experimental results showing the effect of  $FO$  on the fairness and overall quality of solutions.

### 3 Experimental results

We define the individual roster quality of each nurse  $n$   $q_n$ , the average individual roster quality  $\mu$  and the standard deviation  $\sigma$ . The relative standard deviation  $RSD$  is defined as the ratio of  $\sigma$  on  $\mu$ , and allows for comparison of fairness for different scenarios. Low values for  $RSD$  indicate fair solutions, in which little variance exists between individual roster quality. The relative quality gap between the best and the worst individual roster is defined as  $diff$ . A relatively small difference between the best and worst individual roster is indicated by low values for  $diff$ . The overall solution quality calculated with  $WO$  is defined as  $q_{solution}^{WO}$ . This metric allows us to compare the overall quality of solutions obtained with different objective functions.

Experiments are performed on real world data collected from four different wards in two Belgian hospitals. For each ward, two scenarios are considered: one in which each nurse has the same contract and one in which each nurse has a personalised contract. The hyper-heuristic approach presented in [6] is used to find solutions. Each run is repeated ten times, with computation time limited to ten minutes.

Table 1 shows that fairer solutions are obtained when optimising  $FO$ . For all instances, solutions obtained with  $FO$  have lower values for  $RSD$  than those obtained with  $WO$ . This means that the quality of individual rosters varies less when using  $FO$ , thus producing fairer solutions. The reported values of  $diff$  support this result. The difference between the worst and best individual roster is smaller when optimising  $FO$ . From the results in Table 1 it can also be concluded that, for both objectives, it is easier to find fairer solutions when all nurses have identical contracts. Looking at the overall solution quality  $q_{solution}^{WO}$ , there exists no clearly identifiable trend. In some cases it is clear that the new objective makes it harder for the hyper-heuristics to find good solutions. However, the opposite is also true for some instances. These inconsistent results require further investigation into the structure of each problem instance (e.g. which constraints are present).

### 4 Discussion

Preliminary experimental results show that fairer solutions are obtained when optimising  $FO$  than when optimising  $WO$ . Looking at the overall solution quality, a trade-off exists between the fairness of a solution and its overall



Instance	<i>RSD</i>		<i>diff</i>		$q_{solution}^{WO}$		
	WO	FO	WO	FO	WO	FO	gap
Emergency-i	19,24%	7,40%	54,00%	24,11%	192569	169096	-12,19%
Emergency-d	19,10%	8,35%	62,02%	30,91%	215297	167200	-22,34%
Geriatrics-i	32,23%	16,07%	65,08%	45,93%	35930	46986	30,77%
Geriatrics-d	52,53%	27,67%	79,37%	66,69%	53718	69733	29,81%
Psychiatry-i	15,97%	10,75%	47,18%	42,56%	147960	145491	-1,67%
Psychiatry-d	35,44%	29,15%	66,91%	65,20%	129340	125936	-2,63%
Reception-i	55,20%	35,94%	79,18%	59,42%	80909	103832	28,33%
Reception-d	60,40%	38,70%	95,41%	89,24%	57749	61478	6,46%

**Table 1** Results for the centralised approach. Instance-i refers to cases with identical contracts, instance-d refers to cases with different contracts.

solution quality. However, this result is not consistent for all instances under study and deserves further investigation. One obvious future step incorporates optimising *FO* while at the same time attempting to improve *WO*, without decreasing the quality of the worst individual roster. Furthermore, decentralised approaches, e.g. agent-based frameworks [3], present other interesting possibilities for defining new fairness measures in which individual nurses' objectives can be optimised at the same time [4].

**Acknowledgements** This research was jointly carried out within the IWT project (IWT 110257) and the research sabbatical sponsored by the University of Portsmouth, Department of Mathematics.

## References

1. E. K. Burke, P. De Causmaecker, G. Vanden Berghe and H. Van Landeghem, The state of the art of nurse rostering, *Journal of Scheduling*, 7(6):441-499 (2004)
2. M. L. De Grano, J. D. Medeiros, and D. Eitel, Accommodating individual preferences in nurse scheduling via auctions and optimization, *Health Care Management Science*, 12(3):228-242 (2009)
3. S. Martin, D. Ouelhadj, P. Beullens and E. Özcan, A generic agent-based framework for cooperative search using pattern matching and reinforcement learning, Technical report 5861, University of Portsmouth (2012)
4. D. Ouelhadj, S. Martin, P. Smet, E. Özcan and G. Vanden Berghe, Fairness in nurse rostering, Technical report, KAHO Sint-Lieven (2012)
5. E. Rönnberg and T. Larsson, Automating the self-scheduling process of nurses in Swedish healthcare: a pilot study, *Health Care Management Science*, 13:35-53 (2010)
6. P. Smet, B. Bilgin, P. De Causmaecker and G. Vanden Berghe, Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 1-24 (2012)

---

# Patient Admission Scheduling with Operating Room Constraints

Sara Ceschia · Andrea Schaerf

**Abstract** We propose an extension of the PAS problem described in our previous work [6], which considers also constraints about the utilisation of operating rooms for patients that have to undergo a surgery. We design a solution approach based on local search, which explores the search space using complex neighbourhood operators.

**Keywords** Patient Admission · Operating Room · Local Search

## 1 Introduction

The *Patient Admission Scheduling* (PAS) consists in scheduling patients into hospital rooms in such a way to maximise medical treatment effectiveness, management efficiency, and patients' comfort.

The problem has been formalised by Demeester *et al* [8] and further studied by the same research group [2]. We have solved the PAS by local search [5] and we have obtained the best known solutions on all the available PAS benchmarks [7]. We have subsequently refined and extended the problem formulation to take into account several additional real-world features, such as the presence of emergency patients, uncertainty in stay lengths, and the possibility to delay admissions [6].

In this work, we present an extension of the PAS problem described in [6], that constitutes a first step towards the integration of the patient admission scheduling with the surgery scheduling process.

As pointed out in [3, 4], the operating room is a very critical resource, so that the scheduling of surgeries has a deep impact on the other activities of the hospital. Indeed, a patient that undergoes a surgery is expected to

---

S. Ceschia and A. Schaerf  
DIEGM, University of Udine  
Via delle Scienze 206  
E-mail: {sara.ceschia, schaerf}@uniud.it

recover in the hospital for a period, thus she/he needs a bed in a ward of the corresponding specialty. As a consequence, the surgical process scheduling requires to be integrated with the other hospital operations in order to avoid resource conflicts.

The operating room scheduling is usually decomposed in three hierarchical stages (see e.g., [1, 10, 12]). The first level corresponds to a long-term strategic planning whose aim is to assign the operating room resources to surgical specialisms according to historical data or forecasts. The second step is a tactical medium-term planning; it involves the development of the master surgical schedule (MSS) which is a cyclic timetable that defines, for each day of the cycle, which operating room is assigned to a team of surgeons and for how long. The last stage (see, e.g., [11]) is the daily scheduling of the specific intervention of each patient: Emergencies and all kinds of uncertainty are tackled at this level, which is mainly operational.

This work can be positioned between the tactical and the operational level, so that inputs of the problem are all data about departments, rooms and patients, and the MSS resulting from the tactical planning. Using the classification in [9], the problem belongs to the class of “Block scheduling”, because the operating room scheduling is organised in blocks assigned to specialisms.

A solution of the problem is an assignment of patients to beds in rooms for each day of their stay in hospital, satisfying all the constraints of the PAS problem (capacity, gender, specialty, age, room feature, preferences) and taking also into account the constraints related to the MSS.

## 2 PAS with operating room constraints

In order to integrate all the information about the MSS in the PAS problem, we introduce the notions of *operating room slot* and *medical treatment*:

**Operating Room Slot:** An operating room (OR) slot is the smallest time unit for which a operating room can be reserved for a specialty in a day. Each day of the planning cycle, the MSS assigns to a specialty an integer number of OR slots. Different operation rooms can be simultaneously used by the same specialty, since a specialty normally does not identify a specific surgeon but a surgical group.

**Medical Treatment:** Each patient has to undergo a medical treatment that is performed by a specialist. Some treatments include a surgery of the type of the corresponding specialty. In this case, the day of the surgery (typically either the day after the admission day or the admission day itself) and the expected length of the surgery must be specified.

Our problem consists in assigning a room to each patient for a number of days equal to her/his stay period, starting in a day not before his/her planned admission. The assignment is subject to all the PAS constraints defined in [6] and the following additional one:

**Operating Room Utilisation:** For each day and each specialty the total length of the operations for treatments belonging to the specialty must not exceed the time granted to it by the MSS.

This is a hard constraint, and it has a critical impact on the search space. Consequently it requires to reconsider the search method, and, in particular, the definition of the neighbourhood structures.

We remark that we only define the admission day of a patient taking into account the daily utilisation of the operating rooms; the problem of sequencing OR slots in a day and the surgeries within each OR slot is outside the scopes of this work, and is usually performed online (at most the day before), considering also emergency patients.

### 3 Solution technique

We refine the solution technique applied in [6], which is based on a complex neighbourhood structure and on simulated annealing (SA).

A state in the search space is composed by two integer-valued vectors: the first one represents the room assigned to each patient, and the second one is the possible delay of the admission of the patient.

In order to effectively explore the search space in which the admission of patients has to satisfy also the operating room utilisation constraint, we need to define new neighbourhoods. For example, we should consider that a specialty might appear in the MSS only in some specific days, therefore the neighbourhood relation defined in [6], that moves the admission of a patient only to the previous or following day, is not suitable. We thus have to define a neighbourhood that enables longer time leaps in one single move.

Based on these considerations, the neighbourhood is the composition of four basic moves. The first two work on the room assignment, and they are the change of the room assigned to a patient and the swap of the assigned room between two patients. The third and fourth are used to modify the admissions day of patients. One move shifts the admission of a patient forward or backward by a given number of days. The last one swaps the admission days of two patients. Both keep the room unchanged.

The cost function includes the same cost components than those of the PAS problem, but the component *distance to feasibility* takes into account not only the room capacity constraint but also the operating room utilisation.

### 4 Experimental analysis

Unfortunately, there are no real-world instances available at present for this problem, therefore we modified the instance generator designed in [6], which is available at <http://satt.diegm.uniud.it/index.php?page=pasu>. We included all the input data about the MSS, that is the number of OR slots

reserved to each specialty for each day and the features of each medical treatment.

The project is ongoing and the computational results on the generated instances are very preliminary; they will be presented in the forthcoming complete version of the paper.

## References

1. J Beliën and E Demeulemeester. Building cyclic master surgery schedules with leveled resulting bed occupancy. *European Journal of Operational Research*, 176(2):1185–1204, 2007.
2. Burak Bilgin, Peter Demeester, Mustafa Misir, Wim Vancroonenburg, and Greet Vanden Berghe. One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics*, 2011. Online first <http://dx.doi.org/10.1007/s10732-011-9192-0>.
3. J T Blake and M W Carter. Surgical process scheduling: a structured review. *Journal of the Society for Health Systems*, 5(3):17–30, 1997.
4. Brecht Cardoen, Erik Demeulemeester, and Jeroen Beliën. Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, 201(3):921–932, March 2010.
5. Sara Ceschia and Andrea Schaerf. Local search and lower bounds for the patient admission scheduling problem. *Computers & Operations Research*, 30:1452–1463, 2011.
6. Sara Ceschia and Andrea Schaerf. Modeling and solving the dynamic patient admission scheduling problem under uncertainty. Submitted for publication. Available at <http://satt.diegm.uniud.it/index.php?page=pasu>. An abstract appeared in *Proceedings of the 9th Metaheuristics International Conference (MIC 2011)*, February 2012.
7. P. Demeester. Patient admission scheduling website. <http://allserv.kahos1.be/~peter/pas/>, 2009. Viewed: February 17, 2012.
8. Peter Demeester, Wouter Souffriau, Patrick De Causmaecker, and Greet Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine*, 48(1):61–70, January 2010.
9. A. Jebali, A. Hadjalouane, and P. Ladet. Operating rooms scheduling. *International Journal of Production Economics*, 99(1-2):52–62, January 2006.
10. Jeroen M. Oostrum, Eelco Bredenhoff, and Erwin W. Hans. Suitability and managerial implications of a master surgical scheduling approach. *Annals of Operations Research*, 178(1):91–104, 2009.
11. Atle Riise and E. K. Burke. Local search for the surgery admission planning. *Journal of Heuristics*, 17:389–414, 2011.
12. J.M.H. Vissers, J.W.M. Bertrand, and G. De Vries. A framework for production control in health care organizations. *Production Planning & Control*, 12(6):591–604, January 2001.

---

## Timetabling of sorting slots in a logistic warehouse

Antoine Jouglet · Dritan Nace ·  
Christophe Outteryck

Received: date / Accepted: date

**Abstract** We study a problem appearing at the end of a logistic stream in a warehouse and which concerns the timetabling of sorting slots necessary to accommodate the prepared orders before being dispatched. We consider a set of orders to be prepared on some preparation shops over a given time horizon. Each order is associated with the truck handling its transportation to its customer. A sorting slot is an accumulation area where processed orders await to be loaded on a truck. To a given truck, a known number of sorting slots is needed from the time the first order associated with the truck begins to be prepared, and this, until the known departure time of the truck. Since several orders associated with different trucks are processed simultaneously and since the number of sorting slots is limited, the timetabling of these resources is necessary to ensure that all orders can be processed over the considered time horizon without violating the resource constraint on sorting slots. In this talk, we describe the general industrial context of the problem and we formalize it. We state that some particular cases of the problem are polynomially solvable while the general problem is NP-complete. Then, we propose optimization methods to solve it.

**Keywords** warehouse management · sorting slots · scheduling · timetabling

---

Antoine Jouglet, Dritan Nace  
Université de Technologie de Compiègne, UMR CNRS 7253 Heudiasyc  
60200 Compiègne, France  
E-mail: {antoine.jouglet,dritan.nace}@utc.fr

Christophe Outteryck  
a-sis, 8, rue de la Richelandière, 42000 Saint-Etienne, France  
E-mail: christophe.outteryck@a-sis.com

## 1 Introduction

Logistic warehouses are more and more complex installations: the flows regularly increase, the automation of systems become widespread, logistic processes are globalizing. To be competitive, warehouse management softwares (Warehouse Management Systems (WMS) and Warehouse Control Systems (WCS)) have to become a real help to make the best operational decisions.

Since the introduction of automation systems in production and distribution environments, many research studies on operational decision systems can be found in the literature. This subject has led to a large amount of research papers (see for example [1–4] for surveys on this area). In this paper, we study a sub-problem which appears at the end of a logistic stream in a warehouse and concerns the timetabling of sorting slots. To our knowledge, this problem has not been studied in the literature.

We consider a warehouse composed of several order preparation shops and a set of orders to be prepared on a given time horizon. Each order is composed of several order lines. Each order line corresponds to an amount of work to process on one of the preparation shops. Each order is associated with a transportation truck which is in charge to transport it to its final customer. It forces the subset of all orders associated with a given truck to be totally processed before the known departure of the truck.

Once an order has been prepared, it is dispatched to a sorting slot which is an accumulation area where the order waits to be loaded on the truck. The difficulty of the problem comes from a special resource constraint: to a given truck, a known number of sorting slots is needed from the time the first order associated with the truck begins to be prepared, and this, until the known departure of the truck. Since several orders associated with different trucks are prepared simultaneously and since the number of sorting slots is limited, the timetabling of the sorting slots becomes necessary. In other words, we have to establish the timetable of sorting slots, in aim to ensure that all orders can be processed over the considered time horizon without violating the resource constraint on the number of available sorting slots.

## 2 Problem definition

The problem in hand can be formalized in the following way. Let  $n$  be the number of trucks and let  $m$  be the number of preparation shops in the warehouse. A preparation shop is assimilated as a machine. To each truck  $i$  is associated a task  $J_i$  composed of  $m$  operations  $\{o_{i1}, o_{i2}, \dots, o_{im}\}$ . The processing time  $p_{ij}$  of operation  $o_{ij}$  corresponds to the duration of the whole set of order lines which have to be processed by machine (preparation shop)  $j$  associated with truck  $i$ , *i.e.* to the total amount of work which has to be done by shop  $j$  for truck  $i$ . Each of the  $m$  machines has then to process  $n$  operations (one per task). Let  $d_i$  be the known date of the departure of truck  $i$ . In a feasible solution of the problem, each task  $J_i$  is considered being finished by time  $d_i$ , while

the start-time  $s_i$  of  $J_i$  is to be determined such that all operations of  $J_i$  can be performed. The set of sorting slots is assimilated as a cumulative resource of capacity  $E$ ,  $E$  being the number of sorting slots available in the warehouse. Thus, task  $J_i$  needs a given number  $n_i$  of sorting slots among  $E$  from  $s_i$  until  $d_i$ . Note that the period on which the resource constraint applies depends only on the start-time of the job, contrary to classical scheduling problems in which it depends on periods on which the job is really processed. The problem is to determine start times  $\{s_1, \dots, s_n\}$  in such a way that no more than  $E$  sorting slots can be used at a time, while the scheduling of operations is feasible.

Let  $s_{ij}$  and  $c_{ij}$  be respectively the variables representing the start time and the completion time of operation  $o_{ij}$  in a solution. In a feasible solution, each operation  $o_{ij}$  can start after the start of task  $J_i$  and has to be completed before the departure time of the associated truck:  $s_{ij} \geq s_i$  and  $c_{ij} \leq d_i$ . Each machine is disjunctive and can process only one operation at a time. Note that, contrary to a scheduling shop problem (see *e.g.* [5]), the operations of a same task can be processed simultaneously (and, in fact, will be very often).

Two cases are considered:

- The non-preemptive case in which, once an operation  $o_{ij}$  begins to be processed, it has to be continued until its completion ( $s_{ij} + p_{ij} = c_{ij}$ ). This implies that all order lines associated with a truck will be dealt sequentially.
- The preemptive case, in which interruptions in the processing of tasks are possible ( $s_{ij} + p_{ij} \leq c_{ij}$ ). Such interruptions are useful to execute a part of another operation more urgent associated with another truck.

If variables  $\{s_1, s_2, \dots, s_n\}$  can be fixed in such a way that all the previous constraints hold, it follows that  $n_i$  sorting slots among  $E$  will be used from  $s_i$  to  $d_i$  for truck  $i$ .

In the remainder, we use also the following notation. Let  $H = \max_{i \in \{1, \dots, n\}} d_i$  be the time horizon of an instance of the problem. Let also  $T = \{d_i/i \in N\} + \{0\} = \{t_0 = 0, \dots, t_w = H\}$  be the set of departure dates, values  $t_i$  being indexed in non-decreasing order, *i.e.*  $t_0 = 0 < t_1 < \dots < t_w = H$ .

### 3 Complexity study of the problem

The special preemptive case in which we have  $\sum_{i=1}^n n_i \leq E$  is polynomially solvable. Indeed, the resource constraint according to the number of sorting slots disappears in this case. Therefore, it leads to the resolution of  $m$  single-machine scheduling problems with deadlines ( $1|\bar{d}_i|-$ ) which are polynomially solvable [6].

The other special case in which any  $p_{ij} = 1$  is also polynomially solvable. Indeed, the operations can be scheduled in the same way on each machine. Thus the problem is reduced to another one with  $m = 1$ . It then remains to know if we can schedule operations without violating the resource constraint on sorting slots. For that, we iteratively schedule operations from  $H$  to 0 in decreasing order of departure times. When several operations have the same



departure time, they are scheduled in decreasing order of the number of sorting slots. The process stops when all operations are scheduled or when the resource constraint on sorting slots does not hold (the instance is not feasible).

We state that the 3-PARTITION problem [6] can be reduced to the general preemptive case with  $m = 1$ . It follows that both the general preemptive and non-preemptive problems are strongly NP-complete.

#### 4 Some methods to solve the problem

We propose optimization methods to solve both the preemptive and the non-preemptive versions of the problem. The proposed methods relies on the following proposition:

**Proposition 1** *Let  $S$  be a feasible solution of an instance of the problem in which a job  $i$  is executed from  $s_i$  to  $d_i$ . Then  $S$  can be transformed in a feasible solution  $S'$  in which job  $i$  uses  $n_i$  sorting slots from  $s'_i = \max\{t_k, t_k \in R \wedge t_k \leq s_i\}$  and where the scheduling of operations is not modified.*

Thus, we can deduce the following dominance rule.

**Theorem 1** *There exists at least one solution in which the start time of use of sorting slots take values in set  $T$ .*

This last proposition can be used in several ways. From a practical point of view, it offers a method to simplify the timetabling of the use of the sorting slots since they can be established by periods determined by the departure times of the different trucks. Of course, the additional use (a priori useless) of sorting slots by a job can be seen as additional flexibility in the preparation process. From a resolution point of view, this dominance rule is used to establish integer linear programming formulations and constraint based scheduling methods [7] to solve in practice the problems. Thus, we propose an ILP model and a constraint programming model to solve the non-preemptive case. Then, we propose a flow-based ILP model and a multiperiod ILP model to solve the preemptive case. These methods are experimentally compared from a computational point of view.

#### References

1. G. Cormier, E. Gunn, *European Journal of Operational Research* **58**, 3 (1992)
2. J. Van den Berg, *IIE Transactions* **31**(751762) (1999)
3. B. Rouwenhorst, V. Reuter, V. Stockrahm, G. Van Houtum, R. Mantel, W. Zijm, *European Journal of Operational Research* **122**, 515533 (2000)
4. R. De Koster, T. Le-Duc, K. Roodbergen, *European Journal of Operational Research* **182**, 481501 (2007)
5. P. Brucker, *Scheduling Algorithms* (Springer Lehrbuch, 1995)
6. M. Garey, D. Johnson, *Computers and intractability: a guide to the theory of NP-completeness* (W.H. Freeman, 1979)
7. P. Baptiste, C. Le Pape, W. Nuijten, *Constraint-based scheduling, applying constraint programming to scheduling problems, International Series in Operations Research and Management Science*, vol. 39 (Kluwer, 2001)

---

## Days off scheduling

### A 2-phase approach to personnel rostering

Sophie van Veldhoven · Gerhard Post · Egbert van der Veen

Received: date / Accepted: date

#### 1 Introduction

The personnel rostering problem (or nurse rostering problem) is a well-known combinatorial optimization problem, with a rich literature, see [Ernst et al(2004)] and [Burke et al(2004)]. In many practical situations the large number of working time regulations and preferences make it difficult and time consuming to construct a good schedule.

One way to reduce the complexity of the personnel rostering problem is to decompose the problem into subproblems that are easier to solve. Though the subproblems can possibly be solved to optimality in a reasonable time, the combination of the subproblems does not necessarily lead to an optimal solution to the original problem. In this work we present a 2-phase decomposition model. In the first phase we construct a *days off schedule*, that is a schedule that specifies for each employee the working days and the days off. We also consider a refinement in which we consider day off, day shift or night shift. In the second phase we specify which shifts are actually assigned to the employees on their working days, which means that we solve a *shift rostering problem* respecting the days off schedule found in the first phase.

In practice, the construction of a schedule with working days and days off can be a separate step in the process of assigning staff. For individual employees, it may be pleasant to know the working days a long time ahead, so that they can plan their free

---

Sophie van Veldhoven  
Aviv, Langestraat 11, 7511 HA Enschede, The Netherlands,  
E-mail: sophievveldhoven@gmail.com

Gerhard Post (corresponding author)  
Department of Applied Mathematics, University of Twente, P.O. Box 217, Enschede, The Netherlands,  
and ORTEC, Groningenweg 6k, 2803 PV Gouda, The Netherlands,  
E-mail: g.f.post@utwente.nl

Egbert van der Veen  
Department of Applied Mathematics, University of Twente, P.O. Box 217, Enschede, The Netherlands,  
and ORTEC, Groningenweg 6k, 2803 PV Gouda, The Netherlands,  
E-mail: Egbert.vanderVeen@ortec.com

time. The exact hours they have to work, are not essential to know in the long term. In addition, requests for vacation can be taken into account long before the actual planning, which can alert the planner for possible capacity problems. The assignment of the various shifts to the employees can be done in a later stage, for example, a month before the start of the new planning period.

The aim of this paper is two-fold: investigate the computation times for a class of personnel rostering problems with and without decomposition, and, secondly, investigate the quality of the schedules in the 2-phase approach compared to solutions obtained in one run. We will use integer linear programming (ILP) in our research. Hence we will present a mathematical program for the construction of days off schedules, based on the constraints of the original problem.

## 2 Approach

As data sets we use the Employee Scheduling Benchmark Data Sets [Curtois(2007)]. We formulate an ILP for these data sets, which are solved using CPLEX 12.2, with the time limit set to 1 hour. These results serve as the benchmark for our tests. Next we formulate the days off scheduling problems, which we derive from the instances. We consider two variants:

**(On, Off):** Distinguish working days and days off.

**(Day, Night, Off):** Distinguish day shifts, night shifts, and days off.

We include the refinement (Day, Night, Off) in our study, because in many cases the requirements before and after night shifts are very different from (all) other shifts. In fact we reduce the original shift scheduling problem to a shift scheduling problem with 2 shift types (on, off), respectively 3 shift types (day, night, off). Hence in the first and second phase we can use similar ILP models, with the addition that in the second phase the assignment of shifts to employees should obey the decisions of the first phase. Though the ILP models are similar, the reduction in the first phase is not straightforward, and the next subsections describe how this is done. For this, we consider the different types of constraints present in the benchmark, and explain shortly how we handle them.

### Requests and fixed assignments

Employees can have shifts preassigned, can have requests for shifts on or can have requests for a day off. Those can be transferred to the days off scheduling problem directly: work requests or shift on requests result in days on. Shift *off* requests are ignored, since the employee might work on the same day in another shift.

### Cover requirements

Per day the instance describes a minimum, maximum, or preferred cover per shift type or per time period. These have to be aggregated to be useful in the days off schedule. In case the cover requirements are in (overlapping) shift types we formulate an ILP model to determine the minimum number of employees needed, and use this amount for the days off schedule. In case that time periods are used, we proceed in the same way.

### Pattern requirements

The pattern requirements can express a wide variety of constraints for individual schedules. It can contain constraints like the maximum number of shifts per planning interval or week, shift sequences, shifts in weekends, etc. If a pattern concerns all shift types, we can use it in the days off schedule. In the case of (day, night, off), we can incorporate more pattern requirements, potentially leading to better results.

### Workload requirements

Workload requirements describe the number of hours an employee should work in the planning period. Clearly, these are difficult to use in the days off schedule if different shift types have different lengths. Since we use only necessary conditions, the best we can do is to calculate upper and lower bounds on the working days, and add those conditions as constraints to the ILP. In the shift scheduling phase we have full information, so that we can use the correct workload requirements.

## 3 Results

We tested the 2-phase decomposition on 16 instances present at [Curtois(2007)] with 3 to 12 shift types. For almost all of these instances optimal solutions are known. Running CPLEX 12.2 for 1 hour on a standard dual core PC yields an optimal solution in 9 cases. For these cases we find that the (on, off) decomposition gives only good results for 4 of these instances, where using the (day, night, off) decomposition leads to good results for 6 instances. For the other 7 cases we compare the decomposition results with the results without decomposition. Then we find good results in 3 out of 7 cases if we use the (on, off) decomposition, and in 5 cases if we use the (day, night, off) decomposition. In most cases the calculation times reduce by more than 98%. Here ‘good results’ is meant in the sense that the cost is of the same order of magnitude; sometimes the results are better, but usually slightly worse.

Several improvements can be made to the model and the solution method. For the model we can improve the way that the pattern constraints are dealt with; now in several instances we use only a minor portion of these in the days off schedule, which in some instances leads to high costs in phase 2. For the solution part, we can apply the decomposition several times, say 10 times, with a constraint added that makes sure we do not generate the same solution. On some instances, this approach significantly improves the results.

### References

- [Burke et al(2004)] Burke EK, De Causmaecker P, Vanden Berghe G, Van Landeghem H (2004) The state of the art of nurse rostering. *Journal of Scheduling* 7(6): pp. 441–499.
- [Curtois(2007)] Curtois T (2007) <http://www.cs.nott.ac.uk/~tec/NRP>, Employee Scheduling Benchmark Data Sets.
- [Ernst et al(2004)] Ernst AT, Jiang H, Krishnamoorthy M, Owens B, Sier D (2004) An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research* 127: pp. 21–144.

---

## Self-Rostering applied to case studies

### An ILP method to construct a feasible schedule

Suzanne Uijland · Egbert van der Veen ·  
Johann Hurink · Marco Schutten ·

Received: date / Accepted: date

**Abstract** We discuss the self-rostering problem, a concept receiving more and more attention from both theory and practice. We outline our methodology and discuss its application to a number of practical case studies.

**Keywords** Personnel Rostering · Shift Rostering · Self Rostering · Linear Programming

## 1 Introduction

In service industries, like healthcare and security services, people work around the clock. Considering the many employee preferences and labor legislation that are implied on rosters, it is, both in theory, [L. De Grano et al (2009); Rnnberg and Larsson (2010)] and practice, often hard to come up with good or fair shift rosters. A way to better cope with employee preferences, and to

---

Suzanne Uijland  
Department of Management and Governance, University of Twente, P.O. Box 217, Enschede,  
The Netherlands, ORTEC, Groningenweg 6k, 2803 PV, Gouda, The Netherlands

Egbert van der Veen  
Presenting author  
ORTEC, Groningenweg 6k, 2803 PV, Gouda, The Netherlands  
Tel.: +31182540500  
Fax.: +31182540540  
Center for Healthcare Operations, Improvement, and Research (CHOIR), University of  
Twente, PO Box 217, 7500 AE Enschede, The Netherlands  
E-mail: Egbert.vanderVeen@ortec.com

Johann L. Hurink  
Department of Applied Mathematics, University of Twente, P.O. Box 217, Enschede, The  
Netherlands,

Marco Schutten  
Department of Management and Governance, University of Twente, P.O. Box 217, Enschede,  
The Netherlands,

increase job satisfaction, is self-rostering. The main idea in self-rostering is that employees can propose their own shift rosters, and if they do this in a ‘good’ way, they get to work most of their shifts as in their preferred schedule.

Bailyn et al (2007) argue that self-rostering increases job satisfaction by an improved work-life balance, increased predictability and flexibility in schedules, and enhancing the communication and interaction to stimulate cooperative community building. Especially the latter is typical for self-rostering, since self-rostering stimulates employees to propose schedules that are ‘good’ for the organization, see Section 3.

However, the shift rosters employees propose in general do not match up with a shift staffing demand, and, therefore, staffing *shortages* for shifts for specific days occur. The goal of this paper is to design a method that unassigns some of the proposed shifts in order to solve the shortages.

In the next sections, we will subsequently describe the self-rostering process, our solution technique, and discuss some results.

## 2 Self-rostering process

For a given rostering period (of e.g., a month) the self-rostering problem can be decomposed in 5 phases:

1. The organization defines the *staffing demand*, i.e., specifies, per day, the number of employees that need to perform a certain shift.
2. The employees propose their preferred schedule. These schedule have to obey labor legislation, and other scheduling constraints specified by the organization, like forward rotations or planning homogeneous work blocks.
3. The employees’ preferred schedules are combined, after which the staffing demand is subtracted from these, indicating surpluses and shortages on each shift.
4. The information of Phase 3 is returned to the employees, offering them the opportunity to adjust their schedules.
5. The planner solves the remaining shortages after Step 4.

In this paper we focus on the fifth phase and propose a method to solve the remaining shortages. Note that the presented method is independent of the self-rostering process itself and the way employees create and adjust their proposed schedules. In fact our method can be applied to any given schedule where for each employee a preferred schedule is given and where a mismatch between the shift staffing and shift demand is given.

## 3 Method

The objective of our method is to solve as many shortages as possible, whereby making sure that a specified fraction of an employee’s proposed schedule remains.

To resolve the remaining shortages, we use an iterative method. In every iteration an integer linear program (ILP) is solved, that mainly considers two things: the *score* of every employee's schedule, and *swaps*.

The *score* of an employee's schedule is calculated as follows. First, we assign 'points' to shifts using a specified point system. For example, understaffed shifts are assigned 3 points, overstaffed shifts get 1 point, and matching shifts (shifts where the staffing demand is exactly matched) get 2 points. Second, we calculate the score of every employee, by calculating his total points, possibly multiplied by some factor to, e.g., take part time percentages into account. As a consequence, employees with a large score work many relatively unpopular shifts, whereas for employees with a small score the opposite holds.

In a *swap* we unassign an employee from one of his overstaffed shifts, and assign this employee to a shift that is currently understaffed. A swap can be performed on two shifts on the same day, but also on two shifts on different days. A swap may only be performed if it obeys labor legislation and other practical scheduling constraints specified by the organization. This can be precalculated before running the ILP.

Given the *scores* and *swaps* in every iteration, the ILP is used to select a subset of swaps is selected, with at most one swap per employee. We select at most one swap per employee to make sure we do not violate labor legislation or other constraints the organization implies on the schedules. The objective makes a trade-off between two factors. First, we want to maximize the number of swaps selected, in order to maximize the number of shortages solved in an iteration. By maximizing the number of shortages solved in an iteration we expect to solve more shortages in total. Secondly, we preferably perform swaps at employees that have a low score, in order to reward employees that choose relatively many unpopular shifts.

## 4 Results

We applied our method to 168 schedules based on data from real life. The range of the number of employees is 15-80, and the range of the number shortages is 16-212. Labor legislation is dealt with by including only swaps in the model that do not violate labor legislation.

From the experimental results we observe that the mean computation time of our method is 2.8 seconds, the mean number of shortages left is 0.90, and on average 92.2% of the schedules is retained. The number of remaining shortages is calculated as the positive difference between the number of remaining shortages and the number of remaining excesses. Depending on, among others, the point system used, parameter values of the ILP, and swaps, there is a trade-off in the model between either the number of shortages solved and fraction of the proposed schedules that is preserved.

## References

- Bailyn L, Collins R, Song Y (2007) Self-scheduling for hospital nurses: an attempt and its difficulties. *Journal of Nursing Management* 15(1):72–77, DOI 10.1111/j.1365-2934.2006.00633.x, URL <http://dx.doi.org/10.1111/j.1365-2934.2006.00633.x>
- L De Grano M, Medeiros D, Eitel D (2009) Accommodating individual preferences in nurse scheduling via auctions and optimization. *Health Care Management Science* 12:228–242, URL <http://dx.doi.org/10.1007/s10729-008-9087-2>, 10.1007/s10729-008-9087-2
- Rnnberg E, Larsson T (2010) Automating the self-scheduling process of nurses in swedish healthcare: a pilot study. *Health Care Management Science* 13:35–53, URL <http://dx.doi.org/10.1007/s10729-009-9107-x>, 10.1007/s10729-009-9107-x



---

## Towards Fair and Efficient Assignments of Students to Projects

Marco Chiarandini · Rolf Fagerberg ·  
Stefano Gualandi

We consider the following problem in project assignment. We are given a set  $P$  of *project topics* and a set  $S$  of *students*. For each topic  $i \in P$  a limited number  $t_i$  of teams can be created. If a team for topic  $i$  is created, the number of students assigned to it must be between a minimum and maximum bound,  $l_i$  and  $u_i$ , respectively. Students express preferences for the topics by ranking a subset of project topics. For each student  $s \in S$  this ranking is given by an ordered set  $r(s) = (p^{(1)}, \dots, p^{(q_s)})$ ,  $p^{(i)} \in P$ ,  $1 \leq i \leq q_s \leq n$ . Moreover students can register in *groups* of at most  $\ell$  persons, if they want to be assigned to the same team. That is, students are partitioned in groups,  $G = \{g_1, \dots, g_{m'}\}$ ,  $\bigcup_i g_i = S$ ,  $g_i \cap g_j = \emptyset, \forall i \neq j$  and  $|g_i| \in \{1, 2, \dots, \ell\}$ . Students in the same group  $g_i$  have the same preference set, that is,  $r(s_1) = r(s_2)$  for all  $s_1, s_2 \in g_i$ .

We wish to find an assignment of students to project teams,  $\sigma : S \rightarrow \{1, 2, \dots, \sum_{i \in P} t_i\}$  such that students are assigned to exactly one project from their preference set and team bounds and group requirements are satisfied. If no assignment satisfying these constraints exists, we will modify the input data and iterate. If more than one assignment can be found, we wish to choose one among them according to the criteria of fairness and collective welfare.

Preference sets can be transformed into a *score matrix*  $V \in \mathbb{N}_0^{|S| \times |P|}$ , where each element  $v_{si}$  represents how student  $s$  ranks project  $i$ . The score 1 is set for the topics that are ranked first and  $q_s$  for the topics that are ranked in the  $q_s$ -th position. If a project topic  $i$  is not in the preference list of the student  $s$ , the corresponding value  $v_{si}$  will never be used and it can be set to any number, for example, zero.

---

M. Chiarandini · Rolf Fagerberg  
Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark, E-mail: {marco|rolf}@imada.sdu.dk

S. Gualandi  
Department of Mathematics, Università di Pavia, via Ferrata, 1 I-27100, Pavia, Italy, E-mail: stefano.gualandi@unipv.it

The quality of an assignment  $\sigma$  that satisfies all constraints is determined by a vector  $\mathbf{v} = (v_{1,\sigma(1)}, \dots, v_{m,\sigma(m)})$ ,  $v_{s,\sigma(s)} > 0, \forall s \in S$ , and by the distribution of students over ranks  $\boldsymbol{\delta} = (\delta_1, \dots, \delta_\Delta)$ , where  $\Delta = \max\{q_s \mid s \in S\}$  and  $\delta_{i=1.. \Delta} = |\{s \in S \mid v_{s,\sigma(s)} = i\}|$ . Students will prefer assignments over others on the basis of their *individual utility*, that is, their score in the vector  $\mathbf{v}$ . In the decision-making process of an ad hoc committee that has to solve the allocation problem under limited resources, the focus will be on the *collective welfare*. From this viewpoint, the interest is on assignments that are Pareto optimal or *efficient* with respect to their profile vectors  $\mathbf{v}$ . The two most prominent ways to aggregate a profile of preference relations into a collective preference relation are the *classical utilitarian ordering* and the *egalitarian ordering* [4]. For two feasible assignments  $\sigma^1$  and  $\sigma^2$ , the former assigns a weight to each value,  $w : \{1, 2, \dots, \Delta\} \rightarrow \mathbb{Z}^+$  and compares  $\sum_s w(v_{s,\sigma^1(s)})$  with  $\sum_s w(v_{s,\sigma^2(s)})$ , and the latter uses the *leximin order*, which consists in reordering the two vectors  $\mathbf{v}^1$  and  $\mathbf{v}^2$  by increasing coordinates and comparing them lexicographically. Both relations define a strict weak order.

Taking only efficiency into account it is possible to create examples where the overall satisfaction is high to the disadvantage of a few students. The individual welfare or *fairness* criterion aims at ensuring that no student is disadvantaged to the benefit of others. A way to achieve this is by searching for the assignment that minimizes the worst rank, that is,  $\min \max\{v_{s,\sigma(s)} \mid s \in S\}$ . This is also known as the *minimax* criterion [5].

On the other hand, the minimax criterion alone makes no use of additional information to decide among assignments with the same guarantees on the maximum scores in the vector  $\mathbf{v}$ . For example, the two vectors  $(1, 3, 3, 3)$  and  $(1, 2, 2, 3)$  are not distinguishable. An approach that takes both fairness and collective welfare into account consists in first optimizing according to the minimax criterion and then, restricted to only minimax optimal solutions, optimizing collective welfare using the weighted value order. An alternative approach that overcomes the drawback of the minimax criterion and conciliates egalitarianism and Pareto-efficiency is the *leximin* order, which subsumes the minimax criterion.

Without common registrations and no minimum number of students per project team, the problem under a classical utilitarian approach can be formulated as a particular case of minimum cost flow. With all constraints the problem is instead strongly NP-hard as it is a special case of the multiple knapsack problem or the generalized assignment problem that are optimization versions of the strongly NP-complete 3-partition problem [1].

In [3] the authors used a genetic algorithm approach to solve a similar problem but without project topics, lower bounds on team sizes, and group registrations. The issue of fairness is addressed by means of the multiple solutions returned by the genetic algorithm when it solves a formulation with weighted sum. Our work is an adaptation of the study by Garg et al. [2] in the context of conference management for assigning papers to referees. The problem there treated is similar to ours, but with the further issue that referees,

contrary to students, can receive more than one paper and a load balancing criterion has to be included. Garg et al. show that the leximin approach under lexicographic and weighted score orders are both NP-hard for  $\Delta \geq 3$ . Then they give an approximation algorithm for the general case.

We have designed a randomized greedy algorithm that implements a lottery approach. The procedure is appealing from the point of view of fairness. Then we studied different formulations of the problem in integer linear programming (ILP) and constraint programming (CP). We compared these methods on real life instances with up to 300 students and 102 project teams in 80 different topics. Results showed that ILP models based on a *distribution approach* to handle the lexicographic order solve the problem in a matter of seconds and the assignments found outperform those of the lottery approach in terms of both feasibility and quality. The CP models studied so far are instead not yet competitive.

We use our solutions in practice at the Faculty of Science of the University of Southern Denmark, where in the first year of their education students must undertake a group project of the duration of one quarter. Students are left free to rank project topics independently from the specific branch of science in which they will later specialize. In the past we used the lottery algorithm and had to interact with the students because initially no feasible solution was found. Since 2011, we use the ILP model based on lexicographic optimization and assignments that satisfy students and the administrating committee are found immediately even when the lottery algorithm would not find any.

## References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of  $\mathcal{NP}$ -Completeness. Freeman, San Francisco, CA, USA (1979)
2. Garg, N., Kavitha, T., Kumar, A., Mehlhorn, K., Mestre, J.: Assigning papers to referees. *Algorithmica* **58**(1), 119–136 (2010). DOI 10.1007/s00453-009-9386-0
3. Harper, P.R., de Senna, V., Vieira, I.T., Shahani, A.K.: A genetic algorithm for the project assignment problem. *Computers & Operations Research* **32**(5), 1255 – 1265 (2005). DOI 10.1016/j.cor.2003.11.003
4. Moulin, H.: Fair Division and Collective Welfare. The MIT Press (2003)
5. Yager, R.R.: Constrained OWA aggregation. *Fuzzy Sets and Systems* **81**(1), 89 – 101 (1996). DOI 10.1016/0165-0114(95)00242-1

---

# Regulation-based University Course Timetabling

Michael Zeising, Stefan Jablonski

*Chair for Applied Computer Science IV*

*University of Bayreuth*

*Bayreuth, Germany*

{michael.zeising, stefan.jablonski}@uni-bayreuth.de

## 1 Introduction

Conventional models of course timetabling either rely on enrolment or on fixed curricula [1]. At the University of Bayreuth, enrolment is not desired for political reasons. The university's management insists on a maximum of freedom for students so the post-enrolment timetabling techniques cannot be applied here. However, fixed curricula in the sense of "a group of courses such that any pair of courses in the group have students in common" [2] are not present either. The only systems of rules on which the students' course selection is based on are the examination regulations of their program of study. These regulations are legally necessary information in Germany (§ 16 Abs. 1 HRG) and are already represented in most university management solutions for various purposes. Not even the specialised variations of the curriculum model with support for optional courses [3] suffice for timetabling on the basis of regulations. The objective is to implement conventional timetabling approaches in an environment where neither enrolment information nor fixed curricula are present.

In this contribution, we show how to derive timetabling conflicts from examination regulations as they would actually arise from curricula. Moreover, these conflicts are weighted according to the number of students that are affected by the conflict. With the help of these derivations, the information can then be converted to the standard model and utilised by standard timetabling techniques.

## 2 Domain Model

The main principle does not make high demands on the domain model. A regulation is basically a tree whose leaves are courses while each course consists

of one or more sessions which are the planned entities in the end. The inner nodes of the tree can be modules in the sense of the European Credit Transfer System (ECTS) [4] or any other type of unit used for structuring a program of study like e.g. phases, areas, sections and so on. Every inner node can be equipped with two types of constraints concerning its children: *n-out-of* and *prerequisites*. For a node constraint by *n-out-of* only  $n$  out of the child nodes have to be selected by the student. The targets of *prerequisites* are the units that have to be attended in a semester before the source unit.

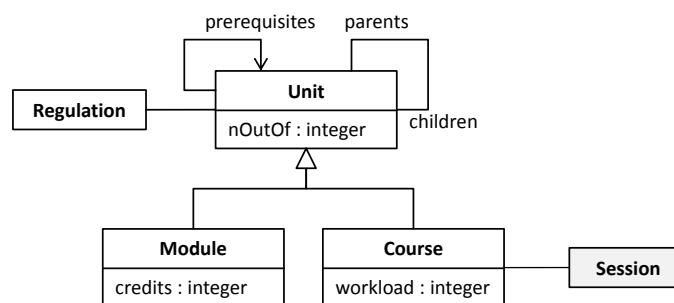


Figure 1. Domain model of examination regulations and sessions

The above domain model for representing examination regulations is compatible with major commercial university management solutions like *CAMPUSonline* [5] and *CampusNet* [6] which eases an integration with these systems.

### 3 From Regulations to Weighted Conflicts

Initially, no knowledge is applied which means that all the sessions of a regulation are considered conflicting. The idea is now to remove as much conflicts as possible and to weight the remaining ones. This will be demonstrated by examples in the following.

The most frequent pattern in regulations is a specialisation choice which means that there is a node constraint by *1-out-of* and several child nodes below. This means that a student has to select either one or the other direction (sub-tree) and no student is expected to attend both sets of sessions. As a result, we may safely remove the conflicts between the two branches.

Furthermore, course  $c_2$  references  $c_1$  as a prerequisite which means that  $c_2$  is based on knowledge acquired in  $c_1$ . Assuming that sessions last for a full semester, we may infer that no student will attend both,  $c_1$  and  $c_2$ , in the same semester. As a result, the conflicts between their sessions can be removed. The remaining

conflict affects half the students in the program so that we weight it with a value of 0.5. The interpretation lead from six conflicts to a single weighted one.

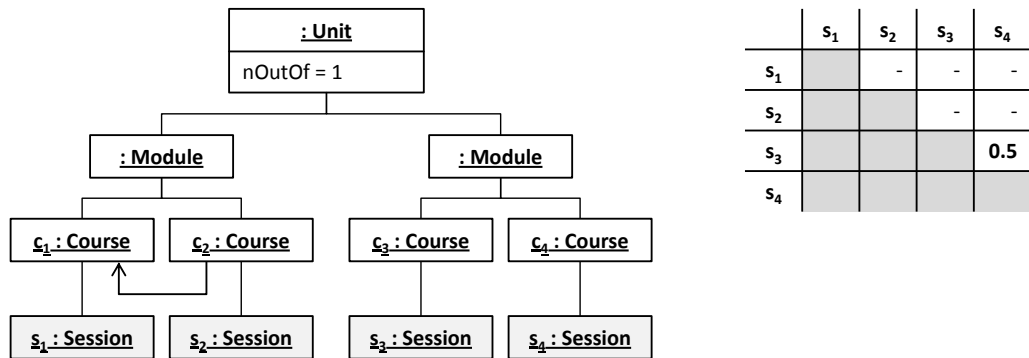


Figure 2. Example model for the 1-out-of pattern with resulting conflict matrix

A more complex example is a 2-out-of choice. In this case, two out of several “packages” have to be selected by the student. As this choice is not of an exclusive kind, we must not remove any conflicts here.

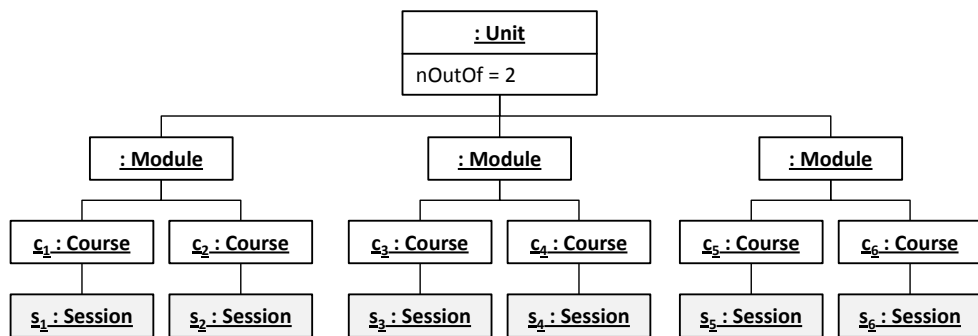


Figure 3. Example model for the 2-out-of pattern

Nevertheless, we gain knowledge by weighting. The conflicts within the three pairs affect two thirds of the students in the program while the others affect only one third of them.

	s <sub>1</sub>	s <sub>2</sub>	s <sub>3</sub>	s <sub>4</sub>	s <sub>5</sub>	s <sub>6</sub>
s <sub>1</sub>		<b>0.66</b>	0.33	0.33	0.33	0.33
s <sub>2</sub>			0.33	0.33	0.33	0.33
s <sub>3</sub>				<b>0.66</b>	0.33	0.33
s <sub>4</sub>					0.33	0.33
s <sub>5</sub>						<b>0.66</b>
s <sub>6</sub>						

Figure 4. Conflict matrix resulting from the 2-out-of example

This information can be used for checking soft constraints in subsequent optimisation approaches.

## 4 Future Work

This contribution only sketches the basic idea of our approach. There are a lot more patterns in regulations that can be interpreted like, e.g., a minimum sum of credit points to be achieved in certain branches of the program of study. These patterns are to be formalised and efficient implementations for their discovery are to be developed.

Another future task is to investigate the relationship between the number of available regulations and the density of the resulting conflict graph of sessions. On the one hand, this will provide a basis for assessing the benefit and usefulness of our approach. On the other hand, many real-world regulations contain such a high degree of freedom so that huge amounts of conflicts result that cannot be accounted for with regard to limited time and space. The process of creating a regulation could therefore be supported by estimating its effect in advance.

## 5 Acknowledgements

This work is kindly funded by the Oberfrankenstiftung grant for project number 02803.

## References

- [1] B. McCollum, "A perspective on bridging the gap between theory and practice in university timetabling," in 6th International Conference on Practice and Theory of Automated Timetabling (PATAT '06), 2006, pp. 3-24.
- [2] F. De Cesco, et al., "Benchmarking Curriculum-Based Course Timetabling: Formulations, Data Formats, Instances, Validation, and Results," in 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), 2008.
- [3] H. Schmitz and C. Heimfarth, "Cross-Curriculum Scheduling with THEMIS - A Course-Timetabling System for Lectures and Sub-Events," in 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010), Belfast, UK, 2010.
- [4] E. Commission, "ECTS Users' Guide," ed: Luxembourg: Office for Official Publications of the European Communities, 2009.
- [5] T. U. Graz. (2012, 27.02.2012). CAMPUSonline. Available: <http://campusonline.tugraz.at>
- [6] D. I. GmbH. (2012, 27.02.2012). CampusNet. Available: <http://www.datenlotsen.de/de/produkte/campusnet/Seiten/CampusNet.aspx>

---

## Co-evolving add and delete heuristics

Jerry Swan · Ender Özcan · Graham Kendall

Received: date / Accepted: date

**Abstract** Hyper-heuristics are (meta-)heuristics that operate at a high level to choose or generate a set of low-level (meta-)heuristics to solve difficult search and optimisation problems. Evolutionary algorithms are well-known nature-inspired meta-heuristics that simulate Darwinian evolution. In this article, we introduce an evolutionary-based hyper-heuristic in which a set of low-level heuristics compete to solve timetabling problems.

**Keywords** Hyper-heuristics · Coevolution · Ruin-and-recreate

### 1 Introduction

Hyper-heuristics are (meta-)heuristics that operate at a high level to choose or generate a set of low level (meta-)heuristics to solve difficult search and optimisation problems [1],[6]. Heuristics can be used to search the solution space directly or construct a solution based on a sequence of moves. In most of the previous studies, the type of the low-level heuristics used is uniform, i.e. they are either constructive or perturbative (improvement) heuristics. Hyper-heuristics aim to replace bespoke approaches by general methodologies for solving different problems. They provide a “*good enough - soon enough - cheap enough*” framework for problem solving.

Timetabling problems are NP hard [2], real-world constraint optimisation problems. A timetabling problem requires scheduling of given events using limited resources, subject to a set of constraints. Evolutionary algorithms are well-known nature-inspired meta-heuristics that simulate Darwinian evolution. In this paper, we introduce an evolutionary based hyper-heuristic in which a set of low level heuristics compete to solve timetabling problems.

---

Automated Scheduling, Optimisation and Planning (ASAP) Research Group,  
School of Computer Science, University of Nottingham,  
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK.  
{jps,exo,gxk}@cs.nott.ac.uk



```

(* Generate feasible binary strings. '+' denotes concatenation.
   Initialize with: generateAddDeleteLists( n, "01" ); *)
function generateAddDeleteLists( var n : Int, var bits :
  BinaryString ) : ListOfBinaryString
begin
  var result : ListOfBinaryString;
  if( n = 1 )
  begin
    result := result + bits;
  end;
  else
  begin
    result := result + generateAddDeleteLists(n-1, bits+ "01");
    result := result + generateAddDeleteLists(n-1, "0"+bits+"1");
    if( bits doesn't have prefix "01" )
    begin
      result := result + generateAddDeleteLists( n-1, "01"+bits);
    end;
  end;
  return result;
end.

```

**Listing 1** Generate feasible binary strings

### 1.1 Low-level heuristics

A solution to a timetabling problem can be reconstructed from a previous solution by successively deleting and adding (re-scheduling) events. In our hyper-heuristic framework, we propose a sequence of delete-add (0-1) operations with a fixed length. Add and delete operations can be handled in many different ways. The simplest approach for the delete operation is choosing an event randomly and putting it into an unscheduled events list. On the other hand, the add operation requires two consecutive actions to be taken. Firstly, an event should be selected from the list of unscheduled events and then a suitable period should be selected for scheduling.

A fixed-length binary string can be used to represent a series of add (1) and delete (0) operations for reconstructing a new solution from an existing one. A feasible string of length  $2n$  can be formed by respecting the following rules:

1. The number of ones must be equal to the number of zeros, that is  $n$ . Hence, the string length is an even number.
2. The first entry in the string is always the delete operation, since there is no unscheduled event at the start.
3. For any prefix of the string, the number of ones must be less than or equal to the number of zeroes.

For example, given a solution  $S$  and a feasible string "0011", two randomly selected events are deleted from  $S$ , and  $S'$  is formed by rescheduling them, successively. On the other hand, "0110" is not a feasible string, since after

```

(* Co-evolve binary strings to be used for reconstructing
   a new solution to a timetabling problem from an old one. *)
procedure coevolve( var pop : Population )
begin
  (* let individuals compete for generating a better individual *)
  for( each individual in pop )
  begin
    var previousTimetable = getTimetable( individual );
    var newTimetable := applyADL( individual.getTimetable(),
      individual.getADL() );
    if( newTimeTable is better than previousTimetable )
    begin
      individual.setTimetable( newTimetable );
    end;
  end;
  sortByFitness( pop );
  bestIndividual = pop(0); (* Keep the best individual *)

  (* divide the population into 4 parts *)
  (* copy individuals in 1st quarter into 2nd quarter *)
  copy(pop, 0, popSize/4, popSize/4, 2*popSize/4);
  (* copy individuals in 1st quarter into 3rd quarter *)
  copy(pop, 0, popSize/4, 2*popSize/4, 3*popSize/4);

  (* mutate timetable state and randomize associated add-delete
     list in 1st quarter *)
  mutateTimetableAndRandomizeADL(pop, 2, popSize/4);
  (* mutate timetable state in 2nd quarter *)
  mutateTimetable(popSize/4, 2*popSize/4);
  (* randomize add-delete-list in the 3rd quarter *)
  mutateADL(2*popSize/4, 3*popSize/4, addDeleteLists);
  (* randomize state and add-delete list in 4th quarter *)
  randomizeTimetableAndADL( pop, 3*popSize/4, popSize );
end

```

**Listing 2** Co-evolve timetables and add-delete lists

deleting and rescheduling an event, the unscheduled event list becomes empty and the add operation at the third location is not possible. Listing 1 shows a divide-and-conquer approach that generates such strings for a given  $n$ . For  $n = 1$ , the only such string is “01”, for  $n = 2$ , we have “0101”, “0011” and for  $n = 3$ , “010101”, “001011”, “001101”, “000111”, “010011”. The running time of the algorithm is  $\mathcal{O}(3^n)$ . Hence, we will use practical values for  $n$  and after generating the feasible strings, they compete for survival while constructing timetables within an evolutionary framework, as described in listings 2 and 3.

## 2 Results

The results presented here are for the 8 publicly-available datasets of the exam-timetabling track from the ITC2007 competition [4], with Müller’s celebrated

```

(* Co-evolution framework. *)
procedure framework( var posSize : Int )
begin
  var allAddDeleteLists , coevolvedAddDeleteLists :
    ListOfBinaryString;
  var population : ListOfTimetable;

  (* Generate feasible add-delete strings *)
  allAddDeleteLists := generateString(n, "01");

  population := generateRandomPopulation( popSize ,
    allAddDeleteLists );

  while( termination condition (e.g. num-iterations ) not met )
  begin
    (* co-evolve a population of timetables and add-delete
       operators *)
    coevolve( pop );
  end;
end.

```

**Listing 3** Top-level co-evolutionary framework

hybrid solver [5] ranking first on each dataset<sup>1</sup>. Gogos et al. [3] are second and have rankings [ 3, 4, 3, 2, 3, 3, 2, 3 ] over these 8 datasets<sup>2</sup> with average rank of 2.85. Table 1 gives the scores obtained by our program on a Pentium 4 dual-core 3GHz PC with 2GB of RAM. The maximum runtime of our program was set to 377 seconds (as determined by the competition benchmark program<sup>3</sup>). The respective rankings for our program were [ 2, 3, 2, 4, 5, 2, 3, 2 ], yielding an average rank of 2.875, which would place us in joint second with Gogos over these 8 datasets. Table 2 shows  $(\bar{x}, \sigma)$  of the results of each dataset for Müller, Gogos and our program for the cases where all 10 sample runs are feasible (i.e. are of the form  $0, x$  for some  $x$ ).

### 3 Conclusion

We have presented a co-evolutionary variant on the ‘ruin-and-repair’ strategy that ranks joint second with the pre-existing finalists on the publicly-available datasets of the ITC2007 competition exam-timetabling track.

Future work involves investigating alternative entity relationships between solution and add-delete string (e.g. N:1 and 1:N as opposed to the 1:1 approach adopted here) and associating some probability (possibly adaptively determined) with this relationship.

In addition, it is interesting to note from Table that while our approach generally exhibits a higher standard deviation, it also yields a greater number

<sup>1</sup> <http://www.cs.qub.ac.uk/itc2007/winner/tomasmuller.htm>

<sup>2</sup> <http://www.cs.qub.ac.uk/itc2007/winner/christosgogos.htm>

<sup>3</sup> Available at [http://www.cs.qub.ac.uk/itc2007/index\\_files/benchmarking.htm](http://www.cs.qub.ac.uk/itc2007/index_files/benchmarking.htm)

dataset 1	dataset2	dataset3	dataset4	dataset5	dataset6	dataset7	dataset8
0,6133	0,942	0,13383	0,27333	0,5922	0,28800	0,5960	0,9590
0,6094	0,893	0,13364	0,30322	0,5333	0,30590	0,6179	0,9825
0,6027	0,933	0,13352	0,38651	0,5407	0,27830	0,6988	0,9738
0,6133	0,935	0,13367	0,29030	0,4734	0,31410	0,6914	0,9709
0,6487	0,927	0,14001	0,37850	0,5205	0,28055	0,6872	0,9772
0,6206	0,913	0,14298	0,29449	0,5831	0,27510	0,5950	0,9507
0,6131	0,972	0,13738	0,33802	0,4877	0,27890	0,5891	0,9814
0,5875	0,1009	0,14163	0,24174	0,5847	0,30560	0,5731	0,9587
0,6336	0,929	0,14334	0,27654	0,5023	0,29250	0,6842	0,9802
0,5992	0,948	0,14348	0,28195	0,5106	0,31600	0,7032	0,10164

**Table 1** Results of co-evolutionary framework for public datasets of ITC2007 examination timetabling track

Name	dataset 1	dataset2	dataset3	dataset4
Müller	(4574.9, 159.7731)	(414, 11.49879)	-	-
Gogos	(6064, 108.8087)	(1048.6, 32.57879)	(14133.5, 227.9553)	-
Swan	(6141.4, 173.3187)	(940.1, 31.89375)	(13834.8, 440.9013)	-
Name	dataset5	dataset6	dataset7	dataset8
Müller	(3320.7, 209.9524)	(27808.5, 1115.487)	(4399.143, 123.4942)	(7922.429, 126.9696)
Gogos	(4229.1, 75.01178)	-	(6759.5, 100.5134)	(10809, 180.8265)
Swan	(5328.5, 420.9968)	(29349.5, 1567.905)	(6435.9, 533.9795)	(9750.8, 181.7635)

**Table 2**  $(\bar{x}, \sigma)$  of feasible solutions for public ITC2007 datasets

of feasible solutions, failing only on dataset 4 in this respect. This is perhaps a counter-intuitive result, since one might expect the ‘repair’ aspect of our ‘ruin-and-repair’ strategy to encounter many infeasible solutions. Hence, there is further work to be done in explaining the underlying reasons for this behaviour.

## References

- Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Handbook of Meta-Heuristics, chap. Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pp. 457–474. Kluwer (2003). URL <http://www.asap.cs.nott.ac.uk/publications/pdf/hhchapv002.pdf>
- Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.* **5**(4), 691–703 (1976)
- Gogos, C., Alefragis, P., Housos, E.: A multi-staged algorithmic process for the solution of the examination timetabling problem. In: The 7th International Conference on the Practice and Theory of Automated Timetabling (2008)
- McCullum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Gaspero, L.D., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS JOURNAL ON COMPUTING* **22**(1), 120–130 (2010). DOI 10.1287/ijoc.1090.0320
- Müller, T.: ITC2007 solver description: a hybrid approach. *Annals OR* **172**(1), 429–446 (2009)
- Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.* **12**(1), 3–23 (2008)

## Semidefinite Programming in Timetabling II: Algorithms

Jakub Mareček · Andrew J. Parkes

*Introduction* Semidefinite programming (SDP) is a subfield of convex optimisation (Wolkowicz, Saigal, & Vandenberghe, 2000). It has recently gained considerable attention, as it makes it possible to derive strong lower bounds for minimisation problems in combinatorial optimisation (Goemans & Rendl, 2000), as well as to obtain very good solutions using randomised rounding. In some of the present-best approximation algorithms, both lower and upper bounds are obtained in this fashion. There seem to be, however, only few applications to practical scheduling, timetabling, or rostering problems.

At PATAT 2010, (Burke, Mareček, & Parkes, 2011) presented an SDP relaxation of bounded graph colouring, which can be used to detect the infeasibility in many timetabling problems. Subsequently, they have introduced relaxation for a number of timetabling problems in an extended version of their paper. In this abstract, we present augmented Lagrangian methods, also known as boundary point methods or proximal methods, for solving such relaxations.

*Semidefinite Programming* Semidefinite programming (SDP, Bellman & Fan, 1963; Alizadeh, 1995; Wolkowicz et al., 2000) is a popular generalisation of linear programming, replacing the vector variable with a square symmetric matrix variable and the polyhedral symmetric convex cone of the positive orthant with the non-polyhedral symmetric convex cone of positive semidefinite matrices. The primal-dual pair in the standard form is:

$$z_p = \min_{X \in \mathcal{S}^n} \langle C, X \rangle \text{ s.t. } \mathcal{A}_A(X) = b \text{ and } X \succeq 0 \quad (\text{P SDP})$$

$$z_d = \max_{y \in \mathbb{R}^m, S \in \mathcal{S}^n} b^T y \text{ s.t. } \mathcal{A}_A^*(y) + S = C \text{ and } S \succeq 0 \quad (\text{D SDP})$$

where  $X$  is a primal variable in the set of  $n \times n$  symmetric matrices  $\mathcal{S}^n$ ,  $y$  and  $S$  are the corresponding dual variables,  $b$  is an  $m$ -vector,  $C$ ,  $A_i$  are compatible matrices, and linear operator  $\mathcal{A}_A(X)$  maps symmetric  $n \times n$  matrices to vectors in  $\mathbb{R}^m$ . The  $i$ th element  $\mathcal{A}_A(X)_i = \langle A_i, X \rangle$ , and the adjoint is  $\mathcal{A}_A^*(y) = \sum_i y_i A_i$ .  $M \succeq N$  or  $M - N \succeq$  denotes  $M - N$  is positive semidefinite. Note that an  $n \times n$  matrix,  $M$ , is positive semidefinite if and only if  $y^T M y \geq 0$  for all  $y \in \mathbb{R}^n$ .

Contact author: J. Mareček, E-mail: jakub@marecek.cz  
School of Mathematics, The University of Edinburgh, JCMB among the King's Buildings, Edinburgh EH9 3JZ.

A. Parkes, E-mail: ajp@cs.nott.ac.uk  
School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK.

All relaxations of (Burke et al., 2011), use linear equalities given by the adjacency matrix of the conflict graph, further linear equalities and further linear inequalities. These are best treated explicitly in the primal-dual pair:

$$z_p = \min_{X \in \mathcal{S}^n} \langle C, X \rangle \text{ s.t. } \mathcal{A}_{A_1}(X) = b_1 \text{ and } \mathcal{A}_{A_2}(X) = b_2 \text{ and } \mathcal{A}_B(X) \geq d \text{ and } X \succeq 0$$

$$z_d = \max_{y_1 \in \mathbb{R}^m, y_2 \in \mathbb{R}^p, v \in \mathbb{R}^q, S \in \mathcal{S}^n} b_1^T y_1 + b_2^T y_2 + d^T v \quad (1)$$

$$\text{s.t. } \mathcal{A}_{A_1}^*(y_1) + \mathcal{A}_{A_2}^*(y_2) + \mathcal{A}_B^*(v) + S = C \text{ and } S \succeq 0 \text{ and } v \geq 0.$$

where  $\mathcal{A}_{A_1}(X), b_1, \mathcal{A}_{A_2}(X), b_2$  are given by the conflict graph and further linear equalities, specific to a particular timetabling problem, respectively,  $d$  is a  $q$ -vector, and linear operator  $\mathcal{A}_B(X)$  maps  $n \times n$  matrices to  $q$ -vectors similarly to  $\mathcal{A}_A$  above. As all linear combinations with non-negative coefficients of positive semidefinite matrices are positive semidefinite,  $X \succeq 0$  should again be seen as a restriction to a convex cone. This extends the approach of Wen, Goldfarb, and Yin (2010), who treat linear inequalities explicitly.

*Semidefinite Programming Solvers* Traditionally, SDP is solved using primal-dual interior point methods (Wright, 1997): From the KKT conditions, comprising of the primal (P SDP) and dual (D SDP) problems and the complementarity condition  $ZX = 0$ , one derives the ‘‘Newton system’’ by relaxing the complementarity condition to  $ZX = \mu I$  or similar. These methods are also referred to as second-order, as they employ the second-order partial derivatives, unlike first-order methods, which use only first derivatives. Povh, Rendl, and Wiegale (2006) observe that implementations of second-order methods for computing theta-like SDP relaxations are currently limited to graphs of about 10,000 edges, which amounts to little more than 100 vertices in the dense graphs found in timetabling applications.

First-order Lagrangian methods have long been used as an alternative. In iteration  $k$  of solving a semidefinite program in standard form, one updates  $X^k$  to  $X^{k+1}$  as follows:

$$(y^{k+1}, S^{k+1}) = \operatorname{argmin}_{y, S} -b^T y + \langle X^k, \mathcal{A}_A^*(y) + S - C \rangle \quad (2)$$

$$X^{k+1} = X^k + \mu^{-1} (\mathcal{A}_A^*(y^{k+1}) + S^{k+1} - C) \quad (3)$$

This approach suffers from two major drawbacks: the convergence may be frail and the minimisation of the Lagrangian (2) may turn out to be expensive. The first drawback may be alleviated by augmenting the Lagrangian (Powell, 1969; Hestenes, 1969) with a Frobenius norm term:

$$L_\mu(X, y, S) = -b^T y + \langle X, \mathcal{A}_A^*(y) + S - C \rangle + \frac{1}{2\mu} \|\mathcal{A}_A^*(y) + S - C\|_F^2 \quad (4)$$

The second drawback can be alleviated by minimising the Lagrangian first for  $y$  and only subsequently for  $S$ . Using fixed point arguments, one can still show the convergence of such two-step minimisation.

*Our Solver* The augmented Lagrangian of the dual (1) is:

$$L_\mu(X, y_1, y_2, v, S) = -b_1^T y_1 - b_2^T y_2 - d^T v \quad (5)$$

$$+ \langle X, \mathcal{A}_{A_1}^*(y_1) + \mathcal{A}_{A_2}^*(y_2) + \mathcal{A}_B^*(v) + S - C \rangle$$

$$+ \frac{1}{2\mu} \|\mathcal{A}_{A_1}^*(y_1) + \mathcal{A}_{A_2}^*(y_2) + \mathcal{A}_B^*(v) + S - C\|_F^2$$

The multiple splitting is elaborated in Algorithm Schema 1.

---

**Algorithm Schema 1** AugmentedLagrangianMethod( $A_1, A_2, B, C, b_1, b_2, d$ )
 

---

- 1: **Input:** Instance  $I = (A_1, A_2, B, C, b_1, b_2, d)$  of SDP (1)
  - 2: **Output:** Primal solution  $Y$ , computed up to a certain precision
  - 3: Set iteration counter  $k = 0$
  - 4: Initialise  $X^k \succeq 0$  with a heuristically obtained colouring
  - 5: Compute matching values of dual variables  $y_1^k, y_2^k, v^k \geq 0$ , and  $S^k \succeq 0$
  - 6: **while** the precision is insufficient **do**
  - 7:   Increase iteration counter  $k$
  - 8:   Update  $y_1^{k+1} = \operatorname{argmin}_{y_1 \in \mathbb{R}^m} L_\mu(X^k, y_1, y_2^k, v^k, S^k)$   
      $= -(A_1 A_1^T)^{-1} (\mu(A_1(X^k) - b_1) + A_1(A_2^T(y_2^k) + B^T(v^k) + S^k - C))$
  - 9:   Update  $y_2^{k+1} = \operatorname{argmin}_{y_2 \in \mathbb{R}^m} L_\mu(X^k, y_1^{k+1}, y_2, v^k, S^k)$   
      $= -(A_2 A_2^T)^{-1} (\mu(A_2(X^k) - b_2) + A_2(A_1^T(y_1^{k+1}) + B^T(v^k) + S^k - C))$
  - 10:   Update  $v^{k+1} = \operatorname{argmin}_{v \in \mathbb{R}^q, v \geq 0} L_\mu(X^k, y_1^{k+1}, y_2^{k+1}, v, S^k)$
  - 11:    $= \operatorname{argmin}_{v \in \mathbb{R}^q, v \geq 0} \left( \left( B \left( X^k + \frac{1}{\mu} \left( A_1^T(y_1^{k+1}) + A_2^T(y_2^{k+1}) + S^k - C \right) \right) - d \right)^T v + \frac{1}{2\mu} v^T (BB^T)v \right)$   
     which is a single-cone second-order cone program (Alizadeh & Goldfarb, 2003)
  - 12:   Update  $S^{k+1} = \operatorname{argmin}_{S \in \mathcal{S}, S \succeq 0} L_\mu(X^k, y_1^{k+1}, y_2^{k+1}, v^{k+1}, S)$   
      $= \operatorname{argmin}_{S \in \mathcal{S}, S \succeq 0} \left\| S - \left( C - A_1^T(y_1^{k+1}) - A_2^T(y_2^{k+1}) - B^T(v^{k+1}) - \mu X^k \right) \right\|_F^2$   
     which can be solved by spectral decomposition of the term subtracted from  $S$  (Stewart, 1993)
  - 13:   Choose any step-length  $\mu \geq 0$
  - 14:   Update  $X^{k+1} = X^k + \frac{A_1^T(y_1^{k+1}) + A_2^T(y_2^{k+1}) + B^T(v^{k+1}) + S^{k+1} - C}{\mu}$
  - 15: **end while**
  - 16: Return  $X$
- 

An important aspect of implementing the augmented Lagrangian method is problem-specific simplification of linear algebra involved. In relaxations of bounded graph colouring of a graph on  $n$  vertices, one can exploit properties of the relaxation to:

- not compute  $(A_1 A_1^T)^{-1}$
- compute  $A_1^T y_1$  in time  $O(m)$
- compute  $(A_2 A_2^T)^{-1}$  in time  $O(n^2)$
- compute  $A_2^T y_2$  in time  $O(n)$
- compute  $(AB^T)^{-1}$  in time  $O(n)$
- compute  $B^T v$  in time  $O(n)$
- evaluate the augmented Lagrangian and its gradient at a given  $v$  in time  $n^2$

This allows for an efficient implementation using a variant of limited memory BGFS search with projection to non-negative  $v$  to minimise the quadratic program on Line 11. The bulk of the run-time is hence spent an eigenvalue decomposition in Line 12. Our particular implementation relies on Intel Math Kernels.

*Recovering an Assignment* Finally, since the seminal paper of Karger, Motwani, and Sudan (Karger, Motwani, & Sudan, 1998), there has been a continuing interest in algorithms recovering a colouring from semidefinite relaxations. Typically, such algorithms are based on simple randomised iterative rounding of the semidefinite programming relaxation. One such algorithm, specialised to simple timetabling is displayed in Algorithm Schema 2.

*Conclusions* SDP solvers are less well-developed than LP solvers, in general. Compared to interior point methods, augmented Lagrangian drastically reduce the dependence of per-iteration run-time on the number of constraints. Preliminary computational results suggest this method is practical for instances with up to millions of edges in the conflict graph.

**Algorithm Schema 2** IterativeRounding( $X$ ) based on Karger, Motwani, and Sudan

---

```

1: Input: Matrix variable  $X$  of the solution to the SDP (??) of dimensions  $n \times n$ , bound  $m$ , number  $a_{\max}$  of
   randomisations to test, plus the input to Simple Timetabling, if required
2: Output: Partition  $P$  of the set  $V = 1, 2, \dots, n$ 
3: Compute vector  $v, X = v^T v$  using Cholesky decomposition
4: for Each attempted randomisation  $a = 1, \dots, a_{\max}$  do
5:   Initialise  $P_a = \emptyset, i = 1, X = V$ 
6:   while There are uncoloured vertices in  $X$  do
7:     Pick a suitable  $c = \sqrt{\frac{2(k-2)}{k \log_e \Delta}}$  for  $\Delta$  being the maximum degree of the vertices in  $X$ 
8:     Generate a random vector  $r$  of dimension  $|X|$ 
9:     Pick  $R_i \subseteq X$  of at most  $m$  elements in the descending order of  $v_i r_i$ , where (1) positive and (2) independent
       of previously chosen and, in Simple Timetabling, (3) the respective events fit within the rooms
       and (4) require only features available
10:    Update  $P_a = P_a \cup \{R_i\}, X = X \setminus R_i, i = i + 1$ 
11:   end while
12: end for
13: Return  $P_a$  of minimum cardinality

```

---

**References**

- Alizadeh, F. (1995). Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5(1), 13–51.
- Alizadeh, F., & Goldfarb, D. (2003). Second-order cone programming. *Math. Program.*, 95(1, Ser. B), 3–51. (ISMP 2000, Part 3 (Atlanta, GA))
- Bellman, R., & Fan, K. (1963). On systems of linear inequalities in Hermitian matrix variables. In *Proc. Sympos. Pure Math., Vol. VII* (pp. 1–11). Providence, R.I.: Amer. Math. Soc.
- Burke, E. K., Mareček, J., & Parkes, A. J. (2011). Semidefinite programming relaxations in timetabling. Available from <http://cs.nott.ac.uk/~jxm/timetabling/bounding-bounded.pdf>
- Goemans, M. X., & Rendl, F. (2000). Combinatorial optimization. In *Handbook of semidefinite programming* (Vol. 27, pp. 343–360). Boston, MA: Kluwer Acad. Publ.
- Hestenes, M. R. (1969). Multiplier and gradient methods. *J. Optimization Theory Appl.*, 4, 303–320.
- Karger, D., Motwani, R., & Sudan, M. (1998). Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2), 246–265.
- Povh, J., Rendl, F., & Wiegele, A. (2006). A boundary point method to solve semidefinite programs. *Computing*, 78(3), 277–286.
- Powell, M. J. D. (1969). A method for nonlinear constraints in minimization problems. In *Optimization (Sympos., Univ. Keele, Keele, 1968)* (pp. 283–298). London: Academic Press.
- Stewart, G. W. (1993). On the early history of the singular value decomposition. *SIAM Rev.*, 35(4), 551–566.
- Wen, Z., Goldfarb, D., & Yin, W. (2010). Alternating direction augmented lagrangian methods for semidefinite programming. *Math Program Comput.*
- Wolkowicz, H., Saigal, R., & Vandenberghe, L. (Eds.). (2000). *Handbook of semidefinite programming*. Boston, MA: Kluwer Academic Publishers. (Theory, algorithms, and applications)
- Wright, S. J. (1997). *Primal-dual interior-point methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM).



---

## A GRASP Algorithm for the University Timetabling Problem

Wallace de Souza Rocha · Maria Claudia  
Silva Boeres · Maria Cristina Rangel

Received: date / Accepted: date

**Abstract** The university timetabling problem is one of great interest in the field of combinatorial optimization. Given a set of classes, students, teachers and rooms, the problem consists of assigning lectures or exams to a limited number of available timeslots and rooms, subject to a set of constraints mostly dependent on the particularities of the school. These constraints are classified as hard or soft. The hard constraints must be always satisfied. For example, a student cannot attend more than one class at the same timeslot. A solution for the timetabling problem is said to be feasible when it does not violate any hard constraint. The soft constraints are those which do not generate infeasibility, but reflect some preferences of teachers, students or even schools. For example, we can penalize a timetabling solution with large gaps between classes. The more soft constraints that are satisfied, the better the timetable. There are many timetabling formulations in the literature, but all of them can be grouped in three categories: school, university and exam scheduling. In this work, we develop an algorithm to solve the University Timetabling Problem in the context of the formulation adopted in the ITC-2007 competition [1]. The main advantage of adopting this formulation is that many authors have worked with it, making comparison of results from different researchers easier. We decided not use the ITC-2011 formulation [2], because our main focus is on university timetables. Many metaheuristics have been used to solve this problem, but none of them was considered as the best for this problem. Good results have been found with Simulated Annealing [3–5], Genetic Algorithm [6–8] and Tabu Search [9], among others. There are also some hybrid techniques combining several metaheuristics and exact methods, generally, each heuristic is consid-

---

Wallace de Souza Rocha  
E-mail: walacesrocha@yahoo.com.br

Maria Claudia Silva Boeres  
E-mail: boeres@inf.ufes.br

Maria Cristina Rangel  
E-mail: crangel@inf.ufes.br

ered in various phases of these algorithms. The metaheuristic GRASP (Greedy Randomized Adaptive Search Procedure) is a technique that stands out in the combinatorial optimization field [10]. It has been applied to set covering problems, spanning tree, among others. Some researches of this algorithm have been found for timetabling problems, but all of them for school timetabling formulation [11,12]. This work implements a GRASP algorithm for generating timetables using the formulation of the ITC-2007. The algorithm has an initial phase where a greedy randomized solution is produced. The classes are ranked in order of difficulty (most difficult to easiest) and are selected one by one to enter the timetable. To choose a timeslot and room for a given class, a restricted list is constructed by counting how many violations of the soft constraints there are with each choice. When trying to insert a class into the timetable and a position does not exist, another class (or classes) previously scheduled is removed from the timetable to open a slot for the problem class. A feasible timeslot is selected randomly and all conflicting lectures allocated in that timeslot are removed from the timetable. If a conflicting lecture does not exist in timeslot, a non-conflicting lecture is selected. With this strategy, called explosion, we can generate feasible timetabling solutions for all competition instances. The random selection of lectures in explosion algorithm avoids lectures cycling. For the GRASP improvement phase, a local search is applied to the initial timetabling solution. The GRASP iteration (initial phase and improvement phase) is repeated several times generating different timetables. The final solution is the best of all generated timetables. Three different local search strategies are presented. The simplest uses a depth-first strategy. The neighbours are generated with two movements: MOVE and SWAP. The first reschedules a lecture in a empty timeslot. The second exchanges the timeslots of two lectures. The algorithm stops if  $n$  consecutive neighbours are generated and the objective function is not decreased. The second local search method is an adaptation of the breadth-first algorithm, where the neighborhood is not explored extensively: only  $k$  neighbours are generated and the best one is chosen to the next iteration. The neighbourhood generation and exploration are identical to the first method. The third local search is a depth-first strategy with a heuristic to reduce the violation of soft constraints. Also, a parallel version of the algorithm is presented. The implementation was tested with all the ITC-2007 instances. The results obtained are compared with the best solutions found in the Curriculum-Based Course Timetabling site [13]. Most instances are difficult to find the optimal solution, but we could do this for two instances.

**Keywords** Timetabling · Metaheuristic · GRASP

## References

1. PATAT. International timetabling competition. <http://www.cs.qub.ac.uk/itc2007/> (2008)
2. PATAT. International timetabling competition. <http://www.utwente.nl/ctit/itc2011/> (2011)

3. P. Kostuch. The university course timetabling problem with a 3-phase approach (2006)
4. R. Bai, J. Blazewicz, E. Burke, G. Kendall, B. McCollum, *4OR: A Quarterly Journal of Operations Research* **10**, 43 (2012). URL <http://dx.doi.org/10.1007/s10288-011-0182-8>
5. M.A.S. Elmohamed, P. Coddington, G. Fox, in *Lecture Notes in Computer Science* (1998)
6. W. Erben, J. Keppler. A genetic algorithm solving a weekly course-timetabling problem (1995)
7. Suyanto, in *Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science*, vol. 6114, ed. by L. Rutkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, J. Zurada (Springer Berlin / Heidelberg, 2010), pp. 229–236
8. H. Kanoh, Y. Sakamoto, *Int. J. Know.-Based Intell. Eng. Syst.* **12**(4), 283 (2008). URL <http://dl.acm.org/citation.cfm?id=1460198.1460201>
9. A. Elloumi, H. Kamoun, J. Ferland, A. Dammak, in *Proceedings of the 7th PATAT Conference, 2008* (2008)
10. M. Resende, C. Ribeiro, *GRASP: Greedy Randomized Adaptive Search Procedures*, 2nd edn. (Springer, 2012)
11. M.J.F. Souza, N. Maculan, L.S. Ochi, (Kluwer Academic Publishers, Norwell, MA, USA, 2004), chap. A GRASP-tabu search algorithm for solving school timetabling problems, pp. 659–672. URL <http://dl.acm.org/citation.cfm?id=982409.982441>
12. A. Vieira, M. Rafael, A. Scaraficci. A grasp strategy for a more constrained school timetabling problem (2010)
13. L.D. Gaspero, A. Schaerf. Curriculum-based course timetabling. <http://tabu.diegm.uniud.it/ctt/> (2012)

---

## Using integer linear programming methods for optimizing the real-time pump scheduling

Louise Brac de la Perriere (\*) · Antoine Jouglet (\*) · Alexandre Nace (\*\*) · Dritan Nace (\*)

Received: date / Accepted: date

**Abstract** The work presented in this talk deals with the management of a drinking water distribution network in terms of planning the use of different installations (treatment works, pumping stations, and valves) in order to convey water from sources (rivers, borings, springs,...) to supply areas.

More precisely we study the real-time pump scheduling problem. Being given a water distribution network and some previsions on the consumption at different nodes of the network during a considered time horizon, the main problem is to schedule water pump jobs under the constraint to satisfy water demands with the quality standards settled by French and European legislation, while minimizing the operating costs (treatment and electricity).

These operations should satisfy technical constraints as the respect of the minimum and maximum level of tanks, some contractual constraints as the respect of power levels defined by electricity supplier contracts, and last take some specific water distribution network constraints related to the impact of pressure in modeling or the need of the raw water flow to be as smooth as possible. The task is difficult because of the number and variety of operational constraints that exist in a water distribution system. Recently, this problem becomes more relevant because of the future liberalisation of the electric market that will make difficult the knowledge of the less expensive time slots.

---

(\*) Authors

Laboratoire Heudiasyc UMR CNRS 7253 Universite de Technologie de Compiegne Centre de Recherches de Royallieu BP 20529 60205 COMPIEGNE cedex FRANCE

Tel.: +33 344 23 46 45

Fax: +33 344 23 44 77

(\*\*) Author

Lyonnaise des Eaux, Ondeo Systems, 38 rue du president Wilson 78230 Le Pecq FRANCE

Tel : +33 134 80 23 45

Fax : +33 134 80 53 80

In this talk we will provide a study on the different constraints needed to be modeled and mathematical models for each of them. One of the main problems encountered in modelling is how to deal with hydraulic constraints in our network. This issue is of primary importance as it will lead the choice of the resolution tool to be used.

**Keywords** real-time · pump-scheduling · water distribution system · integer linear programming

## 1 Context

Different approaches exist in the literature to solve this problem: linear programming [1], non-linear programming [5], ant colony optimization [2], genetic algorithm [4], etc.

Each method has advantages and drawbacks and a strategy could be more or less efficient depending on the characteristics of the studied network. In our case, we have to consider the following network characteristics:

- The considered networks are very large (about 100 pumps and 50 storages in our example);
- The allowed computation time to propose optimized solutions is relatively short (30 minutes maximum in the application in hand) due to the real-time scheduling constraint;
- The discrete behaviour of pump has to be expressed through the model. Indeed, the pumping stations work in some defined level of pumping depending on the type and the power of the station. Furthermore, the changes on pumping regime can be operated only periodically and not anytime.

Therefore, we chose to study linear programming, for its velocity in execution even with a lot of variables. With such an approach, the main difficulties are:

- Modelling through linear programming new constraints not expressed before;
- Express in a linear form the hydraulic constraints of the system;
- Ensuring limited CPU time while dealing with integer linear programming models of large size.

## 2 An integer linear programming model

Some previous search work [3] have already described some constraints with linear programming such as the respect of water demand, the respect of minimal and maximal level authorized for each storage, pumps that cannot work at the same time and pumps that have to work at the same time. In this talk, we show how to modelize new constraints:

- Respect the maximal number of switching on for some pumps;

- Ensure the required water quality level through mixing the water in some storage (for instance by guaranteeing a percentage of different water sources);
- Avoid stagnation in the storage to provide water quality;
- Include transfer delay when using very long pipe;
- Represent the hydraulic behaviour of the network.

If for the first four constraints one can employ some conventional tools to express them through linear equations, the last one is more difficult. Moreover, handling this is of particular importance as the hardness of linearising the hydraulic constraints is the first reason cited in literature to disqualify the use of linear programming for the pump-scheduling problem. It was therefore especially important for us to achieve considering it and showing that, at least for our case, these constraints can be written down through linear equations.

Let us first give an example where such constraints are encountered : when there are different gravitational pipes bringing water from different storages to the same consumption area, the quantity of water coming from each storage depends on the pressure in each pipe and storage. Then, the equations representing this phenomena are highly non linear.

Another problem with pressure was the existence of water exchanges between storages serving the same consumption area, because of the difference of pressure between them: they tend to balance their level of water. We solved the problem studying experimentally the behaviour between two storages, with and without consumption. We found that this behaviour can be linearised with coefficient that can be obtained experimentally.

There is therefore a preliminary work to do when implementing this method on a new network, to calculate the coefficients for each hydraulic problematic configuration. We will provide some experimental results and schemes illustrating this behaviour and justify our findings.

The model has been tested on a real network composed of more than 130 pumps and 30 storages, and several scenarios and hypothesis are already considered. We will report some numerical results in the conference and discuss issues on the efficiency and accuracy of using linear models for such problems.

## References

1. F. Guhl. *Gestion optimale des réseaux d'eau potable*. PhD thesis, Université Louis Pasteur, 1999.
2. M. López-Ibáñez, T. D. Prasad, and B. Paechter. Ant colony optimization for optimal control of pumps in water distribution networks. *Journal of Water Resources Planning and Management*, 2008.
3. D. Nace, S. Demotier, J. Carlier, T. Daguinos, and R. Kora. Using linear programming methods for optimizing the real-time pump scheduling. In *World Water & Environmental Resource Congress*, 2001.
4. J. Nicklow. State of the art for genetic algorithms and beyond in water resources planning and management. *Journal of Water Resources Planning and Management*, 2010.
5. B. Ulanicki, J. Kahler, and H. See. Dynamic optimization approach for solving an optimal scheduling problem in water distribution systems. *Journal of Water Resources Planning and Management*, 2007.

## A study of hyper-heuristics for examination timetabling

Ender Özcan · Anas Elhag · Viral Shah

Received: date / Accepted: date

### 1 Introduction

Examination timetabling is both a difficult and time consuming task, faced by many educational institutions worldwide [5]. The main objective is to assign periods within a specified examination timeframe and rooms to exams, whilst satisfying a range of constraints. There are two common constraint categories: *hard* and *soft*. It is imperative that all hard constraints are satisfied in a given solution, which is then referred to as a *feasible* solution. For example, students must not sit two or more exams simultaneously in the same period. Soft constraints, on the other hand, represent preferences that are not essential but should be satisfied as much as possible. For example, a student should not sit two exams in two consecutive periods on the same day. Once a feasible solution is obtained, the degree to which the soft constraints are violated is used to evaluate the quality of a timetable.

Most of the solutions to examination timetabling problems have been developed due to a need at an educational institution. Hence, different types of examination timetabling problems can be found in the literature which are solved using different types of methodologies. This could be considered as richness, but there is a downside that is comparison of the approaches becomes extremely difficult. The state-of-the-art for any problem is of interest to both practitioners and researcher. A recent competition on examination timetabling was arranged as a part of ITC2007<sup>1</sup>. The instances used in this competition reflects the real world examination timetabling complexities. The winner of the

---

E. Özcan, A. Elhag and V. Shah  
University of Nottingham, School of Computer Science  
Jubilee Campus, Nottingham NG8 1BB UK  
Tel.: +44 (115) 95 15544  
Fax: +44 (115) 846 7877  
E-mail: {exo, axe}@cs.nott.ac.uk and viralshahkach@hotmail.com

<sup>1</sup> <http://www.cs.qub.ac.uk/itc2007/>

examination timetabling competition is a hybrid multistage approach which is described in [4].

Hyper-heuristics are methodologies that perform search via generation or selection of heuristics in problem solving [3]. A goal in hyper-heuristic research is designing methodologies which are capable of solving problem instances having diverse properties automatically without requiring any parameter tuning. There are a few benchmarks for examination timetabling. The most commonly used one is Toronto benchmark. The performance of hyper-heuristics have been investigated on Toronto and Yeditepe problem instances [1,2] as well as ITC2007 instances, which includes instances for Examination Timetabling and Course Timetabling. In this study, we present performance analysis of some selection hyper-heuristics on the Examination Timetabling instances of ITC2007.

## 2 Experimental Results

A subset of ITC2007 instances are used during the experiments. The characteristics of each benchmark instance are summarised in Table 1. An initial timetable is constructed, firstly assigning examinations with room hard constraints to rooms and periods with just enough capacities and lengths, followed by assigning examinations with period hard constraints in a similar fashion. Finally, a weighted graph is used to determine the order in which to timetable the remaining exams. If the resulting timetable is infeasible, it is reset and the entire process of timetabling starts again, first considering exams with room hard constraints and so on. Then the solution in hand is iteratively improved using a selection hyper-heuristic which perturbs this solution generating a new one using a chosen low level heuristic and then decides whether to accept or reject the new solution. Different combinations of heuristic selection {Simple Random (SR), Greedy (GR), Reinforcement Learning (RL)} and acceptance {Improving Only (IO), Improving and Equal (IE), Great Deluge (GD)} methods are used as hyper-heuristics during the experiments. Six different perturbative low level heuristics were implemented. The main objective of these low level heuristics is to make slight modifications on the current timetable, in an attempt to lower the soft constraint violations, such as rescheduling of rooms, or swapping exams.

Each experiment is repeated 50 times and a run is terminated after 500 seconds complying with the ITC2007 competition rules. The experiments are carried out on a 2.83GHz Intel Core 2 Duo E8300 XP machine with a memory of 3.23GB. The results are provided in Figure 2. Feasible solutions are obtained for almost all problem instances, except for Exam4. In the overall, Reinforcement Learning performs better than the rest of the heuristic selection methods, while as an acceptance method, Great Deluge is better than the others. Table 3 shows a comparison between our approach and the approaches of the winners of the competition.



**Table 1** The characteristics of the ITC2007 examination timetabling problem instances.

Problem	No. of exams	No. of students	No. of rooms	No. of time-slots	Conflict density
Exam1	607	7891	7	54	5.05
Exam2	870	12743	49	40	1.17
Exam3	934	16439	48	36	2.62
Exam4	273	5045	1	21	15.00
Exam5	1018	9253	3	42	0.87
Exam6	242	7909	8	16	6.16
Exam7	1096	14676	15	80	1.93
Exam8	598	7718	8	80	4.55

The details of the hyper-heuristic approach and more results using additional hyper-heuristics will be provided at the conference.

## References

1. Bilgin, B., Özcan, E., Korkmaz, E.E.: An experimental study on hyper-heuristics and exam timetabling. In: Practice and Theory of Automated Timetabling VI, *Lecture Notes in Computer Science*, vol. 3867, pp. 394–412. Springer (2007)
2. Burke, E., Kendall, G., Misir, M., Özcan, E.: Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research* pp. 1–18 (2010)
3. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent developments. In: C. Cotta, M. Sevaux, K. Sorensen (eds.) Adaptive and Multilevel Metaheuristics, *Studies in Computational Intelligence*, vol. 136, pp. 3–29. Springer Berlin Heidelberg (2008)
4. Muller, T.: Itc2007 solver description: A hybrid approach. *Annals of Operations Research* **172**(1), 429–446 (2009)
5. Qu, R., Burke, E.K., McCollum, B., Merlot, L., Lee, S.: A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling* **12**(1), 55–89 (2009)

**Table 2** Soft constraints score for the datasets.

Hyper- heuristics	Exam 1			Exam 2		
	min	$\mu$	$\sigma$	min	$\mu$	$\sigma$
RL-EAI	8685	8879.14	774.74	<b>778</b>	<b>800.3</b>	<b>32.9547</b>
RL-I	9608	9858.5	685.562	790	811.44	30.7639
RL-GD	9460	9636.12	770.975	<b>778</b>	<b>800.3</b>	<b>32.9547</b>
SRP-EAI	<b>8584</b>	<b>8825.16</b>	<b>851.734</b>	814	845.82	42.0096
SRP-I	9017	9279.02	902.688	807	845.98	26.6906
SRP-GD	9142	9477.22	842.323	789	826.56	46.194
G-EAI	9178	9387.08	1088.85	799	827.96	44.3133
G-I	9179	9428.28	1150.74	783	812.54	42.8858
G-GD	9178	9387.08	1088.85	787	813.4	50.3648
	Exam 3			Exam 4		
	min	$\mu$	$\sigma$	min	$\mu$	$\sigma$
RL-EAI	32662	35409.5	3574.89	infeasible	n/a	n/a
RL-I	33240	35431.6	2780.72	infeasible	n/a	n/a
RL-GD	31260	34259.4	3064.11	infeasible	n/a	n/a
SRP-EAI	35210	37026.9	2941.63	infeasible	n/a	n/a
SRP-I	34386	39293.4	3101.42	infeasible	n/a	n/a
SRP-GD	31493	34052.3	3657.36	infeasible	n/a	n/a
G-EAI	34071	36090.1	1269.97	infeasible	n/a	n/a
G-I	33574	35340.7	1094.84	infeasible	n/a	n/a
G-GD	<b>31227</b>	<b>32974.6</b>	<b>1901.22</b>	infeasible	n/a	n/a
	Exam 5			Exam 6		
	min	$\mu$	$\sigma$	min	$\mu$	$\sigma$
RL-EAI	<b>7541</b>	<b>7588.2</b>	<b>101.002</b>	30415	31634.1	2464.28
RL-I	<b>7541</b>	<b>7588.2</b>	<b>101.002</b>	30625	30640.1	57.1829
RL-GD	<b>7541</b>	<b>7588.2</b>	<b>101.002</b>	<b>29695</b>	<b>30126.9</b>	<b>1255.71</b>
SRP-EAI	7677	7726.98	100.82	33775	34086.7	723.078
SRP-I	7677	7726.98	100.82	37485	37503.1	69.5532
SRP-GD	7772	7815.56	108.451	38175	38283.9	88.3193
G-EAI	7658	7662.6	24.6792	37900	38855.9	152.445
G-I	7658	7662.6	24.6792	37900	38855.9	152.445
G-GD	7658	7662.6	24.6792	37900	38855.9	152.445
	Exam 7			Exam 8		
	min	$\mu$	$\sigma$	min	$\mu$	$\sigma$
RL-EAI	<b>15116</b>	<b>15539</b>	<b>701.098</b>	21678	27446.3	4199.8
RL-I	16722	16912	579.296	20978	21178.4	253.238
RL-GD	15178	15549.1	631.938	23389	23583.4	405.198
SRP-EAI	15291	15492.3	372.928	21812	22207.3	660.057
SRP-I	16941	17185.1	363.882	21522	22056.6	809.114
SRP-GD	15660	16010.3	476.773	22657	23273.3	810.811
G-EAI	16622	16739.8	322.287	<b>20168</b>	<b>20562.6</b>	<b>710.712</b>
G-I	16796	16863	216.641	20236	20990.9	806.43
G-GD	16622	16739.8	322.287	<b>20168</b>	<b>20562.6</b>	<b>710.712</b>

**Table 3** Comparison between our results and the results of the top winners of the competition.

Ranking	Exam1		Exam2		Exam3	
	Winner	Score	Winner	Score	Winner	Score
1 <sup>st</sup>	Muller	4370	Muller	400	Muller	10049
2 <sup>nd</sup>	Gogos	5905	De Smet	623	Gogos	13771
3 <sup>rd</sup>	De Smet	6670	<b>Özcan</b>	<b>778</b>	Pillay	15917
4 <sup>th</sup>	Atsuta	8006	Gogos	1008	Atsuta	17669
5 <sup>th</sup>	<b>Özcan</b>	<b>8584</b>	Pillay	2886	<b>Özcan</b>	<b>31227</b>
6 <sup>th</sup>	Pillay	12035	Atsuta	3470	De Smet	Infeasible

	Exam4		Exam5		Exam6	
	Winner	Score	Winner	Score	Winner	Score
1 <sup>st</sup>	Muller	18141	Muller	2988	Muller	26585
2 <sup>nd</sup>	Gogos	18674	De Smet	3847	Gogos	27640
3 <sup>rd</sup>	Atsuta	22559	Gogos	4139	De Smet	27815
4 <sup>th</sup>	pillay	23582	Atsuta	4638	Atsuta	29155
5 <sup>th</sup>	<b>Özcan</b>	<b>Infeasible</b>	Pillay	6860	<b>Özcan</b>	<b>29695</b>
6 <sup>th</sup>	De Smet	Infeasible	<b>Özcan</b>	<b>7541</b>	Pillay	32250

	Exam7		Exam8	
	Winner	Score	Winner	Score
1 <sup>st</sup>	Muller	4213	Muller	7742
2 <sup>nd</sup>	De Smet	5420	Gogos	10521
3 <sup>rd</sup>	Gogos	6572	Atsuta	14317
4 <sup>th</sup>	Atsuta	10473	Pillay	15592
5 <sup>th</sup>	<b>Özcan</b>	<b>15116</b>	<b>Özcan</b>	<b>20168</b>
6 <sup>th</sup>	Pillay	17666	De Smet	Infeasible

---

# Nurse Timetabling: Linking Research and Practice

Julie Lane<sup>1</sup>, Barry McCollum<sup>2</sup>,

*1Facilities Directorate  
Sheffield Hallam University  
Howard Street  
Sheffield  
South Yorkshire  
S1 1WB  
j.lane@shu.ac.uk*

*2School of Electronics, Electrical Engineering and Computer Science  
Queen's University,  
Belfast,  
University Road,  
N. Ireland,  
BT7 1NN  
b.mccollum@qub.ac.uk*

Timetabling of courses within the UK University Sector is extremely challenging due to the number of variables and the complex nature of the constraints associated with the management and planning of teaching sessions and those around institutional resource usage issues. (McCollum 2007 a, b). With students paying up to £9K a year for tuition fees and the Higher Education Funding Council for England (HEFCE) dictating greater transparency and value for money in ensuring funds are used for the intended purposes (<http://www.hefce.ac.uk/finance/>), Higher Education Institutions (HEIs) have been encouraged to focus their strategic objectives in attempting to improve the overall student experience (Lord Brown Report 2009). As part of this focus, the delivery of the Institutional Timetable plays a significant role. The Institutional Timetable can be thought of as the window to which the student views the University and whether this view is positive or negative is reflected in question 13 of the National Student Survey. As results can be seen within the public arena (<http://unistats.direct.gov.uk/>), Universities are realising the importance of delivering quality within the timetable while attempting to continue to make best use of the available resource.

Sheffield Hallam University in the UK is divided into four faculties which is further divided by a number of departments. The Department of Nursing and Midwifery along with the Departments of Social Work, Social Care and Community Studies, Department of Allied Health Professions, Department of Biosciences and Department of Sport all are situated within the Faculty of Health and Wellbeing. The Department of Nursing and Midwifery has approximately 100 academic staff, of which 20% are part-time. In 2009/10 Sheffield Hallam University was the fourth largest University in the UK based on the number of student enrolments. All full-time, undergraduate students receive an individual, on-line timetable. Timetables are maintained in a live environment; meaning that changes to teaching delivery during the academic year, are reflected on the personalised on-line timetables. Of all the subject areas within the University, Nursing is one of the most complex and difficult to schedule. At the January Academic Registers' Council (ARC) Timetabling Practitioners Conference in the UK, Julie Lane held a workshop where a number of

HEIs shared the challenges and constraint commonly faced. The Academic Registers' Council (ARC) Timetabling Practitioners Group is a UK sector group, which has been established to share good practice, and keep abreast of up-to-the-minute changes in legislation which impacts on timetabling, and also provides a network to discuss a broad range of challenges which Timetabling Managers face. The work outlined in this abstract is an extension of this initial investigation into the identified practical issues. Specifically in relation to nurse scheduling, there are external contributing factors laid out by the Department of Health and National Health Service which contribute to constraining the way courses are delivered. Module delivery patterns are complex. Irregular week patterns combined with changing durations for each individual teaching session has a significant impact on the ability to manage resources of staff and space to meet the peaks and troughs of demand throughout the year, and success often hinges on the flexibility and costs of provision. Each module is likely to have a unique set of pedagogic requirements whether this is taking into account the specialisms of academic staff, managing specialist rooms and equipment or the ability to manage peaks and troughs of demand throughout the academic year.

In addressing the issues associated with Nursing timetabling, it is also important to understand the external influences which are a significant driver. Until recently the funding of nursing education has been provided through the Department of Health (DOH), with the commissioning of Higher Education programmes devolved to regional Strategic Health Authorities (SHAs). Numbers of commissions and contracts are agreed between SHAs and HEIs. Nurses' tuition fees are paid for by Strategic Health Authorities and in some cases they are paid a monthly bursary too. However there are a number of key changes which will affect Nurse Education and therefore the associated delivery issues. SHAs will be broken up over the next year ([http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH\\_117353](http://www.dh.gov.uk/en/Publicationsandstatistics/Publications/PublicationsPolicyAndGuidance/DH_117353)) and replaced by Local Education and Training Boards (LETBs). These have already been established in regions and are chaired by chief executives of large NHS Trusts. They will commission health courses and greatly influence the NHS workforce. Also nursing is moving to an all graduate profession (currently there is a diploma or degree option). How this will affect commissioning nurse education and ultimately the NHS nursing workforce has not yet been quantified. It may be that fewer nurses are trained; they being the managers of care, and associate practitioners deliver the majority of nursing-type care. Who will train these associate practitioners is not yet clear. These issues along with their influence on the timetable production will be discussed at the conference.

Traditional timetabling follows the academic calendar as defined by each institution. This normally commences late September and finishes May/June and generally consists of two semesters. In contrast, Nursing courses are often delivered all year round and frequently have multiple in-takes of when students commence their studies. Each module of the course will consist of a mixture of classroom based activities, as well as lectures and practical sessions; each with different durations and irregular week patterns. Added to the mix are placements which are controlled by the NHS and IPE (Inter-Professional Education is credit rated and an aspiration of HEIs to successfully deliver. It is recommended internationally, has European quality indicators and is driven by government policy - UK being the most advanced in this respect). All of these factors contribute to peaks and troughs of demand on resources throughout the year. Having initially explored the complexities of what is required; the success to scheduling is the ability of being equipped to manage the resources i.e. staffing and rooms and more importantly specialist rooms. The largest cost to any HEI is wages which therefore dictates that HEIs must maximise the staffing resource. To enhance the student experience in providing added value, there is a key driver to mix and match the specialisms of academic staff and associate them with not just a module but a specific topic within the module which further adds to the levels of complexity to scheduling. Likewise the staff resource itself is likely to fluctuate over the year, whether this is due to managing staff holidays (7 weeks per year), guest speakers, multiple staff required to deliver a single teaching session and perhaps most frustrating for the timetabling practitioner, the lack of information in not knowing the names of staff to associate with teaching activities. The second largest cost to any HEI is associated with running costs of the estate. The balance to maintain a cost efficient estate yet supporting the pedagogic delivery of nursing activities means that there is a driver to maximise efficient use of the estate which is reported through utilisation surveys. For nursing practical sessions, there is a strong need to provide flexible teaching spaces. The impact of this means technical managers have to find solutions to managing equipment. As technology is quickly developed and improved, it is essential that HEIs keep abreast of these developments and invest in the most up to date advances. The cost associated with this often means that purchases are limited and therefore equipment cannot be fixed to a single location. The mix of providing flexible teaching spaces coupled with the need to move equipment from one

location to another based on demand needs, is likely to provide a 'headache' to the technical support manager; who in turn will look to the timetabling practitioner to find a solution.

In summary, external influences significantly restrict the way in which nursing timetables are delivered. External influences coupled with internal pedagogic requirements create peaks and troughs of demand throughout the academic year. Too successfully schedule relies heavily on managing the resources, which in themselves are restricted by financial means. A large number of nursing students are often mature students who have family commitments. To ensure that the student experience is enhanced, the students themselves are looking for a timetable which balances their learning/life balance. This research is aimed at addressing the acknowledged gap which currently exists between research and practice in timetabling research. Specifically the research will allow the description and modeling the real world complexities around academic nurse timetabling in the UK. This work will be compared with the issues raised within the related areas of course timetabling and nurse rostering described in details within the PATAT sponsored 2010 Nurse Rostering Competition Hapeslagh (2010). Once a problem representation drawn from this work is described and made available, work will commence on the development of algorithms that are able to provide workable timetables while maintaining the balance described above.

- McCollum (2008), B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Di Gaspero, R. Qu, E.K. Burke, Setting The Research Agenda in Automated Timetabling: The Second International Timetabling Competition, [http://www.cs.qub.ac.uk/itc2007/ITC2007\\_Background\\_Techreportv1.pdf](http://www.cs.qub.ac.uk/itc2007/ITC2007_Background_Techreportv1.pdf).
- McCollum (2007a), B. McCollum, A Perspective on Bridging the Gap between Theory and Practice in University Timetabling, Practice and Theory of Automated Timetabling VI, Springer LNCS Vol 3867, 2007, pp 3-23.
- McCollum (2007b), B. McCollum, T. Roche, P. McMullan, Optimising Space Through Macro and Micro Planning and Scheduling, Presentation at SCUP-42, Society of College and University Planners International Conference, July 7-11, 2007, Chicago.
- Hapeslagh (2010), S. Hapeslagh, Patrick De Causmaecker, M. Stolevik and A. Schaerf, First International Nurse Rostering Competition 2010, Proceeding of the 8<sup>th</sup> International Conference on the Practice and Theory of International Timetabling, Belfast, 10-31 August, p498 – 501, ISBN 08-538-9973-3
- Lord Brown Report (2009). An independent review of higher education & student finance in England. "Securing a sustainable future for higher education in England", <http://webarchive.nationalarchives.gov.uk/+/hereview.independent.gov.uk/hereview/report/>

---

## Next Steps for the Examination Timetabling Format and Competition

Barry McCollum · Paul McMullan ·  
Tomáš Müller · Andrew J. Parkes\*

June 2012

### 1 Background and Motivations

The second International Timetabling Competition, ITC2007<sup>1</sup>, [4] included a track on examination timetabling with results presented during PATAT-2008. It provided a well-defined representation [3] with many features not appearing in previous benchmarks. For example, the ‘Toronto benchmarks’<sup>2</sup> were highly influential and well-studied (e.g. see [5]) but provided only basic enrolment data and had only a limited mechanism to spread out the exams in time; however, these mechanisms were greatly extended in ITC-2007. A follow-up competition is being arranged, and here we briefly list the most important intended changes (at the time of writing, the exact list and syntax is not yet fixed, and so relatively minor changes are still likely). A challenge in the design of benchmark problems and competitions is to select a compromise between ease of implementation, and the ability to represent all problems that might be encountered in real problems. Accordingly, extensions and changes were selected with the aim they are relatively straightforward and direct to implement, but will make an important contribution to the usability of the format and solvers. Note that changes are inspired by the practical experiences both at Purdue<sup>3</sup> (e.g. see [6]), and also of EventMAP Limited Ltd<sup>4</sup> based at Queen’s University and producing software for many institutions.

### 2 Direct Extensions

By direct extensions we mean those that are closely related to constraints and objectives already used in the previous ITC2007 format; note that the weights assigned to various penalties were collected together into an “Institutional Model (IM)” and many of the proposed changes will simply correspond to new declarations within the IM.

---

Barry McCollum & Paul McMullan  
Queen’s University, Belfast, U.K. E-mail: {B.McCollum,P.P.McMullan}@qub.ac.uk  
Tomáš Müller  
Purdue University, West Lafayette, IN 47907, USA E-mail: muller@unitime.org  
Contact author: Andrew J. Parkes  
University of Nottingham, NG8 1BB, U.K. E-mail: ajp@cs.nott.ac.uk

<sup>1</sup> <http://www.cs.qub.ac.uk/itc2007/>

<sup>2</sup> <ftp://ftp.mie.utoronto.ca/pub/carter/testprob/>

<sup>3</sup> <http://unitime.org/>

<sup>4</sup> <http://www.eventmap-uk.com/>

**Relaxed Conflicts:** Occasionally, some students take a very unusual mix of modules, and the resulting induced conflicts between exams can have a large negative effect on the overall problem, possibly rendering it infeasible or leading to many other students having worse timetables. In such circumstances, it can be better to allow the student to have two of their modules placed into the same time-slot, though with a (high) penalty, and so we will allow a declaration in the IM of the form:

`TwoExamsAtSameTime <W>`

giving a penalty of  $W$  for each student affected and that will hence need special arrangements. Absence of such a declaration will correspond to an infinite penalty as (implicitly) in the ITC2007 format.

**Extended Exam Pattern Penalties:** Multiple copies of the “FrontLoad” and “PeriodSpread” declarations will be permitted. This is an easy extension to implement but will allow a much finer-grained control; permitting dividing exams into multiple size classes rather than just the ‘large’ and ‘small’ of ITC2007. As explained in [3], multiple copies of the period spread will also capture the system used with the Toronto set of benchmark problems. The IM will also allow “NInARow” and/or “NInADay” penalties to extend the current “TwoInARow” and “TwoInADay” penalties in a natural way to  $N=3$ , etc. In combination with `TwoExamsAtSameTime`, this will also allow modelling of cases when a student can demand a special session when they are assigned to multiple exams in the same day.

**Individual Penalties by Pairs of Exam, Room or Period:** Extra sections in the data will specify for each (exam,room) or (exam,period) pair a corresponding specific individual penalty of assigning the exam to that room or period. This is useful for exams having special requirements; they might need facilities only present in given rooms, or alternatively might need a special set of times during which a facility or staff member is available. It will also be possible to penalise pairs of (room,period) to account for rooms being unavailable. Penalties be infinite for cases when a particular combinations is prohibited. Separate penalties for individual “(exam,period,room)” triples, were considered but not thought to be useful enough to warrant the extra complexity.

**Generalised Room Usage Penalties:** In ITC2007, for each room it is only possible to add a penalty for each specific period that it is used. However, for some rooms, their standard configuration might not be suitable for exam usage at all. Preparation of the room for usage at any time during the exam session would have a setup penalty, though the setup would need to be done only once. Hence, it is proposed that the information for each room be extended to:

`<capacity> <per-usage-penalty> <setup-penalty>`

With this, many potential rooms could be given and the solver allowed to select which rooms would best match the problem instance and hence should be prepared for usage. In the extreme case, then the setup penalty might be very high and be taken to correspond to building (or at least remodelling) the room itself. This would allow some (limited) usage for space planning [1] in that it could be used to give insight into what combination of room sizes and types are best suited for the examination timetabling problems.



### 3 Structural Extensions

These are not specific to examination timetabling but are changes to the way that the instances are presented to a solver, and the resources that the solver can use (and will also be retrofitted to previous benchmarks).

**Multiple Time Limits and Cores:** Firstly, the runtime on instances will not be fixed, but allowed to vary. This also has the practical advantage that the same data instance could be used, but with two (radically) different time limits, e.g. with both a 2 minute and a 120 minute limit. This is practically useful as finding suitable data instances can be difficult, and so it is better to make maximal use of them. Also, even desktop machines are rapidly moving to having many cores, and so it is important that future solvers can exploit this. To encourage this, the next competition will have a sub-track using 8 (or more) cores. This can be done trivially using multiple threads each running the same (randomized) solver, but we hope that solvers will make more interesting cooperative usage of the cores.

**Multiple Institutional Models (IMs):** Real-world examination timetabling problems have an objective that contains many terms, and (basically) with the IM providing the weights. A natural way extension to treat them as a multi-objective optimisation (MOO) problem, e.g. see [2]. It was indeed considered to try to make a competition with the evaluation being truly multi-objective. For example, solvers could be expected to produce their best approximation of the Pareto Front, and comparisons could be based on standard MOO evaluation methods such as volume dominance; however, we believed that this would lead to too drastic a change. Instead, a simpler version, “MOO-lite”, is proposed here: that each data instances is simply associated with multiple different IMs. The solver will then be expected to produce a separate solution for each IM, and they will be used for ranking as if solved independently. However, the solver will be free to share the computational resources and intermediate solutions between IMs in any way that it feels appropriate. The hope is that participants will develop methods that are more efficient (and interesting) than simply solving each IM in turn.

### 4 Summary

We have listed the main changes intended for the examinations timetabling format and competition.

### References

1. C. Beyrouthy, E. K. Burke, B. McCollum, P. McMullan, and A. J. Parkes. University space planning and space-type profiles. *Journal of Scheduling*, 13:363–374, August 2010.
2. E. K. Burke, B. McCollum, P. McMullan, and A. J. Parkes. Multi-objective aspects of the examination timetabling competition track. In *Proceedings of PATAT 2008*, pages 3119–3126, 2008.
3. B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke, and R. Qu. A new model for automated examination timetabling. *Annals of Operations Research*, 194:291–315, 2012.
4. B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, R. Qu, and E. K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, Winter 2010.
5. R. Qu, E. K. Burke, B. Mccollum, L. T. Merlot, and S. Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *J. of Scheduling*, 12:55–89, February 2009.
6. H. Rudová, T. Müller, and K. Murray. Complex university course timetabling. *Journal of Scheduling*, 14:187–207, 2011. 10.1007/s10951-010-0171-3.

---

## Directing selection within an extended great deluge optimisation algorithm

Ryan Hamilton-Bryce, Paul McMullan, Barry McCollum

*School of Electronics, Electrical Engineering and Computer Science  
Queen's University,  
Belfast,  
University Road,  
N. Ireland,  
BT7 1NN*

rhamiltonbryce01@qub.ac.uk, p.p.mcmullan@qub.ac.uk, b.mccollum@qub.ac.uk

The challenge of producing scheduling solutions for problems such as Course and Examination timetabling involves a combination of practical and research based approaches [1]. Due to the complexity of the issues involved research has focused on the use of search based heuristic techniques. Within the area of examination scheduling, progress in research has been facilitated by the availability of benchmark data sets [2, 3]. Results using a wide range of techniques have been reported as a result, with varied levels of success based on generality and time taken [4]. A successful technique can be viewed as one which can produce good solutions to a range of differing problems within a problem domain, in a practical timescale.

The 2nd International Timetabling Competition (ITC2007) [2] introduced an examination scheduling track, and new result sets continue to be validated using the competition's online validation service despite the competition closing almost five years ago. The next timetabling Competition (ITC2013) to be announced will further develop the problem definition to take into account additional real world issues. A technique which had previously been very successfully introduced to Course scheduling [5] was adapted and used for the ITC 2007 Examination data sets [6]. This technique is based upon the Great Deluge algorithm and was able to produce feasible and competitive solutions for all of the data sets presented in the ITC2007. It was noted that a link may exist between the initial adaptive construction phase and the stochastic extended great deluge optimisation phase [7].

The technique uses two phases of optimisation for scheduling; construction, to create an initial feasible solution and; improvement, to explore the solution space of the constructed solution for further better solutions. During adaptive based examination timetable construction, an ordered list of "difficult" (hard to schedule) examinations, is maintained [8]. When the improvement phase is reached, this information is discarded in favour of the fully stochastic selection routine, to ensure as much time as possible is spent trying to find an optimal solution. Indeed, when an adaptive based constructor was combined with a stochastic extended great deluge optimiser, it was able to produce results that beat those of the ITC2007 winner in six of the twelve datasets when run on identical hardware [5].

The purpose of this abstract is to investigate the link between an adaptive construction phase and a great deluge based optimisation phase. To this end, we will be investigating two different selection criteria for directing examination selection within the improvement phase. The first selection criteria will use the list of hard to schedule examinations, as it exists in its final state at the end of the construction phase. The second criteria will also initially use the list in this form, however with each iteration of the optimiser this list will be updated with new penalties, therefore the list of difficult to schedule examinations will continue to evolve. In each criteria, we will be investigating the effect of both replacing and supplementing the existing stochastic selection routine. We will also attempt to identify any trends within the adaptive examination list, such as what portions of the list provide the greatest benefit to the optimiser.

For the purposes of this abstract, we will be using the datasets introduced during the ITC2007, which have previously been discussed in great detail [9] to allow us to compare the results generated using this new selection routine against both those presented by B. McCollum, et al [5] and those of the competition winner. The results of this will be presented at PATAT 2012.

## References

1. B.McCollum, A Perspective on Bridging the Gap between Research and Practice in University Timetabling (2007), Practice and Theory of Automated Timetabling VI (eds. E.K.Burke and H.Rudova), Lecture Notes in Computer Science Volume 3867, Springer 2007, pp 3-23
2. B. McCollum, A.Schaerf, B.Paechter, P. McMullan, R.Lewis, A. Parkes, L. Di Gaspero, R.Qu, E. Burke, Setting The Research Agenda in Automated Timetabling: The Second International Timetabling Competition, INFORMS Journal on Computing. Vol 22, No 1, 2010, pp 120-130
3. Examination Timetabling: Algorithmic Strategies and Applications Michael W. Carter, Gilbert Laporte and Sau Yan Lee *The Journal of the Operational Research Society* Vol. 47, No. 3 (Mar., 1996), pp. 373-383
4. R.Qu, E.K.Burke, B.McCollum, L.G.T.Merlot, S.Y.Lee, A Survey of Search Methodologies and Automated System Development for Examination Timetabling, Journal of Scheduling (2009), Volume 12(1), 55-89
5. P. McMullan, An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Lecture Notes in Computer Science, Springer, Vol 4487, pp538-545, 2007.
6. B. McCollum, P.J. McMullan, A.J. Parkes, E.K. Burke, S. Abdullah, "An Extended Great Deluge Approach to the Examination Timetabling Problem", MISTA 2009, Dublin, 10-12 August 2009.
7. E.K. Burke, G. Kendal, B. McCollum, P. McMullan, Constructive versus Improvement Heuristics: An Investigation of Examination Timetabling, 3rd Multidisciplinary International Scheduling Conference: Theory and Applications, 2007
8. Burke, E. K., Newall, J. P., (2004), "Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings", Annals of operations Research
9. B. McCollum, P.J. McMullan, E.K. Burke, A.J. Parkes, Q. Rong (or R. Qu), "A New Model for Automated Examination Timetabling", April 29 2008

---

# Academic Timetabling: Space Sharing Strategies

Hannah White-Overton<sup>1</sup>, Barry McCollum<sup>2,4</sup>, Paul McMullan<sup>2,4</sup>, Edmund Burke<sup>3,4</sup>

*<sup>1</sup>Imperial College London,*

*South Kensington*

*London SW7 2AZ*

*h.white-overton@imperial.ac.uk*

*<sup>2</sup>School of Electronics, Electrical Engineering and Computer Science*

*Queen's University,*

*Belfast,*

*University Road,*

*N. Ireland,*

*BT7 1NN*

*b.mccollum@qub.ac.uk*

*<sup>2</sup>School of Computer Science*

*University of Sterling*

*Scotland,*

*United Kingdom,*

*NG8 1BB*

*<sup>4</sup>EventMAP Ltd*

*SARC Building,*

*Queen's University,*

*Belfast,*

*N. Ireland*

*BT95AH*

Implemented correctly, Automated Timetabling can provide many benefits to Academic Institutions [McCollum 2008, 2007a]. The ability to make better use of Institutional resource is a key area due to the need to deliver the best educational environment possible [McCollum2007b].

Imperial College is based in London, UK and specialises in Science, Technology, Engineering and Medicine. There are four faculties, Business, Engineering, Medicine and Natural Sciences, with the faculties of Engineering, and Natural Sciences having nine and four departments respectively. The total number of FTEs is approximately 15,000, of which approximately 8,500 are Under Graduates, 2,500 taught Post Graduates and 3,000 research-based Post Graduates.

Of the above, approximately 8,500 are home students, 2,000 are EU with the remainder being overseas students.

Within Imperial College London a recent decision to move towards automated timetabling was driven by the need to reduce capital spend and the ability to effectively utilise the College's estate. In order to minimize the need to build new or refurbish existing teaching space, and to maximize potential for revenue generation the ability to identify empty space and capitalize on opportunities for rental to external bodies was also considered advantageous. To meet these objectives, it was considered necessary to assess the possibility of centrally controlled and owned shared space. The aims of the project are:

- (i) To ensure that timetabling and strategic modelling software is 'fit for purpose' for the next decade and beyond.
- (ii) It was also considered that in order to effectively measure the ability to increase recruitment to existing courses and bring on-line new offers, the development and deployment of a strategic modelling environment needed to be assessed.
- (iii) To improve student experience and reflect current 'real-time' timetables via web technologies, allowing students instant access to course and examination schedules and be informed of changes to timetable.
- (iv) To reduce department overhead by locally engaging in support functions that could be more effectively resourced centrally.

Currently within Imperial College all space is 'owned' and scheduled by individual departments. This is detrimental to the College as it allows for little opportunity for collaborative courses, as departmental timescales are out of sync with each other. Additionally, the College has very poor utilisation rates, with a 'block booking' culture being prevalent. Departments have developed a very ad-hoc arrangement for sharing space, which is primarily done by directly contacting other departments and requesting usage.

Within the current project a methodology of centralising the collection of data has been developed, and a College-wide timescale adopted. The methodology employed was firstly to compile a complete set of building blocks that provide the foundation of the timetable i.e. Modules offered (compulsory, options), Delivery mechanism (lectures, seminars, tutorials), Staff members, Student groups, Room data, Cohort sizes and Constraints. Once the building blocks are in place, an attempt to auto-schedule events would be executed, excluding rooming.

- (i) Staff and student clashes would be explored, and resolved.
- (ii) Events would then be de-scheduled, with all constraints attached to individual events.
- (iii) Large lecture space would be scheduled, and departmental zoning applied.
- (iv) Specific space requests would then be processed, and any clashes resolved.
- (v) Finally small teaching space and laboratories would then be allocated.

As an interim measure for the first year of the project, a decision at board level to allow departments to specify days and times of teaching events was taken. Post the September implementation, these time constraints are to be removed and events rescheduled. At this stage pre and post utilisation figures will be analysed, and used as a benchmark for distance travelled within the timetabling project. With the goal of sharing space, The College are looking to centrally allocate all teaching space, to model and reconfigure timetables in order to achieve maximum utilisation, and to produced an informed capital investment programme dependent on the results of potential models.

This talk will discuss the space strategies and show simulations on how their implementation has the ability to make better use of resource. An update will be given on the implementation and examples provided of best practice when implementing space sharing strategies as part of a decentralised timetabling system.

- McCollum (2008), B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Di Gaspero, R. Qu, E.K. Burke, Setting The Research Agenda in Automated Timetabling: The Second International Timetabling Competition,
- McCollum (2007a), B. McCollum, A Perspective on Bridging the Gap between Theory and Practice in University Timetabling, Practice and Theory of Automated Timetabling VI, Springer LNCS Vol 3867, 2007, pp 3-23.
- McCollum (2007b), B. McCollum, T. Roche, P. McMullan, Optimising Space Through Macro and Micro Planning and Scheduling, Presentation at SCUP-42 , Society of College and University Planners International Conference, July 7-11, 2007, Chicago.

---

# A Hybrid Evolutionary Algorithm for the Generalized Surgery Scheduling Problem

Atle Riise • Edmund Burke • Carlo Mannino

## 1 Introduction

The term surgery scheduling is used about a variety of strategic, tactical and operational scheduling problems [1], many of which are critical to an efficient use of hospital resources. Our focus is on operational surgery scheduling, which may be informally described as the task of assigning times to surgery-related activities for each patient, while reserving capacity for these activities on a set of constrained renewable resources. Such resources may be, for example, operating rooms, operation teams, surgeons, equipment, or post-operative bed capacity. Objectives are typically overtime, hospitalization costs, intervention costs, operating room utilization, patient's waiting time, and patient or personnel preferences, among others. These scheduling problems are often NP-hard [2]. The exact problem formulation varies substantially between hospitals, or even hospital departments. In addition, the degree of detail vary between different planning situations; patient admission planning may consider only one or two kinds of resources, is mainly concerned with allocating a date of admission for each patient, and typically has a long time horizon. Closer to the day of surgery, such as when scheduling surgeries for the next day or week, the number of activities, resources and choices to make increase. This diversity presents a challenge for those who wish to create scheduling methods that are applicable to surgery scheduling problems in general.

## 2 The Generalized Surgery Scheduling Problem

In [3], we approached this challenge by generalisation, introducing the "Generalized Surgery Scheduling Problem" (GSSP). The GSSP can be seen as a rich extension to the Resource Constrained Project Scheduling problem (RCPS) [4]. It has multiple projects (one per patient), multiple resources per project activity, multiple modes, and setup times. The GSSP also has time dependent resource capacity, block constraints, and maximum delay constraints. Furthermore, it contains some new constraints: The "mode compatibility constraint" limits the simultaneous choice of modes for sets of related activities. For example, if a surgery activity uses a certain operating room, other related activities must also use that same room. Another constraint is the "mode dependent precedence constraint", which means that depending on the chosen modes for two activities, there may or may not be a precedence constraint between them. Finally, the "project disjunction constraint" dictates that for some resources, all activities related to a given project must be performed before the resource can be used for any activity of any other project, even if the resource has available capacity. This comes from the fact that one wish to complete all tasks relating to one patient in the operating room before starting any tasks relating to another patient.

The problem is naturally modelled as a directed activity-on-node graph. This problem graph can be seen as a union of project graphs and resource graphs. Project graphs represent precedence-, time window-, and maximum delay constraints. Resource graphs contain a sequence of resource periods [3], each represented by a pair of artificial start and end nodes. A range of constraints and objectives can be calculated directly from the propagated earliest start time of the end nodes of these periods. Resource period capacities and activity demands are modelled by constraints on a flow of resource units in the resource graphs.

A solution can be constructed by inserting each activity into the resource graphs of the activity's selected mode. Such a solution is feasible with respect to time-related constraints if the resulting solution graph is positively acyclic.

### 3 Algorithmic approach

Perhaps reflecting the variety of real world surgery scheduling problems, the literature includes the use of a wide range of resolution methods, both exact and heuristic [5]. Many variants of the problem are NP-hard, and several authors conclude that a meta-heuristic approach is needed for problems of realistic size. This is also our conclusion for the GSPS [3]. In this paper, we present two meta-heuristic methods for the GSSP. As for many studies of the RCPSP, we use a random key list solution representation. A schedule generation scheme (SGS) is used to produce a schedule by inserting activities into the solution graph in the order given in such a list. Our SGS is a heavily modified version of the classical sequential SGS [6]. It handles maximum delay constraints, project disjunctions and resource periods. Furthermore, this SGS can perform a search in the possible modes for each activity that is inserted, taking mode consistency and mode-dependent precedence constraints into account. As a baseline meta-heuristic we create a random restart algorithm which uses the SGS to construct schedules based on randomly selected random key lists. We then go on to present an evolutionary algorithm whose combination operators work on the level of mode choices and random key representations. Child schedules are constructed by applying the SGS. Using realistic test data from three different planning situations (“admission planning”, “weekly surgery scheduling”, and “daily surgery scheduling”) [3], we demonstrate that this algorithm performs better than the base-line algorithm. We also show that it produces good approximations to the optimal solutions, using computation times that are acceptable in real life planning situations.

**Acknowledgements** This work is supported by the Research Council of Norway, through the HOSPITAL project.

### References

1. Blake, J.T. and M.W. Carter, *A goal programming approach to strategic resource allocation in acute care hospitals*. European Journal of Operational Research, 2002. **140**(3): p. 541-561.
2. Hans, E., et al., *Robust surgery loading*. European Journal of Operational Research, 2008. **185**(3): p. 1038-1050.
3. Riise, A. and C. Mannino, *The Surgery Scheduling Problem - A General Model*, in *SINTEF rapport*, SINTEF, Editor. 2012, SINTEF: Oslo.
4. Cardoen, B., E. Demeulemeester, and J. Beliën, *Operating room planning and scheduling: A literature review*. European Journal of Operational Research, 2010. **201**(3): p. 921-932.
5. Riise, A. and E. Burke, *Local search for the surgery admission planning problem*. Journal of Heuristics, 2010: p. 1-26.
6. Artigues, C., S. Demasse, and E. Néron, eds. *Resource-constrained Project Scheduling: Models, Algorithms, Extensions and Applications* Control Systems, Robotics and Manufacturing. 2008, ISTE: London, UK.



---

# An exact decomposition approach for the optimal real-time train rescheduling problem

Carlo Mannino <sup>\*</sup>      Leonardo Lamorgese <sup>†</sup>

Trains are running through a rail network trying to meet a predefined schedule, the *Official Timetable*, which specifies when each train enters and exists the stations on its route. When one or more trains deviate from the official timetable, new schedules and possibly new routes must be identified and implemented very quickly. Also, the new plan should minimize some measure of the delays.

In a first and very simplified picture, a rail network may be viewed as a set of stations connected by tracks. Each train follows a specific route in this network, namely an alternating sequence of stations and tracks. The trains run their routes trying to agree with the *production plan*, which specifies the movements (*routing*) and the times when a train should enter and leave the various segments of its route (*schedule*), including stations arrival and departure times.

In principle, the production plan ensures that no two trains will occupy simultaneously the same railway resource, or incompatible resources such as a platform in a station and the track to access it. In other words, a production plan is a *conflict free* schedule. The problem to design optimal production plans is of crucial relevance for railway operators. As pointed out in [11] *optimum resource allocation can make a difference between profit and loss for a railway transport company*. However, due to different causes the actual train timetables can deviate from the official ones, and potential conflicts in the use of resources may arise. As a consequence, re-routing and re-scheduling decisions must be taken in real-time. These decisions are still, in most cases, taken by human operators (*dispatchers*), and implemented by re-orienting switches and by controlling the signals status (i.e. setting signalling lights to green or to red), or even by telephone connections with the drivers. The dispatchers take their decisions trying to minimize delays, typically having in mind some ranking of the trains or simply following operating rules. So, what the dispatchers are actually doing, is solving an optimization problem (and of a very tough nature). We call this problem the *Real-time Traffic Control in Rail Systems* problem (*RTC*).

In short, the RTC problem amounts to establish *in real-time* for each controlled train a route and a schedule so that no conflicts occur with other trains and some

---

<sup>\*</sup>SINTEF ICT, Oslo, e-mail: carlo.mannino@sintef.no

<sup>†</sup>SINTEF ICT, Oslo, e-mail: leonardo.lamorgese@gmail.com

function of the deviation from the official timetable is minimized. As such, the RTC problem falls into the class of *job-shop scheduling problems* where trains correspond to jobs and the occupation of a railway resource by a train is an operation. Two alternative classes of formulations have been extensively studied in the literature for job-shop scheduling problems and consequently also applied to train scheduling and routing problems, namely the *time indexed formulations* [10] and the *disjunctive formulations* [2].

In time indexed formulations (TI) the time horizon is discretized, and a binary variable is associated with every operation and every period in the time horizon. Conflicts between operations are prevented by simple packing constraints. Examples of applications of (TI) to train optimization can be found in [3], [4], [5], [6], [18]: actually the literature is much wider, and we refer to [8], [11] and [16] for extensive surveys. To our knowledge, basically all these works deal with the track allocation problem, which is solved off-line and where the feasible time periods associated with train routes are strongly limited by the tentative timetable. In contrast, in the RTC problem the actual arrival and departure times may differ substantially from the wanted ones. Consequently, the number of feasible time periods grows too large to be handled effectively by time-indexed formulation within the stringent times imposed by application, as extensively discussed in [13].

In disjunctive formulations, continuous variables are associated with the starting times of the operations, whereas a conflict is represented by a disjunctive precedence constraints, namely, a pair of standard precedence constraints at least one of which must be satisfied by any feasible schedule. The *disjunctive graph* ([1]), where disjunctions are represented by pairs of directed arcs, can be associated to any disjunctive formulation and exploited in solution algorithms. The disjunctive formulation associated can be easily transformed into a Mixed Integer Linear Program (MILP) by associating a binary variable with every pair of (potentially) conflicting operations and, for any such variables, a pair of *big-M precedence constraints* representing the original disjunction. These constraints contain a very large coefficient and they tend to weaken the overall formulation and this is mainly the reason why (TI) formulations were introduced.

The connection between railway traffic control problems, job-shop scheduling and corresponding disjunctive formulations was observed quite early in the literature. However, a systematic and comprehensive model able to capture all the relevant aspects of the RTC was described and studied only much later in the Ph.D. thesis by Alessandro Mascis [14] and further developed in [15]. In these works, the authors also introduce a generalization of the disjunctive graph that they call *alternative graph* but referred here simply as disjunctive graph. After these early works there has been a flourishing of papers representing the RTC by means of disjunctive formulations and exploiting the associated disjunctive graph. Recent examples can be found in [7], [8], [9], [17]. A comprehensive list of bibliographic references is out of our scope and again we refer to the above mentioned surveys. The great majority of these papers, however, only use the disjunctive formulation as a descriptive tool and resort to purely combinatorial

heuristics to solve the corresponding RTC problems. The explicit use of the disjunctive formulation to compute bounds is quite rare, and typically limited to small or simplified instances. Examples are [13], which handles small-scale metro instances, and [17], which introduces several major simplifications, drastically reducing the instances size.

So, methods based on mathematical programming are rarely applied to solve real-life instances of the RTC problem: time-indexed formulations tend to be too large and often cannot even generate a solution within the time limit; big-M formulations tend to be too weak and they can fail to produce feasible solutions within the time limit.

In this presentation we introduce a new modelling approach to RTC and a solution methodology which allow to overcome some of the limitations of the standard big-M formulations and solve to optimality the corresponding big-M MILP within the stringent running times imposed by the application for a number of real-life instances in single-track railways. The methodology is based on a structured decomposition of the RTC into two sub-problems: the *Line Traffic Control Problem* (LTC) and the *Station Traffic Control Problem* (STC). The LTC amounts to establishing where potentially conflicting trains should meet along the network. When dealing with single-track lines, this may only happen in stations (or similar infrastructures). The STC problem is the problem of routing and scheduling trains in a station (in real-time). The LTC problem and the STC problem give raise to distinct sets of variables and constraints, which are then solved in a joint model by row and column generation.

The decomposition has two major advantages. First, the number of variables and big-M constraints is drastically reduced with respect to the standard big-M formulations. The second advantage is that we have some degrees of freedom in modelling the STC problem. Indeed, we will show that the (general) STC problem is NP-hard. However, in some cases of practical impact, simpler models can be considered, leading to polynomial cases: we will describe one such case, very common in practice. Actually, since the lines may contain quite different stations' layouts, different models can/must be applied simultaneously.

Interestingly, this decomposition resembles the normal practice of railway engineers to distinguish between *station tracks* and *line tracks* (see, e.g. Conte 2007) and of actually tackle the two problems separately.

This decomposition approach has been successfully applied in a rescheduling system operating a number of single and double track lines in Italy [12].

## References

- [1] Balas, E., Machine sequencing via disjunctive graphs, *Operations Research* 17 (1969) pp. 941–957.
- [2] Balas, E., *Disjunctive programming*, *Annals of Discrete Mathematics*, 5, pp. 3–51, 1979.

- [3] Borndörfer R., T. Schlechte, *Models for Railway Track Allocation*, ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Eds. Christian Liebchen and Ravindra K. Ahuja and Juan A. Mesa.
- [4] Brännlund, U., P.O. Lindberg, A. Nou, J.-E Nilsson, *Railway Timetabling using Lagrangian Relaxation*, Transportation Science, Vol 32 (4), pp. 358-369, 1998.
- [5] Caprara, A., M. Fischetti, P. Toth, *Modeling and solving the train timetabling problem*, Operations Research, 50 (5) 292, pp. 851-861, 2002.
- [6] A. Caprara, L. Galli, P. Toth. *Solution of the Train Platforming Problem*, Transportation Science, 45 (2), pp 246-257, 2011.
- [7] Conte C, *Identifying dependencies among dealys*, Ph. D. Thesis, University of Göttingen, 2007.
- [8] Corman F., *Rail-time railway traffic management: dispatching in complex, large and busy railway networks*, Ph.D. Thesis, TRAIL Thesis Series T2010/14, the Netherlands TRAIL Research School.
- [9] F. Corman, A. D'Ariano, D. Pacciarelli, M. Pranzo *A tabu search algorithm for rerouting trains during rail operations*, Transportation Research Part B 44 (2010) 175-192
- [10] Dyer., M., L. Wolsey, *Formulating the single machine sequencing problem with release dates as a mixed integer program*, Discrete Applied Mathematics, no. 26 (2-3), pp. 255-270, 1990.
- [11] Luthi M., *Improving the Efficiency of Heavily Used Railway Networks through Integrated Real-Time Rescheduling*, Ph. D. Thesis, ETH Zurich, 2009.
- [12] C. Mannino, *Real-time traffic control in railway systems*, Proceedings of Atmos'11, A. Caprara and S. Kontogiannis (Eds.), OASICS Vol. 20, 2011
- [13] C. Mannino, A. Mascis, *Real-time Traffic Control in Metro Stations*, *Operations Research*, 57 (4), pp 1026-1039, 2009
- [14] Mascis A., *Optimization and simulation models applied to railway traffic*. Ph.D. thesis, University of Rome "La Sapienza", Italy, 1997. (In Italian).
- [15] Mascis A., D. Pacciarelli, *Job shop scheduling with blocking and no-wait constraints*, European Journal of Operational Research, 143 (3), pp. 498-517, 2002.
- [16] J. Törnquist, *Railway Traffic Disturbance Management*, Ph.D. Thesis, Blekinge Institute of Technology, 2006

- [17] J. Törnquist, J. A. Persson, *N-tracked railway traffic re-scheduling during disturbances*, Transportation Research Part B 41, 342–362, 2007.
- [18] Zwaneveld, P.J., L.G.S. Kroon, H.E. Romeijn, M. Salomon, S. Dauzere-Peres, S.P.M. Van Hoesel, H.W. Ambergen, *Routing trains through railway stations: model formulation and algorithms*, Transportation Science, 30 (3), pp. 181-194, 1996.

---

## A real world personnel rostering problem with complex objectives

Oddvar Kloster, SINTEF Applied Mathematics, Oslo, Norway

We consider a practical personnel rostering problem faced by a public transportation company. The problem has some interesting objectives that we are not aware of having been treated before.

The problem consists of assigning shifts to a group of drivers over a time period. Each driver is to be assigned one shift on each day in the period. Except for shifts that represent a day off, the available shifts are unique; each can be assigned only on one specific day, and not to more than one driver. A number of day-off shifts are pre-assigned in the plan, yielding a fixed free day pattern for each driver. Shifts for the initial part of the period are also pre-assigned, providing a historical plan that must be sensibly continued.

Each shift has a start time, an end time and two durations: the work time and the driving time. Driver can express preferences as to which shifts they like or not, yielding a positive or negative preference value for each shift /driver pair.

The chief hard constraints are:

- Driver/ shift compatibility
- The accumulated work time for a driver may at no time deviate more than 40h from the nominal value

The objectives considered are:

- Maximize the preference values of the shift /driver assignments, fairly among the drivers
- Penalize large deviations of the accumulated work times from the nominal values at the end of the period
- Minimize the number of days off assigned in excess of the free day patterns
- Avoid many similar (by start and end time) consecutive shifts
- Avoid violations of the EU work and driving time regulations, plus rules set by local agreements

We solve the problem using local search. The following operators are used to modify the solution:

- Replace the shift assigned to a cell (driver/day pair) with an unassigned shift
- Swap the shift assigned on one day between two drivers
- Swap the shifts assigned on two distinct days between two drivers
- Swap the shifts assigned on a range of three or more consecutive days between two drivers

To establish an initial solution, we start with an empty solution, where all assignments not initially fixed are set to a day off. We make a few passes of applying the Replace operator in each cell, only accepting improving changes. We then move to the main search phase.

The main search is a focused tabu search. Each objective is responsible for identifying focal points, i.e. areas where the incumbent solution could be improved, and estimate the possible associated gain in objective value. In each iteration, the focal point with the highest estimated gain is selected. A neighborhood is generated by restricting the

full set of solutions defined by the above operators, in two ways. Firstly, the solution must be modified in a region given by the selected focal point (e.g. a particular driver or day range). Secondly, the objective that generated the focal point must be improved locally in that region. This eliminates many solutions that need not be evaluated by all constraints and objectives. The best solution in the neighborhood is chosen as the new incumbent solution. A tabu criterion based on the recently modified cells is used to further reduce the neighborhood and to avoid cycling.

The major motivation for solving this problem is to increase driver satisfaction by allowing them to express their duty preferences and satisfying those preferences. Thus modeling the preference satisfaction objective correctly is vital to the implementation's success, and we present it in more detail.

There are several requirements for this objective:

- The basic effect is to maximize each driver's total preference for his assigned shifts
- If not all drivers can get their preferred shifts, the good assignments shall be distributed fairly among the drivers
- A driver should benefit from being flexible, that is, giving preferences that are easy to satisfy, as this also helps satisfying other drivers' preferences
- A driver should not be able to play the system, that is, benefit from expressing preferences that are not real

We propose the following objective formulation:

For each driver and day, define the driver's *luck* with the day's shift assignment as  $l = P - E(P)$ .  $P$  is the actual assignment's preference value, and  $E(P) = \frac{1}{n} \sum_{i=1}^n p_i$  is the expected preference value, assuming that each compatible shift for that day is equally probable.

Let  $L_j$  be the sum of daily luck values over the period for driver  $j$ . Then we wish to maximize each value in  $\{L_j\}$ , while also minimizing inequality among the values.

Let  $a_1 \dots a_m$  be a non-decreasing sequence of positive numbers with  $a_1 < a_m$ , and let  $K_1 \dots K_m$  be the values in  $\{L_j\}$  sorted from high to low. The total preference to be maximized is defined as  $V = \sum_{j=1}^m a_j K_j$ .

We argue that this objective definition satisfies the requirements above.

Another objective of interest penalizes violations of the EU work and driving time regulations. This objective is expensive to evaluate due to the rules' complexity. In each week, a driver's plan must contain a weekly rest period (from one shift's end time to the start time of the next) of sufficient length. Several aspects of the rules make the choices of rest period in different weeks interdependent:

- There is a limit on the maximum time between rest periods in consecutive weeks
- A rest that spans the border between two weeks can be assigned to either week, but not both
- A rest can be shorter than the normal length, but only if the next rest has at least normal length, and providing that the reduction is compensated over the next three weeks

Testing whether a solution has a legal weekly rest assignment requires us to perform a costly tree search. This leads us to putting this objective among the ones to be evaluated last.

We present results on real life problem instances covering up to 46 drivers over an 8 week period. User reports on the quality of the solutions produced during pilot tests are favorable.



# Demonstrations

---

# An Intelligent, Interactive & Efficient Exam Scheduling System (IIEESS v1.0)

**Zhu Chunbao and Tha Nu**

School of Engineering, Nanyang Polytechnic, Singapore

Tel: (65) 65501808, Fax: (65) 6454 9871

Email: ZHU\_Chun\_Bao@nyp.gov.sg

**Abstract** The purpose of this paper is to introduce and demonstrate an exam scheduling software system that performs efficient, accurate and robust solution searching to solve large and complex exam scheduling problems. The paper describes the main features of the system and in particular the test paper conflictive analysis method that can provide a highly efficient data model to significantly improve search efficiency and interactivity.

**Keywords** Exam timetable scheduling, test paper conflictive analysis, swarm intelligence, indirect clash-checking

## 1 Introduction

The IIEESS v1.0 is an intelligent, interactive & efficient exam scheduling system designed and developed to solve large and complex exam scheduling problems using the patent pending technologies (Zhu Chunbao, 2008, 2010). The software solution can be applied to schools, institutions, universities and training centers which need to schedule examination activities and allocate venue resources to facilitate these activities.

Traditional exam scheduling systems directly examine the vast amount of student registration data for checking student conflicts and constraint violations repeatedly during solution searching cycles. This is not efficient and not robust particularly when iteration based search algorithms are used, such as GA and ACO based systems (Shu-Chuan Chu, Yi-Tin Chen, Jiun-Huei Ho, 2006). As results, computer runtime is lengthy (Nelishia Pillay and Wolfgang Banzhaf, 2007), such as for example, it takes 4 ~ 5 hours for the program to search for a solution.

Unlike direct clash checking, the IIEESS v.10 system firstly carries out the test paper conflictive analysis, which yields a conflictive coefficient matrix  $\Omega_{n \times n}$  and then further creates the mutually exclusive paper lists  $MEL_k$  ( $k=0$  to  $n$ ,  $n$  is the

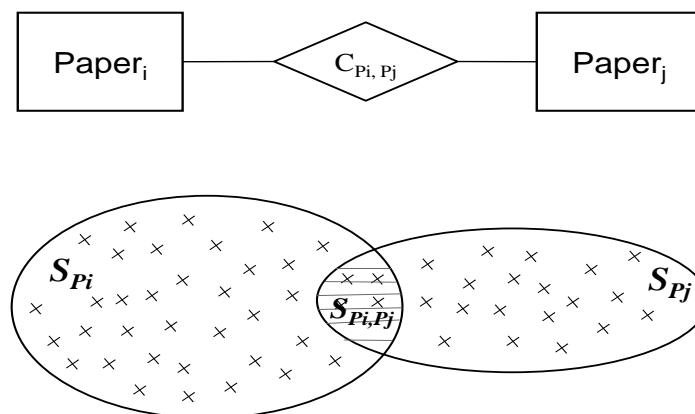
number of exam papers). Note that the number of elements in  $MEL_k$ , denoted  $N_{MEL_k}$  is much less  $n$ . The system then indirectly examines the conflictive coefficients in  $MEL_k$  for student conflicts in solution searching cycles, rather than directly examining the huge amount student registration records and original constraints imposed. The system also utilizes the conflictive coefficients to minimize constraint violations to further increase the system's efficiency and accuracy.

Because the number of exam papers  $n$  (say hundreds) and  $N_{MEL_k}$  (say tens) is much less than the number of student registration records (tens thousands to millions), the new system enjoys high efficiency, accuracy and robustness. Our computational experiments show that the IIEESS v1.0 system is much faster than direct-clash-checking systems. The high efficiency enables the new system not only to provide fast auto-searching, but also to facilitate the system with user-friendly and truly effective drag-&-drop features which are critically important for planners to perform manual amendments to the auto-generated schedule.

The system provides a powerful automatic venue resource allocation engine and user-friendly drag-&-drop features as well for facilitating the scheduled examinations.

## 2 The Method of Test Paper Conflictive Analysis

To simplify the explanation of the paper conflictive analysis, Fig. 1 utilizes two test papers, Paper<sub>*i*</sub> and Paper<sub>*j*</sub> for illustration purpose. The set of students, who take Paper<sub>*i*</sub> and Paper<sub>*j*</sub> denoted  $S_{P_i}$  and  $S_{P_j}$  respectively. The intersection of the set  $S_{P_i}$  and the set  $S_{P_j}$ , is denoted as  $\Delta S_{P_i, P_j}$ .



**Fig. 1** Test paper conflictive coefficient

It is important to note that the candidates, if any, in the intersection of the Set  $S_{P_i}$  and Set  $S_{P_j}$  are *common students* who take both Paper<sub>*i*</sub> and Paper<sub>*j*</sub>. If  $\Delta S_{P_i, P_j}$  is not empty, i.e., the number of students in the intersection set  $\Delta S_{P_i, P_j}$  is large than zero, we conclude that Paper<sub>*i*</sub> conflicts with Paper<sub>*j*</sub>. By conflicts, we mean that the Paper<sub>*i*</sub> and Paper<sub>*j*</sub> cannot be scheduled at same period of time because they have at

least one common student. In other words, Paper<sub>i</sub> and Paper<sub>j</sub> are mutually exclusive each other.

#### *Indirect Constraint Evaluation Method*

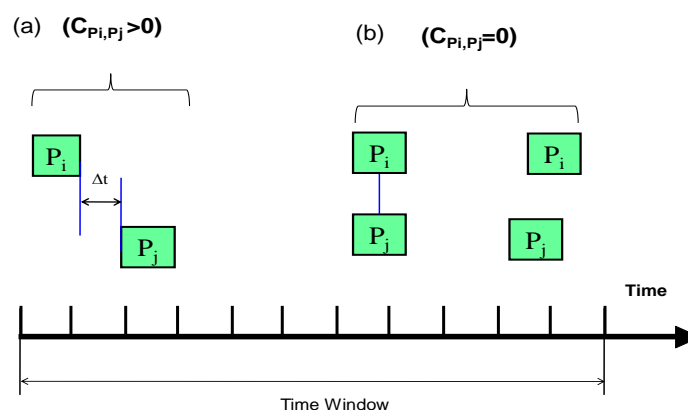
In scheduling, it is convenient for the search engine (program) to examine the original constraints imposed to avoid constraint violation, which is “direct-constraint-checking”. Indirect constraint evaluation method performs the task in two separate steps. First step is to study the event conflictive features among all events according to the original constraints imposed. Output of the first step is a short checking list. The second step is to check the short checking list rather checking the original constraints imposed to avoid constraint violation in solution searching cycles.

Note that the first step is one time operation before solution searching; the second step is to be repeated in solution searching cycles. Because generally for large and complex ETPs, the short checking list is much shorter than the original constraints imposed, therefore the indirect checking method enjoys much higher efficiency, accuracy and robustness. In this paper, the short list used to check student conflicts is constructed using the test paper conflictive coefficients which will be described as follows.

#### *The test paper conflictive coefficient*

It is convenient to use a single numerical number (integer) to describe the conflictive relation among test papers. We utilize the paper conflictive coefficient,  $C_{P_i, P_j}$  to measure how two test papers are conflictive each other, where the indexes  $P_i$  and  $P_j$  refer to any two test papers in the schedule. For example,  $C_{P_i, P_j}$  is the paper conflictive coefficient for Paper<sub>i</sub> and Paper<sub>j</sub> as shown in Fig. 1. In general, for every two papers, Paper<sub>i</sub> and Paper<sub>j</sub>, the paper conflictive coefficient,  $C_{P_i, P_j}$  can be obtained as follows:  $C_{P_i, P_j} = |\Delta S_{P_i P_j}|$ . Where  $S_{P_i P_j}$  is the intersection of the student set for Paper<sub>i</sub> and Paper<sub>j</sub>; and  $|\Delta S_{P_i P_j}|$  denotes the cardinality of the intersection set  $\Delta S_{P_i P_j}$ .

Fig. 2 shows that the exam events can be more efficiently scheduled using test paper conflictive coefficients.



**Fig. 2** Exam event scheduling using test paper conflictive coefficients

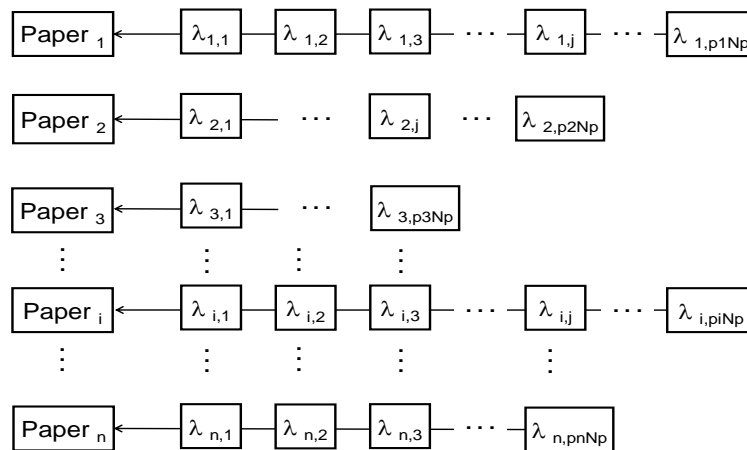
For example, if  $C_{P_i,P_j}$  is equal to zero, Paper<sub>i</sub> and Paper<sub>j</sub> are independent each other; which means that they can be scheduled at same time slots or at a different time slot but with overlapped period, as can be seen in Fig. 2 (b); otherwise Paper<sub>i</sub> and Paper<sub>j</sub> are mutually exclusive, which means that they cannot be scheduled at same time, there must be a time gap ( $\Delta t > 0$ ) between the two exams, shown in Fig. 2 (a).

The quantitative value of the paper conflictive coefficient,  $C_{P_i,P_j}$ , is important for the system to evaluate soft constraint violations. For example, if two mutually exclusive exam papers, such as Paper<sub>i</sub> and Paper<sub>j</sub>, are scheduled with a narrow time gap ( $\Delta t$ ), which will result in B2B constraint violation, or “Multiple Exams A Day Conflicts” - the multiple papers are scheduled on the same day, the system has to examine  $C_{P_i,P_j}$  which is the number of students involved. It is necessary to minimize the total number of students who are scheduled to do multiple papers within one day.

Fig. 1 and Fig. 2 show two test papers for illustration on how to use a paper conflictive coefficient to measure the conflictive grade. In practice, the number of test papers, denoted  $n$ , can be quite large. Therefore, it is necessary to express the conflictive relations among  $n$  papers, that is,  $P_1, P_2, \dots, P_{n-1}, P_n$ ; the matrix of the conflictive coefficients among  $n$  papers is introduced, its denotation is  $\Phi$ . Where  $\Phi$  is an  $n \times n$  matrix expressed as  $\Phi = [C_{i,j}]_{n \times n}$ . Where, the element  $C_{i,j}$  is the conflictive coefficient for Paper<sub>i</sub> and Paper<sub>j</sub>. Let  $C_{i,j} = 0$  if  $i=j$ ; because a paper can never be conflictive or mutually exclusive with the paper itself. It is noted that Matrix  $\Phi$  is symmetrical, that is, element  $C_{i,j} = C_{j,i}$  because conflictive nature between Paper<sub>i</sub> and Paper<sub>j</sub> is same as the one between Paper<sub>j</sub> and Paper<sub>i</sub>.

*The paper’s mutually exclusive paper lists*

Furthermore, we remove the elements whose value is zero from in Matrix  $\Phi$ , we can get a shorter conflictive coefficient list and then obtain a mutually exclusive paper list for every paper as shown in Fig. 3.



**Fig. 3** The mutually exclusive paper lists

For example, for Paper<sub>i</sub>, its mutually exclusive paper list is represented using a list of value pairs, each being denoted  $\lambda_{P_i, P_j}$  as  $\lambda_{P_i, P_j} = \langle P_j : C_{P_i, P_j} \rangle$ .

The element  $\lambda_{P_i, P_j}$  is called <paper index - conflictive coefficient> value pair. Fig. 3 shows the mutually exclusive paper lists in form of the value pairs,  $\lambda_{P_i, P_j}$ . Note the subscript,  $i = 1$  to  $n$ , where  $n$  is the number of total test papers;  $j = 1$  to  $p_i N_p$ , where  $p_i N_p$  is the total number of conflictive papers which are conflictive with Paper  $P_i$ . As can be seen from Fig. 3, the value of  $p_i N_p$  varies. In general it is much less than  $n$  in most of the cases. An exemplary value of the value pair  $\lambda_{P_{10}, P_{20}}$  is  $\langle P_{20}:400 \rangle$ , which means that paper  $P_{10}$  is conflictive with the paper  $P_{20}$ , and the number of students who take both paper  $P_{10}$  and paper  $P_{20}$  is 400.

As can be seen from Fig. 3, the value pair  $\lambda_{P_i, P_j}$  is used to construct the mutually exclusive paper lists. For paper  $P_i$ , its mutually exclusive paper list  $MEL_i$  is expressed as follows.

$$MEL_i = \{ \lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3}, \dots, \lambda_{i,j} \dots \lambda_{i, P_i N_p - 1}, \lambda_{i, P_i N_p} \} \quad (1)$$

It is the mutually exclusive paper list  $MEL_i$  that is used in the IIEES 1.0 system for clash-checking. That is, if  $T(\text{Paper}_i)$  denotes the time slot scheduled for Paper<sub>i</sub>,  $T(\text{Paper}_j)$  denotes the time slot scheduled for Paper<sub>j</sub>, for any Paper<sub>j</sub> which is in Paper<sub>i</sub>'s mutually exclusive paper list  $MEL_i$ , following student conflict free constraint must be satisfied.

$$T(\text{Paper}_i) \neq T(\text{Paper}_j) \quad (2)$$

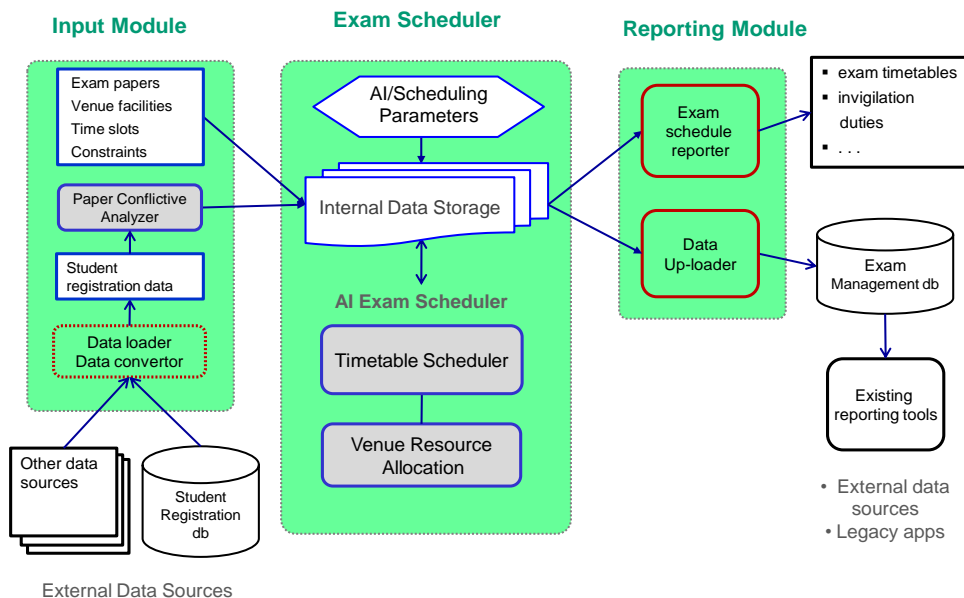
If the number of students taking paper  $P_i$ , is  $P_i N_p$ , say 500 students, that is, who were enrolled with Paper<sub>i</sub> as shown in Fig. 1, and the number of students taking Paper<sub>j</sub> is 400, traditional direct clash checking method has to make massive comparisons  $500 \times 400 = 200,000$  in order to find if there is any student conflict. However, using the new indirect clash checking method, the system only needs to check whether the Paper<sub>j</sub> is in Paper<sub>i</sub>'s mutually exclusive paper list  $MEL_i$ , if answer is yes, Paper<sub>j</sub> and Paper<sub>i</sub> cannot be scheduled at same time due to student conflicts.

It should be highlighted that to check student conflicts using the indirect clash checking method, the number of comparisons is  $P_i N_p$ , which is the length of Paper<sub>i</sub>'s mutually exclusive paper list  $MEL_i$ . Typical value of  $P_i N_p$  for a medium sized exam scheduling problem, is tens, say 0 to 20, which is much less than  $P_i N_p \times P_j N_p$ , say 200,000 as described previously. As the result, the new indirect constraint checking method is many times (i.e., 10,000) faster than traditional direct constraint checking for the example illustrated in Fig. 1.

### 3 The System Architecture and Software Modules

The IIEESS 1.0 system consists of three functional modules: 1.) input module, 2.) exam scheduler, and 3.) reporting module, as shown in Fig. 4. The input module consists of a data loader that down-loads student registration data from external sources such as databases or other forms of data storage. The input module also stores into the internal storage the exam scheduling information such as exam papers, venue facilities, and time slots as well as constraint information.

Once registration data are loaded into the system, the test paper conflictive analyzer will performs data pre-processing, test paper conflictive analysis in particular, and store the paper conflictive information for the exam scheduler to use. The scheduler contains an internal storage, a timetable scheduler and venue resource allocation module. The reporting module provides functions to upload the exam schedule solution to legacy database such as exam management system, and it can also generate various exam timetable reports for trail release or formal publication.



**Fig. 4** IIEESS 1.0 System Architecture

### 4 Input Data

The input data to the IIEESS v1.0 system are categorized as follows: 1.) student registration data, and 2.) exam scheduling information. The student registration data describe “who studies what and in which group?”, i.e., the candidates-papers relationships, which is critically important because the exam scheduler has to generate a conflict-free timetable solution. The exam scheduling information includes the following:

- 1.) Exam paper information and related constraints
- 2.) Venue facilities and capacity/availability constraints
- 3.) Time slot specification
- 4.) Soft constraints

### ***Student registration data***

The student registration data can be presented and stored in many different forms. The format adopted by the IIESS 1.0 is as follows.

<Student Admin No>   <Module Code>   <Module Group>

The school name and campus code in Fig. 5 are specific in our school and for venue resource allocation use. Note that the student admin number, module code and module group must be unique in the schedule.

**Who Studies What, in Which Group?**

Student <u>Adm. No</u>	Module Code	Module Group
062158U	BM0055	BM0055-FC

SCH	CAMPUS_CODE	ADM_NO	MODULE_CODE	MODULE_GRP
SCH	AMK	046856W	BM6386	BM6386-01
SBM	AMK	046856W	BM6386	BM6386-01
SBM	AMK	062158U	BM0055	BM0055-FC
SBM	AMK	066050J	BM0055	BM0055-BH
SBM	AMK	066050J	BM0902	BM0902-BH
SBM	AMK	066050J	BM0903	BM0903-BH
SBM	AMK	066849B	BM6382	BM6382-01
SBM	AMK	066849B	BM6386	BM6386-01
SBM	AMK	066850U	BM6382	BM6382-01
SBM	AMK	066850U	BM6386	BM6386-01
SBM	AMK	066851G	BM6382	BM6382-01
SBM	AMK	066851G	BM6386	BM6386-01
SBM	AMK	066853F	BM6382	BM6382-01
SBM	AMK	066853F	BM6386	BM6386-01
SBM	AMK	066854W	BM6382	BM6382-01
SBM	AMK	066854W	BM6386	BM6386-01
SBM	AMK	066855J	BM6382	BM6382-01
SBM	AMK	066855J	BM6386	BM6386-01

SCH   Campus Code

\* Note that student adm. No, module code and group code must be **unique**.

**Fig. 5** Student Registration Data

### ***Time Slot Specification***

The planning period must be specified before the scheduling. Typical time slot specification is shown in Fig. 6. Note that in the exam scheduling, time is represented in form of integer numbers (namely slots). The numerical numbers are mapped back into real time for reporting purpose after schedule is complete.



Time Slots					(for Ref into only)
Day	Date	Week Day	Slot in Day	Time	Slot S/N (1 to N)
1	7-Feb-11	Mon	1	1000	1
			2	1500	2
2	8-Feb-11	Tue	1	1000	3
			2	1500	4
3	9-Feb-11	Wed	1	1000	5
			2	1500	6
4	10-Feb-11	Thu	1	1000	7
			2	1500	8
5	11-Feb-11	Fri	1	1000	9
			2	1500	10
6	14-Feb-11	Mon	1	0900	11
			2	1230	12
			3	1600	13
7	15-Feb-11	Tue	1	0900	14
			2	1230	15
			3	1600	16
8	16-Feb-11	Wed	1	0900	17
			2	1230	18
			3	1600	19
9	17-Feb-11	Thu	1	0900	20

Fig. 6 Time Slot Specification

**Test Paper Information**

The format of the test paper information in the IIEESS 1.0 system is shown in Fig. 7, which includes school code, paper ID; paper title; duration; and list of modules covered in the exam paper. The No. of students will be auto-counted by the system according to the student registration data set. Note that paper IDs must be unique within the whole schedule.

		Paper ID	Module List				
		EG309	EGB309, EG3244				
S/N	School	Paper ID	Paper Title	No of Students	Dur (hrs)	Note	list of modules (codes) covered by the test/exam paper
205	SEG-EGM	EG8301	EG8301	65	2		EG8301
206	SEG-EGM	EG8302	EG8302	64	2		EG8302
207	SEG-EGM	EG8307	EG8307	14	2		EG8307
208	SEG-EGM	EG309	EG8309	92	2		EG8309, EG3244
209	SEG-EGM	EG310	EG8310	139	2		EG8310, EG3245
210	SEG-EGM	EG8311	EG8311	19	2		EG8311
211	SEG-EGM	EG8314	EG8314	21	2		EG8314
212	SEG-EGM	EG8315	EG8315	21	2		EG8315
213	SEG-EGM	EG8317	EG8317	40	2		EG8317
214	SEG-EGM	EG8318	EG8318	19	2		EG8318
215	SEG-EGM	EG101	EGC101	737	2		EGC101, EGB101, EGD101, EGF101, CLC101, CLB101, CLG101, EGH101, EGJ101, EGK101
216	SEG-EGM	EM102	EGC102	303	2		EGC102, EGB102, EGD102, EGF102
217	SEG-EGM	EM103-a	EGC103-a	255	2		EGC103, EGB103, EGH103, EGJ103
218	SEG-EGM	EC107	EGC107	927	2		EGC107, EGB107, EGD107, EGF107, CLC107, CLB107, CLG107, EGH107, EGJ107, EGK107
219	SEG-EGM	EM108-a	EGC108-a	269	2		EGC108, EGB108, EGF108, EGH108, EGJ108
220	SEG-EGM	EM109-a	EGC109-a	218	2		EGC109, EGB109, EGF109

↑ Paper ID
 } List of modules tested in the paper

Fig. 7 Test Paper Information

**5 Test Paper Conflictive Analyzer**

After the student registration data and test paper information are loaded into the IIEESS 1.0 system, the system will perform test paper conflictive analysis which yields the mutually exclusive paper lists for every test paper as shown in Fig. 8.

For example, the paper titled “Materials Technology” with paper ID “EGC105”, has a mutually exclusive paper lists as follows.

$$MEL_{EGC105} = EGB205:1, EGF212:1$$

This states that the paper EGC105 is mutually exclusive with EGB205 and EGF212. There is one common student who takes both EGC105 and EGB205; another student taking both EGC105 and EGF212. The system will not schedule the papers EGC105, EGB205 and EGF212 at same time to avoid the student conflicts. The mutually exclusive paper lists are also used when the system optimizes the searched solutions by minimizing the total number of common students involved in back-to-back (B2B) and multiple-papers-a-day (MPD) conflicts.

S/N	School	Paper ID	Paper Title	Stds	Dur	Nmxc	List of mutually exclusive paper list and common students
1	SEG-EGM	EGC105	Materials Technology	302	1	2	EGB205:1 EGF212:1
2	SEG-EGM	EGC111	Computer Programming	355	1.5	10	EGC205:1 EGC211:1 EGC204:2 EGB205:1 EGC210:1
3	SEG-EGM	EGC205	Manufacturing Information System	162	1.5	5	EGC111:1 EGC204:89 EGC305:2 EGC307:1 EGS213:6
4	SEG-EGM	EGC211	Computer-Aided Manufacturing/Enginee	121	1.5	5	EGC111:1 EGC210:92 EGC307:1 EGF206:23 EGS213:48
5	SEG-EGM	EGB204	Micro-controller Applications	144	1.5	6	EGB205:95 EGH204:37 EGB210:1 EGB211:1 EGB306:2
6	SEG-EGM	EGC204	Metrology & Quality Control	117	1.5	6	EGC111:2 EGC205:89 EGC210:1 EGC305:1 EGC307:1
7	SEG-EGM	EGB205	Quality Assurance	105	1.5	6	EGC105:1 EGC111:1 EGB204:95 EGB210:1 EGB306:1
8	SEG-EGM	EGF210	Metrology & Quality Control	26	1.5	2	EGF212:26 EGS213:13
9	SEG-EGM	EGC210	Robotics System & Peripherals	102	1.5	4	EGC111:1 EGC211:92 EGC204:1 EGS213:47
10	SEG-EGM	EGC305	Manufacturing Systems & Simulation	42	1.5	3	EGC205:2 EGC204:1 EGC307:31
11	SEG-EGM	EGC307	Computer-Aided Design & Analysis	33	1.5	4	EGC205:1 EGC211:1 EGC204:1 EGC305:31
12	SEG-EGM	EGD301	Shopfloor Monitoring & Control	82	1.5	5	EGC321:7 EGD211:3 EGD212:1 EGD308:26 EGF212:1
13	SEG-EGM	EGC312	Advanced Metrology & Quality Assuranc	22	1.5	0	
14	SEG-EGM	EGC321	Advanced Machining Technology	26	1.5	1	EGD301:7
15	SEG-EGM	EGD211	Quality Process Control & Management	66	1.5	3	EGD301:3 EGD212:63 EGS213:26

**Fig. 8** List of mutually exclusive paper lists and conflictive coefficients

## 6 Exam Timetable Scheduler

The main GUI of the IIEESS v1.0 scheduler is shown in Fig. 9. The main features of the exam scheduler module are listed as follows.

- Load exam scheduling data and constraint information
- Auto-generate exam timetable solution
- Clash-checking and optimization support for manual operations
- Save searching results

Fig. 10 shows the drag-&-drop features provided for manual operations. Note that as the clash checking and constraint evaluation speed of the IIEESS 1.0, the system is generally many thousand times faster than the existing direct-clash checking systems, it can provide efficient and effective support for manual operations. Typically the system can confirm a manual alteration made to the schedule in few micro-seconds, whereas old systems can take several minutes (as long as 30 minutes) to confirm a change made to the schedule.

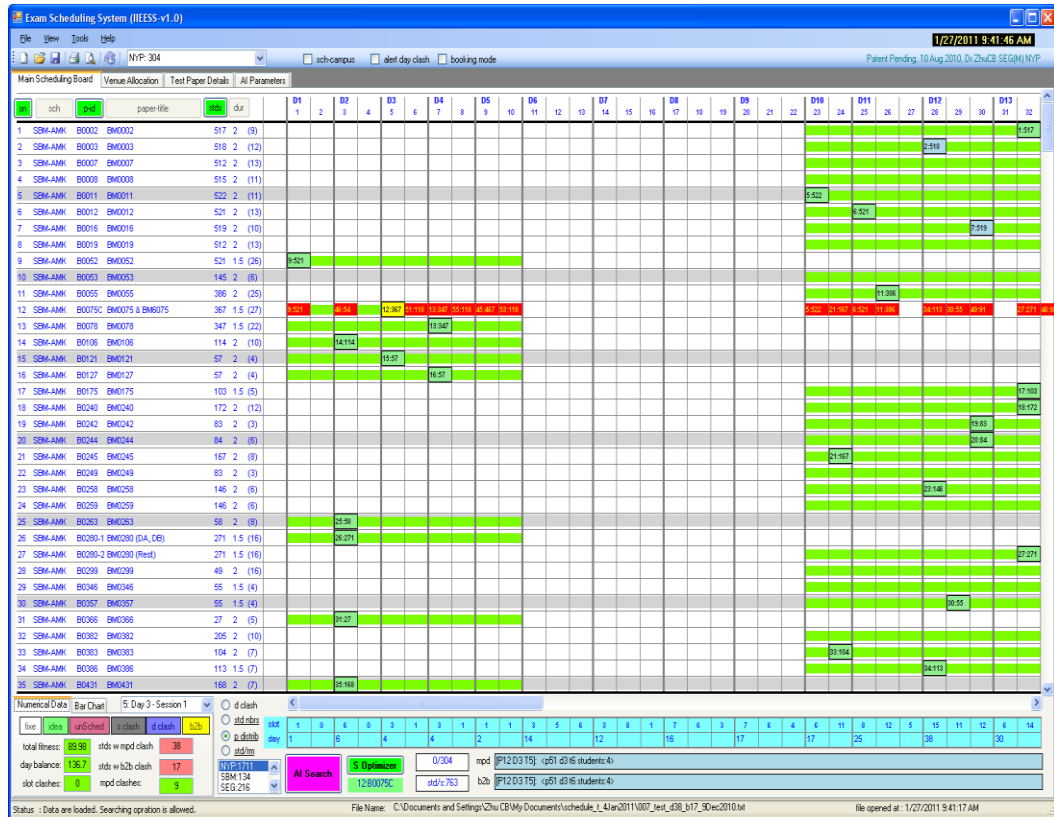
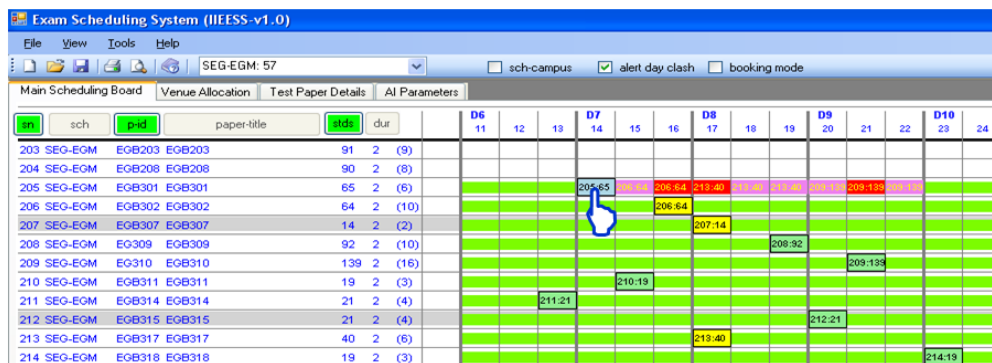


Fig. 9 IIESS v1.0 Main –Scheduler



When using drag-&-drop, clash-checking and constraint validation are auto-performed by the system:

- 1.) The slot in red color is not allowed to use because of student time clash.
- 2.) The slots in pink color can be used but not optimal because of mdp constraint violation conflict, i.e., students need to take 2 papers a day.

Fig. 10 Drag-and-drop features for manual operations on a schedule

## 7 Venue Resource Allocation Module

The un-facilitated exam activities which are scheduled using AI scheduler are then facilitated with venue resources by the venue allocation module (see Fig. 11). The module provides an auto searching mode and a manual drag-&-drop feature for manual venue allocation.

The fully facilitated exam schedule is saved a file as a final solution that is ready for reporting or exporting to the legacy database system.

The screenshot displays the 'Main Scheduling Board' with a 'Venue Allocation' tab. It shows a list of sessions (Day 7-9, Session 1-3) and a table of allocated venues. A green box at the top right shows file saving and loading information. A blue box highlights a table of allocated venues with three circled items: (1) a paper assigned to a venue, (2) multiple venues assigned to a paper, and (3) multiple papers assigned to a venue. A legend box titled 'Drag-&-Drop Operations:' lists five actions: dragging a paper to a venue, a paper to another venue, a venue to a paper, a venue to another paper, and multiple papers/venues to a venue.

SN	Sch	P-ID	Dur	Stds	Seats (Venues Allocated)	(Papers Assigned)	SN	Venue	Sch	Cap	Used	Seats left	
1	SHS-HSN	H1085	2	679	679	G.221(1) (G.221(2)1360)	EG310	1	A.248	NYP	170	139	31
2	SHS-HSN	H3113	2	57			EG310	2	A.339	NYP	330		
3	SCL-CLC	CLC327	2	42	42	S.476	ET3861	3	E.308	NYP	111	20	91
4	SEQ-EGM	EG310	2	139	139	A.248	H1085	4	G.221 (Z1)	NYP	500	500	
5	SEQ-EGM	EGC311	2	23	23	S.476	H1085	5	G.221 (Z2)	NYP	300	179	121
6	SIT-MT	ET3861	2	20	20	E.308		6	G.221 (Z3)	NYP	300		
								7	D.403-407	SBM	134		
								8	S.474	SEG	72		
								9	S.475	SEG	72		
								10	S.476	SEG	72	65	7
								11	H.210	SHS	69		
								12	H.211	SHS	69		

**Drag-&-Drop Operations:**

- Drag a paper to a venue
- Drag a paper to another venue within venues board
- Drag a venue to a paper
- Drag a venue to another paper within papers board
- Drag multiple papers to a venue
- Drag multiple venues to a paper

(1) One paper to one venue; (2) Multiple venues to one paper; (3) Multiple papers to one venue

Fig. 11 Drag-and-drop features for resource allocation

## 8 Exam Timetable Reporting

There are two ways to generate the exam schedule reports. Firstly, the IIEESS 1.0 system can upload the schedule solution to a legacy database if any, so that the existing reporting tools can be used for the students and staff to access the exam schedule, as can be seen from Fig. 8. Secondly, the system can generate the required exam timetables directly using solutions created by the system, as can be seen from Fig. 12 and 13.

## 9 Main Features

- 1.) Most suitable for Large exam scheduling problems, with complex cross school/department registrations (no of candidates can be as high as many thousands), solutions are accurate and robust.
- 2.) User-friendly registration data entry and system parameter setting; ease constraint and scheduling requirement specification.
- 3.) Fast solution searching (runtime is within few minutes for a sizable exam scheduling problem).
- 4.) User-friendly drag-&-drop features, conflict checking is done in few micro-seconds.
- 5.) Advanced reporting and integration tools for schedule output and statistics.

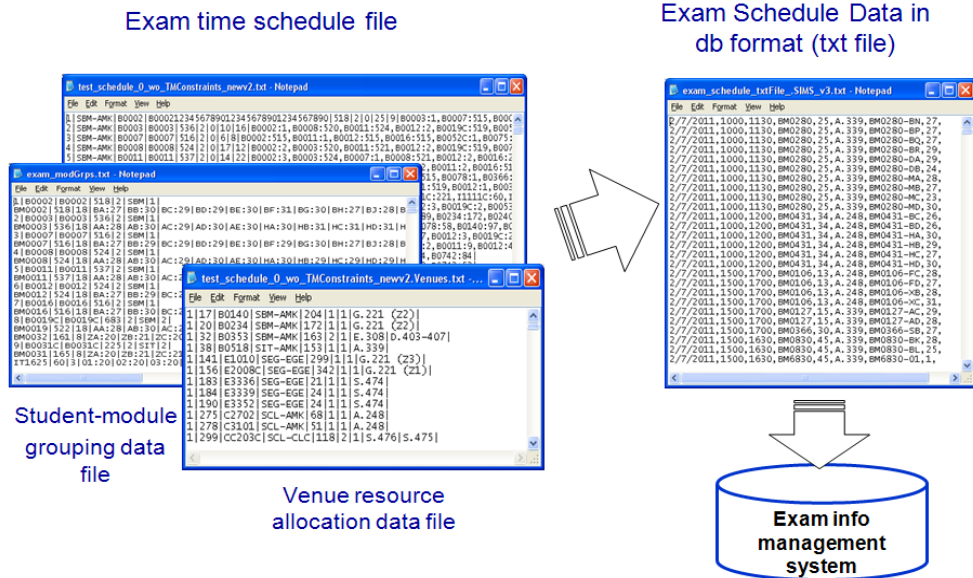


Fig. 12 Export the exam schedule to database

Exam Timetable (Summary)								
Date	Time	Paper	Module	Groups	StdNo	Venue	Senior Inv	
6/27/2011 Mon	0830-1000	Engineering Mathematics 1A/B	EGC101	C1-C4	91	LTR1		
			EGC101/EGB101	C5,B1-B2	68	LTQ1		
			EGB101	B3-B5	66	LTQ2		
			EGD101	D1-D3	56	LTQ3		
			EGF101/EGH101	F1,H1-H2	65	LTQ4		
			EGJ101/CLC101	J1,C1-C2	70	LTQ7		
			CLC101/CLB101/CLG101	C3,B1,G1	72	LTQ8		
			Aerospace Material & NDT Technology	EGC322/EGF301	C1,F1-F2	42	LTQ9	
				1100-1230	Engineering Mathematics 2C	EGC207	C1-C3	65
	EGC207/EGB207	C4-C5,B1				62	LTQ8	
	EGB207	B2-B4				62	LTQ9	
	EGB207/EGD207	B5,D1-D2				63	LTQ11	
	EGD207/EGF207/EGJ207	D3,F1,J1				70	LTQ12	
	Industrial Safety & Loss Prevention	CLC301	C1-C3			63	LTQ1	
		Product Innovation & Rapid Prototyping	CLB301/CLG301	B1,G1	44	LTQ2		
EGC310/EGD306			C1,D1-D2	69	LTQ3			
Automated Machine Design	EGC302	C1-C2	35	LTQ4				

Fig. 13 Exam Timetable Schedule Report

## 10 Hardware/Software Requirements

**Hardware:** IBM PCs, laptops or equivalents; monitor resolution 1680x1060.

**Software:** MS Windows XP or above, MS .NET framework 2.0, or above, MS Office 2005 or above.

## 11 Conclusions

The IIEESS v1.0 system is *highly efficient*; it is much faster than traditional direct-clash-checking systems in terms of solution searching and constraint validation. The system particularly performs well in scheduling large number of exam activities with large number of candidates who registered with multiple modules (papers) cross schools or departments. The runtime for a sizable exam scheduling problem is extremely fast, e.g., in few minutes.

The system is truly *interactive*. It provides user-friendly drag-&-drop features for the planners to book a time slot for an exam before scheduling or to amend the schedule after scheduling. When booking a slot for an exam or amend the schedule auto-generated using the system, full supports will be provided by the IIEESS v1.0 system, such as clash-checking and optimizing solution searching and satisfying constraints imposed. The changes made can be confirmed in few micro-seconds. The new system is *transparent* and *robust*. The IIEESS 1.0 system is always able to generate a solution. If no complete solution exists, the system generates an in-complete solution and indicates the un-scheduled papers and displays reasons why they cannot be scheduled, such as like conflict constraints being imposed.

Although the IIEESS 1.0 system is designed for exam scheduling, the new method and patent technology can be applied into a wide range of other applications, such as transportation planning, sports activity scheduling, vehicle routing and man power scheduling in various production and service industries where event conflictive features and relations must be analyzed before solution searching.

## References

- Nelishia Pillay and Wolfgang Banzhaf, (2007). "A Genetic Programming Approach to the Generation of Hyper-Heuristics for the Incapacitated Examination Timetabling problem", in Progress in Artificial Intelligence, 13<sup>th</sup> Portuguese Conference on Artificial Intelligence, EPIA 2007, Springer, ISBN: 978-3-540-77000-8, pp.223-234,
- Asmuni, H., Burke. E.K., Garibaldi, J.M. (2005). "Fuzzy Multiple Ordering Criteria for Examination Timetabling", vol.3616, pp.147-160. Springer, Heidelberg.
- M. Dorigo, and V. Maniezzo, and A. Coloni. Ant System, (1996). "Optimization by a Colony of Cooperating Agents". IEEE Trans. Sys., Man, Cybernetics 26(1996) 1, p29-4.
- Goldberg, David E. (1989), "Genetic Algorithms in Search Optimization and Machine Learning". Addison Wesley. pp. 41. ISBN 0201157675.

- Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948, ICNN.1995.488968.
- Kennedy, J.; Eberhart, R.C. (2001). Swarm Intelligence. Morgan Kaufmann. ISBN 1-55860-595-9.
- Shi, Y.; Eberhart, R.C. (1998). "Parameter selection in particle swarm optimization". Proceedings of Evolutionary Programming VII (EP98). pp. 591–600.
- Eberhart, R.C.; Shi, Y. (2000). "Comparing inertia weights and constriction factors in particle swarm optimization". Proceedings of the Congress on Evolutionary Computation. 1, pp. 84–88.
- Shu-Chuan Chu, Yi-Tin Chen, Jiun-Huei Ho, (2006). Timetable Scheduling Using Particle Swarm Optimization, Proceedings of the First International Conference on Innovative Computing, Information and Control (ICICIC'06), 0-7695-2616-0/06, © 2006 IEEE.
- UniTime LLC, Examination Timetabling Problem Description, © 2008 - 2011 UniTime LLC, [http:// www.unitime.org/exam\\_description.php](http://www.unitime.org/exam_description.php).
- Examination Timetabling Problem Description,  
[http://www.unitime.org/exam\\_description.php](http://www.unitime.org/exam_description.php)
- Zhu Chunbao, (2008). US patent No. 7,447,669, "Method and System for Timetabling Using Pheromone and Hybrid Heuristics Based Cooperating Agents", November 4, 2008.
- Zhu Chunbao, (2010). PCT Patent Application Number PCT/SG2010/000388, "Method and System for Examination Timetable Scheduling Using Test Paper Conflictive Analysis and Swarm Intelligence", November 1, 2010.

---

# Aspen Scheduler: a Web-based Automated Master Schedule Builder for Secondary Schools

Baiyun Tao, Rick Dwyer

*Follett Software Company/X2 Development Corporation, Hingham, MA, USA*

[btao@FollettSoftware.com](mailto:btao@FollettSoftware.com)

[rdwyer@FollettSoftware.com](mailto:rdwyer@FollettSoftware.com)

## 1 Introduction

Building a master schedule involves assigning courses, teachers and rooms to time slots, maximizing usage of resources and fulfilling student course requests. It is one of most complex tasks performed by administrators at a school each year. This is particularly true for secondary schools in the United States where the number and variety of constraints (such as multi-day rotation, combined courses, room capacities, teacher teaming, and student grouping, etc.) increases the complexity of the scheduling process.

Designed to accommodate both large and small schools with the most complex schedule needs, the system described here, Aspen master schedule builder, implemented by X2 Development Corporation, successfully deals with producing a satisfactory and efficient solution from the large solution space in a reasonable amount of time, while at the same time handling the usability, flexibility and completeness of the scheduling process (see Fig 1 for real life scheduling examples).

## 2 System Overviews

Aspen master schedule builder is part of a Student Information System that is completely web-based (see Appendix A for terms we use in this paper). It is a Java 2 Enterprise Edition (J2EE) application that runs on Apache Tomcat web server, uses Apache Struts framework, OJB and supports three database backend: MySQL, SQL Server and Oracle.



Example School	Scheduling Profile
School - 1	<p>Schedule terms: Full year academic courses, semester elective courses            Number of days per cycle: 5            Number of periods per day: 7</p> <p>Total number of courses: 172            Total number of staff: 53            Total number of rooms: 50            Total number of Students: 700            Number of requests per student: 7-10            Total number of student requests: 5,600            Total number of user specified rules: 29            Total number of sections in master schedule: 370</p> <p>Time for building master schedule: 2 minutes            Time for loading students: &lt; 1 minute</p>
School - 2	<p>Schedule terms: All courses are semester length            Number of days per cycle: 2            Number of periods per day: 8</p> <p>Total number of courses: 314            Total number of staff: 102            Total number of rooms: 90            Total number of Students: 1,200            Number of requests per student: 8            Total number of student requests: 9,600            Total number of user specified rules: 109            Total number of sections in master schedule: 800</p> <p>Time for building master schedule: 10 minutes            Time for loading students: 5 minutes</p>
School - 3	<p>Schedule terms: All courses are semester length            Number of days per cycle: 10            Number of periods per day: 15</p> <p>Total number of courses: 752            Total number of staff: 216            Total number of rooms: 253            Total number of Students: 4,200            Number of requests per student: 16            Total number of student requests: 67,200            Total number of user specified rules: 525            Total number of sections in master schedule: 3,500</p> <p>Time for building master schedule: 90 minutes            Time for loading students: 45 minutes</p> <p>*A special constraint: 4 lunch waves splitting over 3 periods</p>

**Fig.1.** Real life scheduling examples

There are four main objectives used by Aspen master schedule builder to build a schedule:

1. Assign each section with an available teacher, an appropriate time slot and a suitable room.
2. Satisfy all hard constraints.
3. Maximize the satisfaction of student requests.
4. Balance student distribution cross different sections of the same course.

The system breaks the overall problem of building a master schedule into sub-problems, find solutions for each of the sub-problem and combine them to reach an overall solution. An example of a sub-problem is to schedule a section group, a group of sections that are related to each other because of blocking rules (see Fig.4.2) or tightly shared resources such as teacher and room.

The system first orders all sections by their difficulty to schedule. A section's difficulty to schedule is determined by analyzing the related course's attributes, teacher availability, room availability and other constraints of the section. Sections are scheduled in rank order with the most difficult ones being scheduled first. Section group is formed when a section is chosen to be scheduled based on previous analysis. A section group contains one or more sections.

To schedule each section group, an iterative metaheuristic is used that combines hill-climbing, simple random and greedy algorithms. Three key functions are defined: search, validation and evaluation. The search function first finds potential solutions to validate and evaluate. It starts with an initial solution and iteratively works to find better ones. A good solution must be a valid solution – one that meets all the hard constraints. A fitness function is used to check a solution against all hard constraints to objectively determine its validity. Once a solution is determined to be valid, its quality can be measured and scored. The quality of a solution is determined by three primary measures: the number of student requests satisfied, the enrollment balance cross sections of the same course, and the flexibility of the solution, i.e. the ability for students to change sections. An objective function evaluates the solution and assigns a value to the solution.

Solutions then can be compared and the solution with smallest value is considered as the best solution and will therefore be chosen for the section group.

While choosing a solution for the current section group, if the solution affects the ranking of sections still to be scheduled, unscheduled sections are either re-ranked or required resources will be reserved for the future.

When a solution cannot be found for the current section group without violating a hard constraint, the system attempts to repair the schedule by moving resources already scheduled to make the resources needed by the current section group available. If such repair attempt is not possible, the system stops with appropriate feedbacks to the user.

## **3 Key Features**

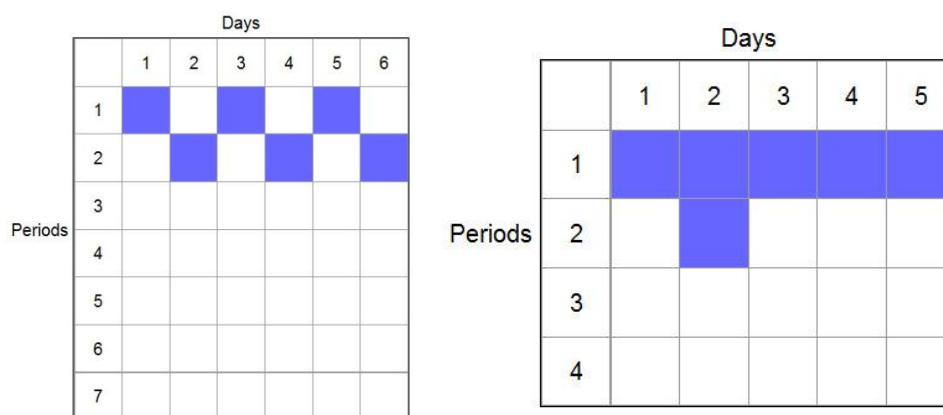
### **3.1 Time Structure**

Schools have their own unique ways to structure time based on the needs to schedule. Successfully building a master schedule depends on an efficient and flexible way to represent these unique time structures. The structure of a schedule is defined by three core parameters: the number of terms per year (TPY), the number of days per cycle (DPC) and the number of periods per day (PPD).

In addition, scheduling patterns are introduced to represent the valid ways to schedule a course of a particular shape. It helps users visualize their schedule (See Fig. 2). Patterns with similar shapes are grouped together as pattern sets, then assigned to courses to define the list of possible time slots a course can be scheduled into.

### **3.2 Input and Constraints**

There are five fundamental input components to build and load a schedule: Course, Student, Staff, Room and Request with each imposing a set of constraints on the schedule. See Fig.3. for the most common constraints associated with each type of input.



**Fig.2.** Different types of schedule patterns

Input	Constraint
Course	<ul style="list-style-type: none"> <li>• Number of sections</li> <li>• Valid schedule patterns</li> <li>• Balance type cross section and subject</li> <li>• Load priority used for conflict resolution for individual student</li> </ul>
Student	<ul style="list-style-type: none"> <li>• Student grouping code: house, team and / or platoon</li> <li>• Unavailable time</li> <li>• Load priority</li> <li>• Enrollment weight</li> </ul>
Staff	<ul style="list-style-type: none"> <li>• Teacher assignments</li> <li>• Preferred room and location</li> <li>• Maximum number of rooms and locations allowed</li> <li>• Maximum number of consecutive teaching periods allowed</li> </ul>
Room	<ul style="list-style-type: none"> <li>• Max capacity</li> <li>• Department and location usage</li> </ul>
Request	<ul style="list-style-type: none"> <li>• Preferred section and / or teacher</li> <li>• Partial content</li> <li>• Inclusion</li> <li>• Section type</li> </ul>

**Fig.3.** Common constraints for each type of input

### 3.3 Rules

Rules are introduced to allow additional and more complex constraints on the schedule. Rules can be defined as either hard or soft. Hard rules must be satisfied in order for the schedule to be valid. Soft rules are satisfied whenever possible - it

is at the discretion of the system. The system will decide which soft constraint to drop in the event of a conflict between soft constraints. The system can also drop a soft constraint if the impact on the overall schedule is too costly such as a valid solution cannot be achieved.

The system currently provides 27 different types of rules for the scheduling process including both build rules and load rules (See Fig.4.1 and Fig. 4.2).

Although the system separates the build and load process. All load rules are included and evaluated during the build process to ensure potential solution don't violate the load rules.

### 3.4 Sandbox Approach

The system takes a sandbox approach to allow users to create and save multiple scheduling scenarios within each school. Scenarios can be copied at any point to serve as a back-up. Each scenario can include different set of students, courses, teachers, and/or rules. Scenarios have their own master schedule that is built and loaded independently of all other scenarios. Users can use different scenarios to experiment with different constraints and even different types of schedules (See Fig.5.).

Scenarios					
		0 of 7 selected 		Custom Selection	
Details	<input type="checkbox"/>	Name	Term	DPC	PPD
Preferences	<input type="checkbox"/>	1. Basic Scenario	1/2,1/1	1	7
Terms	<input type="checkbox"/>	2. Demo Scenario Build	1/2,1/1,1/4	5	7
Days	<input checked="" type="checkbox"/>	3. Demo Scenario Master Matrix	1/2,1/1,1/4	5	7
Periods	<input type="checkbox"/>	4. Room Management Scenario	1/1,1/2	1	7
Rotations	<input type="checkbox"/>	5. Staff Management Scenario	1/1,1/2	1	7
Bell Schedules	<input type="checkbox"/>	6. Student Grouping Scenario	1/1,1/2	1	7
	<input type="checkbox"/>	7. Validate/Build/Load Scenario	1/1,1/2	1	7

**Fig.5.** Different scenarios

<b>Rule Name</b>	<b>Rule Description</b>
Bell - Course Restrictions	Restrict courses for a particular bell schedule
Bell - Room Restrictions	Restrict rooms for a particular bell schedule
Course - Alternates	Schedule alternate courses for students upon conflicts
Course - Blocking	Schedule a set of courses together following particular relationships
Course - Group Exceptions	Specify the courses that can be excluded from any grouping
Course - Pattern Sets Restriction	Restrict usage of patterns for a course
Course - Pull-out	Allow student to be pulled out from one course to go to another
Course - Sequence Concurrent	Schedule two courses in the same term for a student
Course - Sequencing	Schedule a courses in a student's schedule for completion before the start of another course
Course - Term Link	Specify the list of courses that a student must be scheduled in specified terms
Course - Terms Restriction	Restrict usage of terms for a course
Room - Course Restriction	Reserve rooms for a course
Room - Exceptions	Exclude rooms from being used by a course
Room - Proximity	Define room proximity
Room - Reservations	Reserve rooms for courses
Room - Unavailable	Room is unavailable for scheduling during these time slots
Room- Teacher Restriction	Reserve rooms for a teacher
Student - Avoid Teacher	Student should not be scheduled with specified teacher
Student - Student Relationship	Student should be/not be scheduled with specified student
Teacher - Avoid Student	Teacher should not be scheduled with specified student
Teacher - Common Planning	Schedule teachers on a team into a common planning time
Teacher - Concurrent	Allow a teacher to teach multiple courses concurrently
Teacher - Dovetail	Use the minimum periods when schedule a teacher for partial course
Teacher - Max-in-a-row Reset	Reset teacher's max-in-a-row count after a specified period
Teacher - Part-time	Schedule part-time teacher with a minimum span of periods
Teacher - Reserve Time Block	Reserve a time block in a teachers schedule from a list of possible periods
Teacher - Unavailable	Teacher is unavailable for scheduling during these time slots

**Fig.4.1** Different types of rules

**Rule Definitions :: Course - Blocking :: Schedule 121, 123 in a Simultaneous blocking**

Save Cancel

Rules  
▸ Details

General Patterns Teams

Rule priority: Hard

Type: Simultaneous

# of section blocks:

Use classes:

Class ID: US History I & Skills Class max: 30

Use same teacher:

**Courses**

ID	Number	Description
<input type="checkbox"/> 1	121	US History I
<input type="checkbox"/> 2	123	US History I Skills

Save Cancel

**Fig.4.2** Detail of a course blocking rule

## 4 Conclusions

The system demonstrated here provides a complete solution to the scheduling needs of secondary schools. It contains a flexible and friendly user interface with simple patterns and easy-to-understand rules that captures the uniqueness of all type of schedules. The fully automated solver accommodates complex scheduling needs and produces high quality schedules in a reasonable period of time. It also provides powerful tools to provide feedback and assist manual adjustments. In addition, multiple scheduling scenarios can be built separately for a school and then combined into an unified schedule.

## Appendix A. Glossary

*Section:* Section is an instance of a course. A section needs to be scheduled with an available teacher, an appropriate time slot and a suitable room. A course has one or more sections.

*Time slot:* A time slot specifies when a section is scheduled. It contains two components: the schedule term and the day-period combinations within the duration of the schedule term. For example, a section can be scheduled on day 1 period 1 and day 2 period 2 for the first semester of the year.

*Build:* Build is the process which assigns teachers, time slots and rooms to each section in the master schedule. The build also ensures all hard constraints are satisfied.

*Load:* Load is the process which schedules students into sections based on their requests.

*Bell schedule:* A bell schedule specifies the starting time and duration of each period during a day. A school can have more than one mutually exclusive bell schedules used to schedule students in different grades or different campuses.

*Partial course:* A course that is not scheduled every day in the cycle and every term in the year. For example, a semester course or a full year course that meets only 3 days out of the 5 day cycle is considered as partial course.

*Partial content:* Students are allowed to take only a portion of a particular course. For example, a student who fails the second semester of a full year course can take just the second semester of the same course in subsequent year.

*Inclusion:* Inclusion students are special education students who are scheduled into the same section with regular education students.

*Section type:* It is an attribute on both a section and a student request that allows only some students to be scheduled into a particular section.



## A Open source timetable production system for courses and examse

Ruben Gonzalez-Rubio · Balkrishna Sharma  
Gukhool

Received: date / Accepted: date

**Abstract** Diamant is a software system used to manually or automatically produce course and exam timetables at the Université de Sherbrooke, where it has been in use since 2001. We have decided to publish Diamant in Summer 2012 as an open source distribution. To be useful to other institutions, the source code must be modified as little as possible to facilitate the integration of new algorithms and new input/output formats. We present here how the program is organized and where modifications will take place.

**Keywords** University Timetabling · Design Patterns · Diamant

### 1 Extended Abstract

Producing course and exam timetables with Diamant at the Université de Sherbrooke has been done now, for more than 10 years, in different faculties. The course timetable can use post-enrolment mode or curriculum-based mode. We have decided to publish Diamant as an open source project to provide access to other universities that could be interested in using it. We think that, in some cases Diamant can be used as it is. The report [Gon07] is a detailed account of how timetables are produced at the Université de Sherbrooke, and some of its external characteristics were also presented in [Gon10]. The timetables can be produced manually or automatically.

Diamant was designed by our research group `exit` and the work was done by various developers. It was developed in an iterative way, and functionalities were added according to the users demands. The system is divided into two parts: one that takes the

---

Ruben Gonzalez-Rubio  
Directeur du `exit` Lab  
Département de génie électrique et de génie informatique  
Université de Sherbrooke  
Tel.: +819 821 80 00 x 6 29 31  
Fax: + 819 821 79 37  
E-mail: Ruben.Gonzalez-Rubio@Usherbrooke.ca

Balkrishna Sharma Gukhool  
Dpartement de génie électrique et de génie informatique  
Universit de Sherbrooke

data from a database at the University, and formats the data in a way that Diamant can read. The second part is using Diamant to produce the timetables. Here, we present only Diamant, which is written in Java. In some cases, the software could be adapted to particular needs by taking the source code and modifying it. However, the problem with that approach is that there will be  $n$  different versions of the code. A second approach would be to apply, as much as possible, the open-closed principle, which means that the code can be open to modifications by adding new classes but closed to other modifications. This could be achieved by using the design patterns. The advantage is that the code can be customized without having different versions, the objective being to add only new classes. The classes needed for an implementation will be instantiated at run time.

At the core of Diamant is the MVC pattern (Model, View, Controller); the Model is a class containing a set of events, a set of instructors, a set of rooms and a set of students. Furthermore, the Model contains also a timetable which consists of slots (Periods), where Events will be assigned. The Model keeps the state of the timetable. The Views give the user a visual representation of parts of the sets or the timetable. The Controllers are there to take user interactions to change the Model, and when the Model is updated, the View is also updated and the user can see the change on the screen. The raw data used in the sets are only integers and strings, which helps to display them in the Views. Manual operation is carried out by opening a dialog, from a menu, the dialog is in fact a View, changing some data in the Model, and sending a feedback to the users in a View. Automatic operation is triggered by selecting an optimisation algorithm from a menu. The algorithm makes data assignments avoiding conflicts and optimising when possible, and when no more optimisation is possible, the algorithm quits and the Model is updated. The Model data is read from a file and written to a file, that preserves the state of the Model, which can be the final state when the timetable is finished.

Before publishing Diamant as it is, we are conscious of some of its disadvantages :

- First, that the code was implemented to satisfy the Université de Sherbrooke needs.
- The code is not homogenous, as it was written by various programmers.
- Finally, the system is in French only.

The last disadvantage is easy to change. To offer the application in other languages, we think that we must provide resources to facilitate internationalization; we need to have external messages in English and in French, as well as examples of how to work in another language. The messages of each language will be stored in a language resource bundle. So, normally no modification to the code is needed.

The first two disadvantages may need a large refactoring of the program. This refactoring will be done in an iterative way.

Our two goals before doing the refactoring are:

- We must offer an program that can be used by other developers, in principle they will add only new classes.
- The refactoring must reduce the class coupling, which is high in the current version.

These two goals are compatible. Diamant will be published as it is and new versions containing the refactoring work will be published.

In the next section, we present some details of the system to illustrate how it is organized.

## Details of the work

Basically, a timetable is prepared from data concerning the timetable: the timetable itself, the events, the students, the instructors and the rooms. At the end, each event is assigned to a Period in the timetable, for each event a room, an instructor and a set of students are assigned. When the work is done manually, the assignments are done by indicating each assignment via the GUI. For example, in a dialog box, it can be indicated that an event will take place Monday at 13h00, and another Wednesday at 14h00, and so on. In all dialog boxes, there are list of values where the user can select one. By design there are no text boxes where any value can be typed. When the work is done by the optimisation algorithm, it will allocate automatically all possible assignments.

When trying to produce a timetable, conflicts can arise; so the system must indicate this. In the case of automatically producing the timetable, the program tries to reduce to a minimum the number of conflicts.

To explain the program, we distinguish the following parts:

- The input and output of data.
- The GUI and the manual data manipulation
- The optimisation algorithms

### 1.1 Using the design patterns

A design pattern is a general reusable solution to a commonly occurring problem within a given context in software [GHJV95]. We are going to refer to some of these design patterns.

We already mentioned that Diamant is organized using the MVC pattern (Model, View, Controller). The Model is a big class it contains the data representing the state of the timetable, those data are divided in classes SetOfX<sup>1</sup> and a Timetable. The sets of instructors, of rooms and of students are used as information that help decisions when a timetable is produced. For example the characteristics of a room (size among others) can help to assign the room to an event. The sets of events and the time table are used to keep the state of the Model when a decision is taken.

There are various Views and various Controllers. The Views give the user a visual representation of parts of the sets or the timetable. The Controllers are there to take user interactions (a external change in the View) and send them to the Model, and when the Model is updated the Views are also updated, and the user can see the change on the screen.

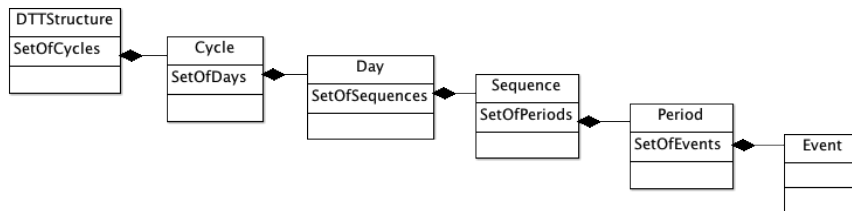
For example in a View, that can be a dialog, it is possible to assign a instructor to an event. That can be done by selecting the instructor name for this event in the View, when the button "ok" is pressed the action takes place, that means that the Controller sends the name of the instructor to be assigned to the corresponding event in the Model where the change takes place.

The Model gets its data from a file and at the beginning sends its current state of the model to a file when some work is done. This work can represent the final state or an intermediate state.

---

<sup>1</sup> Where the SetOfX could be a set of events, a set of instructors, a set of rooms and a set of students.

The DTTStructure represents the timetable with its Model organized hierarchically Figure 1. It can contains  $c$  cycles, each cycle contains  $d$  days (normally 5 days), that can be seen as a week, in a day there can be  $s$  sequences, and each sequence can contain  $p$  periods. There is no overlap in periods contained in a sequence, and there is no overlap in sequences for a day. With this organization, it is possible to offer a very flexible way to define a timetable. The periods can be seen as slots where events are assigned.



**Fig. 1** The timetable hierarchy

Where the sets are lists of Objects that are defined for the type of its constituents.

## 1.2 The input and output of data.

The raw data (input and output) used in the sets are only integers and strings. This helps to display them in the Views and a developer can read the files if needed.

We can have the following scenarios with the data:

- The data used in an other university are the same as that of Diamant, then there is no work to do.
- The new data can be mapped onto the Diamant data; maybe some attributes have another name. That can be done by reading the new data and put the data in a file that can be read by Diamant.
- The new data are a subset of Diamant's data, that means that some of the Diamant data must be initialized to safe values; these values are safe when there are no exceptions and the optimisation algorithm does not use these values.
- The new data contains new attributes, in which case these new attributes must be part of the data.

The first three scenarios are easy to deal with. The last one, however will need the usage of the Strategy pattern, Figure 2. In that pattern a Diamant file (one for each type of data) can be read by one class, namely DiamantFormat and a different format can be read by another class NewDataFormat. The readFile is a variable of type ReadFileBehavior and each class in the Model has one. That means that the readFile method can be fixed at runtime. The interface ReadFileBehaviour contains one method readFile. The classes DiamantFormat and NewDataFormat implement this method, so the variable readFile is used to call readFile without knowing which class is instantiated. For each new format only new classes will be added.

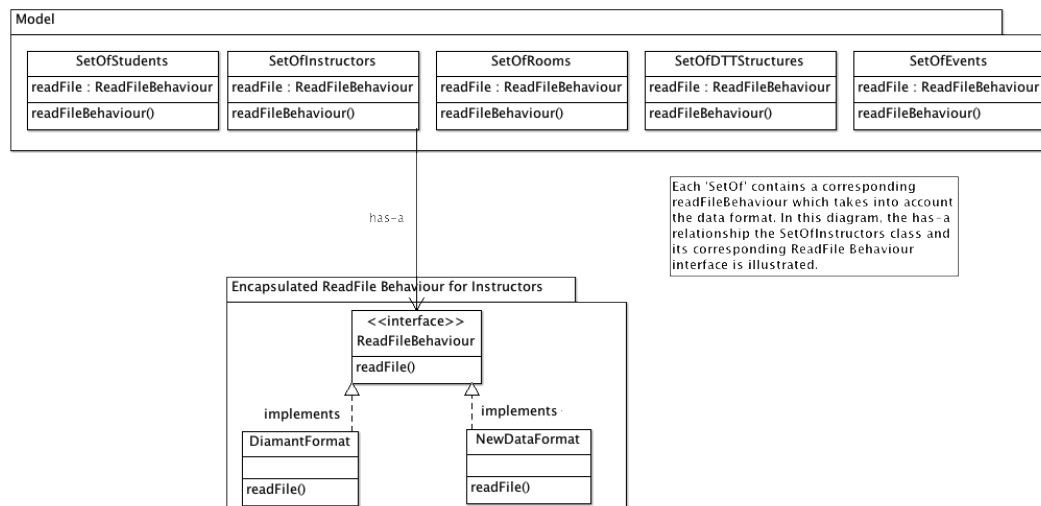


Fig. 2 Strategy pattern for reading files

### 1.3 The GUI and the manual data manipulation

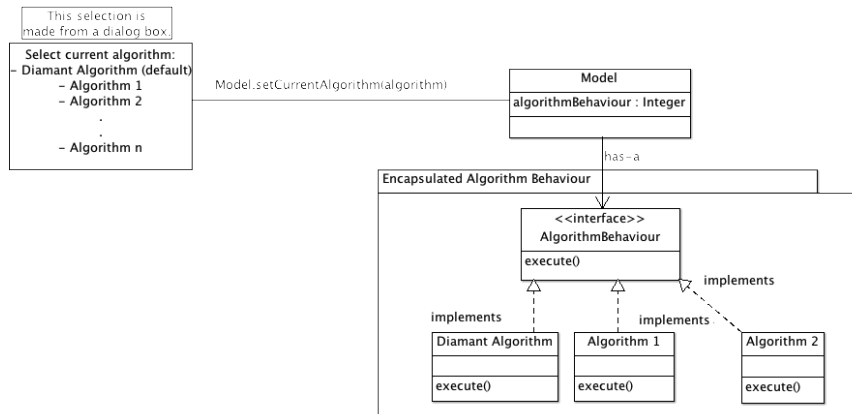
The Model contains two types of data: immutable and mutable. The mutable data are used to make assignments (see next subsection). All the other data in the Model are immutable, that means that they do not change during program runtime. For example, it is not necessary to change an instructor name by Diamant to produce a timetable. If any change is needed on immutable data, it must be done outside Diamant. For example, if an instructor name has an error, it must be changed somewhere else, possibly in the database, then the files can be reloaded into Diamant. To add attributes to the Model, there are new PropertyLists containing couples with each couple having the name of the attribute and the value. So for example, in a University they add an attribute sex for students. This can be done by adding a list of attributes to the students in that list the new attribute will be a couple with the String sex and the value M or F. This information can be used to set groups with an appropriated mix of students. The new attributes must be also immutable. All the dialog boxes will implement interfaces and this allows the addition of a new dialog box which can display the new attribute. Again, modifications to offer new functionalities will be done by adding new classes.

### 1.4 The optimisation algorithms

The timetable is built by assigning events to Periods and assigning data to Events. In fact, to make an assignment, Periods and Events contain instance variables, which are lists. For example to assign an Event to a Period, the Event is added to the list; when the Event is removed from the list, then the assignment is no longer valid. Manual operation allows the assignment process by using dialogs, where choices are made by selecting a value from a list and this changes the model. Automatic operation is started by a menu where an optimisation algorithm is defined; when started it works on the

Model. It makes assignments and removes assignments to minimize conflicts. When the algorithm finishes, the View is updated, showing the current status of the timetable.

Also different optimisation algorithms can be implemented by a strategy as illustrated by Figure 3.



**Fig. 3** The strategy pattern for optimisation algorithms

Each strategy can be instantiated when a menu item is activated; if the menus are implemented by the command pattern, then when a new command is needed a new class is written and the menu can be added to the configuration.

In some cases the algorithm could need the data in a different data structure, for example instead of using the timetable of Diamant a simple matrix can represent the days and the periods. That means that the data in the Model must be mapped onto new structures, so the algorithm will be divided in three steps : mapping the Diamant Model onto the new structure, the optimisation algorithm and mapping the result onto the Diamant Model.

## Conclusion

We think that the system Diamant could be used and extended for other universities. Customisation can be done in a simple way without having to rewrite large parts of the system. We will be glad to help other institutions to adapt Diamant to their special needs. Finally the diamant URL is:

<http://tictacserver.gel.usherbrooke.ca/exitopenprojets/>

## References

- GHJV95. E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- Gon10. R. Gonzalez Rubio. Diamant: A timetable production system for courses and exams. In *PATAT'10*, Belfast, Northern Ireland, August 2010.
- Gon07. R. Gonzalez-Rubio. La production et la consultation d'horaires dans une université. Technical report, Université de Sherbrooke, 2007.

---

# School Time Tabling ITT software

Ilana Cohen-Zamir

Doron Bar<sup>1</sup>

Abstract

We have developed software for automatic school timetable scheduling. The system is flex and can handle different types of schools and requirements. In addition, we developed complexity indicators for a given school. This may help to predict if there is any solution, as well as serve as a comparable tool.

*School timetable scheduling characteristics, complexity indicators, cloud*

## 1. Introduction:

School scheduling in our country (Israel) is usually performed manually by a school specialist. The process is assisted by a computer aided tool that performs validation tests at every step, as well as supplies variety of information to help the specialist with the tedious work. However, the actual scheduling steps are manually planned by the specialists themselves.

The specialist cannot predict, if there is any solution for a given school dataset.

Usually, when they reach a dead-end, they simply replace the requirements.

We performed a full scheduling on several schools, supporting all their data flow: we got the data on papers, and delivered the final web timetables and reports. We found that we also could not predict the success of our software, when we got school dataset. Therefore, we suggest complexity indicator for a given dataset. These indicators may serve as a comparable tool, but they can also help school principal to prepare their data set, at the early stage.

---

<sup>1</sup>Ilana Cohen-Zamir

Tel. 972-0775302938 972-523368710

[ilanacohenz@walla.co.il](mailto:ilanacohenz@walla.co.il)

Doron Bar , Technion

Tel. 972-504039288

[doron.e.bar@gmail.com](mailto:doron.e.bar@gmail.com)

## 2. Problem characteristic description:

At this section, we describe the special characteristic that we had to contend with. For simplicity, we will refer to the case of 30 teachers, 6 days a week.

### 2.1. Teachers calendar system: a day-off for every teacher

In Israel schools run 6 days a week, but in our educational system all teachers have at least one day off. It is called a "free day." This has a major effect. Therefore, the scheduling task becomes dramatically harder.

### 2.2. General teachers weekly balancing:

In addition to the "free-day" system, in elementary schools teachers usually can't take their day- off on some of the days: global-meeting days (e.g. Sunday) as well as Friday. Therefore - instead of having in every day the same amount of available teachers - some of the days may be overloaded with "too many" teachers, and on other days there are "not enough" teachers to be scheduled for the required lessons time slots. To make scheduling possible, there must be minimum number of teachers that can work in every day. In addition to this, Friday is a short day. So, there are too many teachers, that are at school, but they "don't have work" on this day.

### 2.3. Specific teachers weekly balancing:

In elementary schools teachers "days-off" scheduling and teachers substitutions have a major effect on each other. This happens because when the main teacher takes a day off, other teachers have to "fill" this day. Therefore, all other class teachers have to be at school on this day, and they cannot take this day off. Therefore, there is a long chain of effects between the teachers that teach in the same class. This problem is critical in middle size schools, while most of the class hours are taught by one teacher. The principle is that the teachers have to "help" each other to fill each others day-off. For example, we had a case that 2 sport teachers took the same day off. The result was that most of the home class teachers could not take this day off at all. This



caused a chain of other effects, and we had to perform a special algorithm to find a combination of possible teachers' days off (potential space of  $6^{30}$  options).

Note, this would not happen, when class work load is divided among several teachers – as it is held in high schools and middle schools.

#### **2.4. Part time teachers effect:**

Part time professional teachers are frequently a bottle-neck: Arabic, Music and Sport and English (as a second language) teachers often come only for two-three days in a week. Also, these subjects usually must be taught in different days. Therefore, in many schools all school scheduling is determined majorly according to these teachers.

#### **2.5. "Ofek Hadash" requirements:**

Another new huge constraint was added recently: now in elementary schools teachers cannot start work at the second hour or later or leave school earlier, and their empty time slots must be minimized.

#### **2.6. Classes calendars**

In most of the classes at elementary schools the class calendar is fixed. Students must leave after 5 hours, and although the teacher is available, students are not there. Special education classes calendar is also fixed.

#### **2.7. Split lessons and concurrent lessons**

Recently, in some of the lessons, classes have to split into two groups with two teachers for different subjects. This constraint means that an hour that both teachers are available should be found. Another requirement is that sometimes lessons in different classes have to be concurrent: i.e. few classes have to be scheduled to the same time slot.

### 2.8. Grouping, clusters of courses and splits lessons:

In high schools in Israel, in contrast to other countries, schools classes continue to be held as "home classes". But, most of the lessons are not delivered to the class: N classes are grouped together, and then they are split to M different group classes. M can be > N, < N or == N. This is a complex system, and the description of this system is behind the scope of this article.

Fig 1. Example: 'second language' cluster displayed in home classes view.

Combining home-classes scheduling system with course classes scheduling is needed, and we developed a hybrid model that combines two scheduling systems.

### 3. Complexity measuring

While looking at different schools, we wanted indicators to be able to compare between schools complexity, as well as to be able to compare between two algorithms, or even between two specialists.

We look at problem complexity measurement not as one calculated number,

but as a vector of complexity separate indicators:  $\langle I1, I2, I3, \dots In \rangle$

A school may be complex in some of the indicators, while it is simple in other indicators. Therefore, we don't want to "smash" them all to one "average" number.

3.1.  $I_1$  - cluster lessons participants counter

This Factor takes into account the complexity of the cluster lessons.

$Re\ qL$  Group of all required lessons.

$C_{ij}$  Number of classes that participate in lesson  $li$

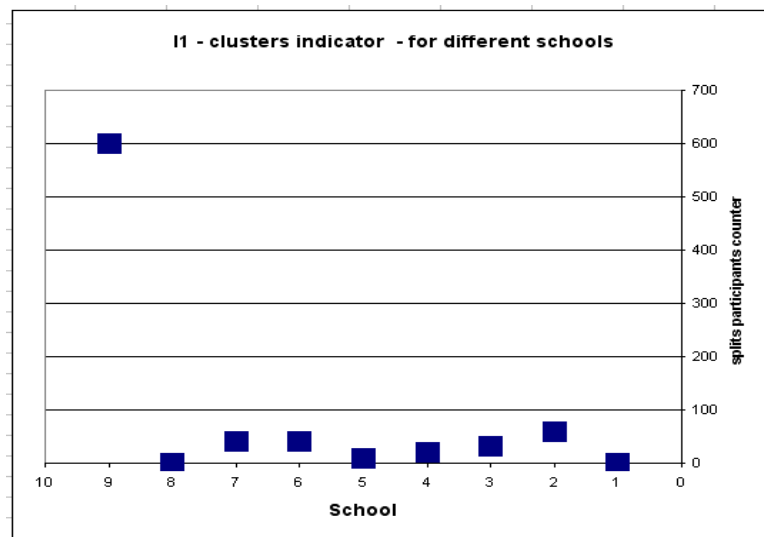
$Tik$  Number of teachers that participate in lesson  $li$

$$TLMcMt = \sum_{li \in Re\ qL} \left( \sum_{C_{ij} \in li} C_{ij} \right) \times \left( \sum_{Tik \in li} Tik \right)$$

$$L_{total} = \sum_{li \in Re\ qL} li$$

$$I_1 = \frac{TLMcMt}{L_{total}}$$

- For schools, that has no clusters the value of this indicator is 0
- Complex clusters with, say 10 classes grouped into 12 teachers has major effect on this complexity indicator.
- A lesson that is split to two teachers or two classes has a small effect on this indicator.



3.2.  $I_2$  - requirements to availability to ratio

This factor computes a weighted average of the ratio between each *teacher*  $\cap$  *class* availability to the *teacher*  $\cap$  *class* total required hours.

$L_{total}$  Total number of all required lessons at school

$$THR_{ij} = \sum_{hour\ h_k} \left( Req(t_i, c_j) \right)$$

Total required hours of teacher  $t_i$  to class  $c_j$

$$THA_{ij} = \sum_{hour\ h_k} \left( \left( Available(t_i, h_k) \right) \wedge \left( Available(c_j, h_k) \right) \right)$$

Total available hours intersection

$$T_{ij} = \frac{THR_{ij}}{THA_{ij}}$$

$$TRi_{total} = \sum_{c_j} THR_{ij}$$

Total teacher  $t_i$  required hours in all classes

$$TTi = \sum_j T_{ij} \times \left( \frac{THR_{ij}}{TRi_{total}} \right), \text{ only if } THR_{ij} \neq 0$$

The average of specific teacher

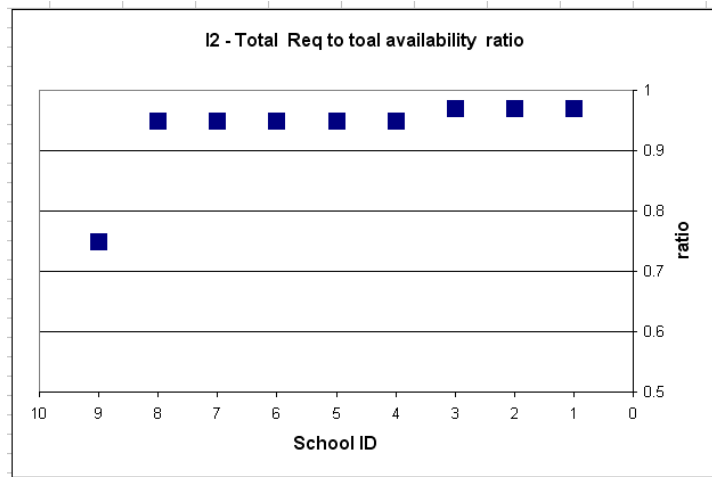
$$I_2 = \sum_i TTi \times \left( \frac{TRi_{total}}{L_{total}} \right)$$

Average of all teachers

First, for every couple,  $\langle \text{teacher } t_i, \text{class } c_j \rangle$  the ratio is computed.

Then an average is computed.

Here, for simplicity, the graph we show the ratio for the school as total number:



### 3.3. $I_3$ - rooms requirements to availability to ratio

Similar factor, that computes a weighted average on all the rooms of the ratio between each  $class \cap room$  availability to the  $class \cap room$  total required hours.

### 3.4. $I_4$ =

This factor takes into account only the max of the ratio between each  $teacher \cap class$  availability to the  $teacher \cap class$  total required hours.

$$THR_{ij} = \sum_{hour\ h_k} \left( Req(t_i, C_j) \right)$$

Total required hours of teacher  $t_i$  to class  $C_j$

$$THA_{ij} = \sum_{hour\ h_k} \left( \left( Available(t_i, h_k) \right) \wedge \left( Available(C_j, h_k) \right) \right)$$

Total available hours intersection

$$T_{ij} = \frac{THR_{ij}}{THA_{ij}}$$

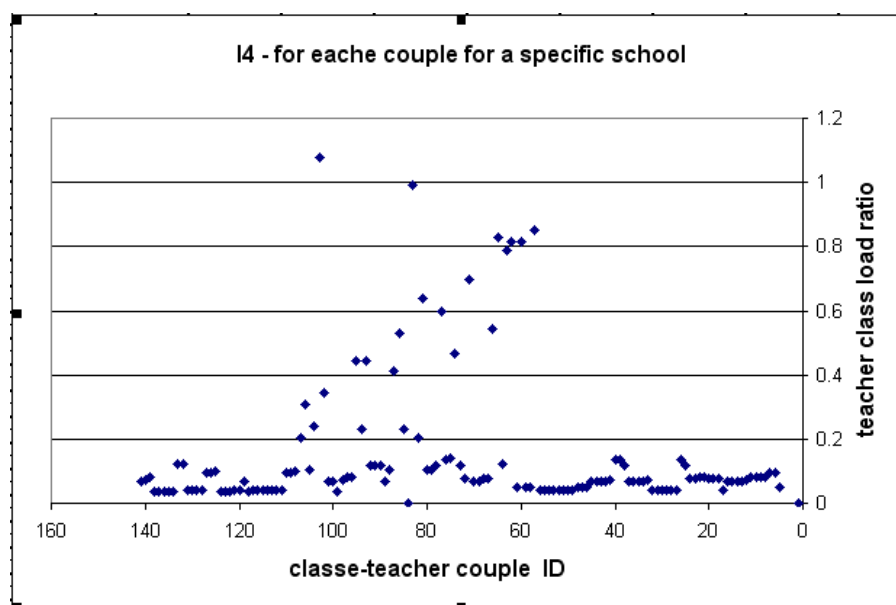
$$M_{c_j} = \text{Max} \{ ij \mid Req(t_i, C_j) \} \quad \text{Max ratio for specific class}$$

$$I_4 = \sum_{c_j \forall c_j} if (M_{c_j} \geq 0.8)$$

Total number of all classes, with "bad"  $M_{c_j}$

First, for every couple,  $\langle \text{teacher } t_i, \text{class } c_j \rangle$  the ratio is computed.

Then, only the max for each class is taken. Then the total number of the classes, that their ratio is  $\geq 0.80$ , is counted. This is computed only for teachers that don't work every day.



### 3.5. $I_5$ - class-lesson-day- constraints counter

Number of lessons that cannot be scheduled at the same day /  $L_{total}$   
(E.g. sport). Here, we only count lessons of different teacher.

### 3.6. $I_6$ - percentage of constraints second indicator:

Number of lessons that must be scheduled in sequence /  $L_{total}$   
(e.g. science lab). Here, we only count lessons of different teacher.

### 3.7. $I_7$ - Spreading of class Load:

Total number of classes that home class teacher has a day off, and that the ratio between home teacher total hours to the total class hours is  $\geq 0.8$

### 3.8. $I_8$ - days effect

This factor is relevant for the teachers "free-day" system.

Count for all classes that home class teacher has a day off: How many different teachers must be at school, in the day-off of the home class teacher. To calculate it, we use a simple method.

Example: if the home class teacher teaches 23/29 class lessons; and all other lessons cannot be taught at the same day, then all other 5 teachers must be at school at her day of. So, this class increases  $I_8$  by 5. Note,  $I_8$  is not normalized.

### 3.9. $I_9$ - Data flexibility –

there are cases that data case be changed easily (like switching between teachers or replays working days) , while in other cases the data requirements are fixed. This indicator may be input to a table manually by the user. We can't calculate in for a specific input data.

This indicators vector should be further investigated and improved.

## 4. Results , the software and conclusions

### 4.1. The existing process

Today, in our country, the scheduling tasks lasts a long time and it is performed either by a dedicated internal school specialist - at high schools - or it is outsourced to a specialist for a fee. Usually, the constraints are so difficult, that the problem is not solved. Therefore, when the specialist reaches a dead-end, ( consulting with the school principal), he changes some of the input data. In other words, in order to solve the problem, the problem is changed. This trial by error process continues. Accordingly, during this procedure, the input data is changed: some courses are re-substituted, some

teachers are switched, and some teachers' working days are changed. When the scheduling reaches a dead-end, the specialist knows how to choose which data re is a wide range of options to choose from, to replace the input data. Specialists have developed heuristics not only to choose the next scheduling steps: they also developed heuristics to choose which and how input data should be replaced to make the scheduling "work".

This is one of the reasons why they don't believe that this process can be automatic.

In addition to this, there are also external sources for input data changes: During the scheduling process, teachers' total hours budgets are often increased or decreased by external cause.

Another common problem is that data is often updated after scheduling is done: re-scheduling, is usually out of question, and therefore the "old" tables are used ,and the new teachers cannot be utilized to the system efficiently.

## **4.2. Our Test cases**

IttTimeTable has performed a full scheduling on several schools, including all their data flow, and generated the tables, posters and reports. We got the data on papers, and delivered the final timetables

Using this system enabled the schools to change the input data many times . This would never be possible with the existing manual procedure. That is, because of the clear fact that the specialist would not "throw away" all the work he/she has already done, and re start it all over.

## **4.3. Results, and success ratio**

External vendor scheduling specialists claim to promise to satisfy at least 80% of the school requirements and constraints. We don't know the actual success ratio.

IttTimeTable has scheduled 97%-99% of all the cells (a cell per hour) in the tables. The remaining 1-2% were scheduled manually, using IttTimeTable commands language.





## 4.5. Conclusion

- Today, in our country, school scheduling specialists do not believe that a computer program can perform the scheduling, and replace their wisdom.
- IttTimeTable success ratio is 97%-99%.
- IttTimeTable is currently being converted to a cloud application.
- IttTimeTable software is at beta phase, and we hope to be of valuable service in this area.

# The Third International Timetabling Competition

# The Third International Timetabling Competition

Gerhard Post, Luca Di Gaspero,  
Jeffrey H. Kingston, Barry McCollum, and  
Andrea Schaerf

Received: date / Accepted: date

**Abstract** This paper is the organizers' report on the Third International Timetabling Competition (ITC2011), run during the first half of 2012. Its participants tackled 35 instances of the high school timetabling problem, taken from schools in 10 countries.

**Keywords** High school timetabling · International timetabling competition

## 1 Introduction

High school timetabling is a long-established area of timetabling, but it has received less scientific attention than some other areas, such as university timetabling. Its research community has been fragmented, and very little data has been shared.

Over the last few years, a group of researchers has worked on this problem. There is now an XML data format in which unsimplified instances and solutions of the problem can be specified precisely (Post et al. 2012, 2010b), and a web site for evaluating solutions (Kingston 2010). Instances have been widely sought. At the time of writing, 35 instances from 10 countries are available for download (Post 2011).

Building on this work and on two previous competitions (Paechter et al. 2002; McCollum et al. 2007), and supported financially by the PATAT conference series, EventMAP, and the Centre for Telematics and Information Technology at the University of Twente in the Netherlands, the Third International Timetabling Competition (ITC2011) was run by the authors in the first half of 2012.

This paper is the organizers' report on that competition.

---

Jeffrey H. Kingston  
School of Information Technologies  
The University of Sydney  
E-mail: jeff@it.usyd.edu.au

## 2 Modelling high school timetabling

The competition used a data format called XHSTT in which high school timetabling problems and solutions can be expressed. This format has been described previously (Post et al. 2012, 2010b), and a full specification is available online (Kingston 2010). An overview, omitting syntactic details, is given here for completeness.

An XHSTT file is an XML file containing one *archive*, which consists of a set of instances of the high school timetabling problem, plus any number of *solution groups*. A solution group is a set of solutions to some or all of the archive's instances, typically produced by one solver. There may be several solutions to one instance in one solution group, for example solutions produced by the same stochastic solver, but using different random seeds.

Each instance has four parts. The first part defines the instance's *times*, that is, the individual intervals of time, of unknown duration, during which events run. Taken in chronological order these times form a sequence called the instance's *cycle*, which is usually one week. Arbitrary sets of times, called *time groups*, may be defined, such as the Monday times or the afternoon times. A *day* is a time group holding the times of one day, and a *week* is a time group holding the times of one week. For the convenience of display software, some time groups may be labelled as days or weeks.

The second part defines the instance's *resources*: the entities that attend events. Resources are partitioned into *resource types*. The usual resource types are a Teachers type whose resources represent teachers, a Rooms type of rooms, a Classes type of *classes* (sets of students who attend the same events), and a Students type of individual students. However, an instance may define any number of resource types. Arbitrary sets of resources of the same type, called *resource groups*, may be defined, such as the set of Science laboratories, or the set of senior classes.

The third part defines the instance's *events*: meetings between resources. An event has a *duration* (a positive integer), a *time*, and any number of *resources* (sometimes called *event resources*). The meaning is that the resources are occupied attending the meeting for *duration* consecutive times starting at *time*. The duration is a fixed constant. The time may be preassigned or left open to the solver to assign. Each resource may also be preassigned or left open to the solver to assign, although the type of resource to assign is fixed. Arbitrary sets of events, called *event groups*, may be defined. A *course* is an event group representing the events in which a particular class studies a particular subject. Some event groups may be labelled as courses.

For example, suppose class 7A meets teacher *Smith* in a Science laboratory for two consecutive times. This is represented by one event with duration 2 containing three resources: one preassigned Classes resource 7A, one preassigned Teachers resource *Smith*, and one open Rooms resource. Later, a constraint will specify that this room should be selected from the *ScienceLaboratories* resource group, and define the penalty imposed on solutions that do not satisfy that constraint.

If class 7A meets for Science several times each week, several events would be created and placed in an event group labelled as a course. However, it is common in high school timetabling for the total duration of the events of a course to be fixed, but for the way in which that duration is broken into events to be flexible. For example, class 7A might need to meet for Science for a total duration of 6 times per week,

**Table 1** The 15 types of constraints, with informal explanations of their meaning.

Name	Meaning
Assign Resource constraint	Event resource should be assigned a resource
Assign Time constraint	Event should be assigned a time
Split Events constraint	Event should split into a constrained number of sub-events
Distribute Split Events constraint	Event should split into sub-events of constrained durations
Prefer Resources constraint	Event resource assignment should come from resource group
Prefer Times constraint	Event time assignment should come from time group
Avoid Split Assignments constraint	Set of event resources should be assigned the same resource
Spread Events constraint	Set of events should be spread evenly through the cycle
Link Events constraint	Set of events should be assigned the same time
Avoid Clashes constraint	Resource's timetable should not have clashes
Avoid Unavailable Times constraint	Resource should not be busy at unavailable times
Limit Idle Times constraint	Resource's timetable should not have idle times
Cluster Busy Times constraint	Resource should be busy on a limited number of days
Limit Busy Times constraint	Resource should be busy a limited number of times each day
Limit Workload constraint	Resource's total workload should be limited

in events of duration 1 or 2, with at least one event of duration 2 during which the students carry out experiments. One acceptable outcome would be five *sub-events*, as these fragments are called, of durations 2, 1, 1, 1, and 1; another would be three sub-events, of durations 2, 2, and 2. This is modelled by giving a single event of duration 6. Later, constraints specify how this event may be split into sub-events, and define the penalty imposed on solutions that do not satisfy those constraints.

The fourth and last part of an instance contains an arbitrary number of *constraints*, representing conditions that an ideal solution would satisfy. There are 15 types of constraints, stating that events should be assigned times, prohibiting clashes, and so on. The full list appears in Table 1. More types may be added in the future, if necessary.

Each type of constraint has its own specific attributes. For example, a Prefer Times constraint lists the events whose time it constrains, and the preferred times for those events. Each constraint also has attributes common to all constraints, including a Boolean value saying whether the constraint is hard or soft, and an integer weight.

The *infeasibility value* of a solution is the sum over the hard constraints of the number of violations of the constraint multiplied by its weight. The *objective value* of a solution is similar, only summed over the soft constraints. One solution is considered better than another if it has a smaller infeasibility value, or an equal infeasibility value and a smaller objective value.

As mentioned earlier, solutions are stored separately from instances, in solution groups within the archive file. A solution is a list of sub-events, each containing a duration, a time assignment, and some resource assignments. The HSEval web site (Kingston 2010) calculates the infeasibility and objective values of the solutions of an archive, and displays comparative tables, lists of violations, and so on.

### 3 The competition

The competition attracted 17 registrations, although only 5 teams submitted solutions in the end. This is many fewer than the over 100 registrations and about 40 active

**Table 2** The source country, instance name, number of times, teachers, rooms, classes (groups of students), individual students, and events, of each of the 21 instances of Round 1 (archive XHSTT-2012.xml).

Country	Instance	Times	Teachers	Rooms	Classes	Students	Events
Australia	BGHS98	40	56	45	30		387
Australia	SAHS96	60	43	36	20		296
Australia	TES99	30	37	26	13		308
Brazil	Instance1	25	8		3		21
Brazil	Instance4	25	23		12		127
Brazil	Instance5	25	31		13		119
Brazil	Instance6	25	30		14		140
Brazil	Instance7	25	33		20		205
UK	StPaul	27	68	67	67		1227
Finland	Artificial	20	22	12	13		169
Finland	College	40	46	34	31		387
Finland	HighSchool	35	18	13	10		172
Finland	Secondary	35	25	25	14		280
Greece	HighSchool1	35	29		66		372
Greece	Patras2010	35	29		84		178
Greece	Preveza2008	35	29		68		164
Italy	Instance1	36	13		3		42
Netherlands	GEPRO	44	132	80	44	846	2675
Netherlands	Kottenpark2003	38	75	41	18	453	1156
Netherlands	Kottenpark2005	37	78	42	26	498	1235
South Africa	Lewitt2009	148	19	2	16		185

teams of ITC2007. Why fewer teams registered is not known, but it could be because of high school timetabling's lower profile, or because the 15 types of constraints make the instances awkward to handle in practice.

The competition had three independent parts, called Rounds 1, 2, and 3. In Round 1, participants were invited to submit solutions to 21 published instances. No restrictions were placed on how the solutions could be obtained. For each instance, a small prize was awarded to the participant who submitted the best solution to that instance, if it improved on the best solution previously known to the organizers. Such improved solutions were found to 15 of the 21 published instances during Round 1.

The 21 published instances are listed in Table 2. The table shows that they differ greatly in size. For example, the number of times varies between 25 and 148, and the number of classes (groups of students) varies between 3 and 84. They also differ in structure: some require most events to be split into sub-events, others do not; some require teachers to be assigned as well as rooms, although most do not; and so on.

Round 2 compared solvers under uniform conditions. Solvers were allowed to use only freely available software libraries, and a time limit was imposed: 1000 seconds of single-threaded execution per instance on the organizers' computer. Participants used a benchmark program to estimate how much time on their own computer was equivalent to the time limit on the organizers' computer.

For initial testing, the Round 1 instances were used, except that the Australian instances from Table 2 were omitted. This was because these are the only instances so far to use Avoid Split Assignments and Limit Workload constraints, and omitting them reduced the implementation burden for the participants.

The original plan was to select up to 5 finalists based on this initial testing, but since there were only 5 active teams, all were invited to become finalists. One team declined owing to problems with their solver. This left 4 finalists, who submitted their solvers, which the organizers ran on their own computer. For each of 18 hidden instances, and for each of 10 random seeds, each solver was run on that instance and seed for at most 1000 seconds, and the solutions for that instance and seed were ranked. Each solver's ranks were averaged. The solver with the lowest average rank was declared the winner.

The hidden instances were instances not previously published, although not very different from the published instances, and consisted of 14 completely new instances plus 4 small corrections or variants of the published instances. Although none of the organizers were participants, one finalist contributed some of these hidden instances. Their results on their own instances were excluded from the rankings.

The Round 2 finalists have described their solvers in invited short papers for the PATAT 2012 conference, written before results were announced (Domrös and Homberger 2012; Fonseca et al. 2012; Kheiri et al. 2012; Sørensen et al. 2012).

After Round 2, the hidden instances were published and Round 3 was conducted on them. As for Round 1, no restrictions were placed on how the solutions could be obtained. The participant with the lowest average rank over the hidden instances was declared the winner, again excluding participants' results on their own instances.

After the competition ended, the solutions were published on the competition web site, as XHSTT archive files. The results of Rounds 2 and 3 were calculated by passing these files to the appropriate HSEval operation, so can be publicly verified.

#### 4 Conclusion

The aim of the competition, in brief, was to raise the profile of high school timetabling. This it has undoubtedly done.

Judging by previous competitions, the instances used in ITC2011 are likely to continue to attract researchers for years to come. These are unsimplified instances from real high schools around the world—a great foundation to build on.

#### References

- Jonathan Domrös and Jörg Homberger, An evolutionary algorithm for high school timetabling, Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), Son, Norway, August 2012
- G.H.G. Fonseca, H.G. Santos, T.A.M. Toffolo, S.S. Brito, and M.J.F. Souza, A SA-ILS approach for the high school timetabling problem, Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), Son, Norway, August 2012
- Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes, HySST: hyper-heuristic search strategies and timetabling, Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), Son, Norway, August 2012



- Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, <http://www.it.usyd.edu.au/~jeff/hseval.cgi> (2010)
- B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. Parkes, L. Di Gaspero, R. Qu, and E. Burke, Setting the research agenda in automated timetabling: the Second International Timetabling Competition, *INFORMS Journal on Computing* 22, 120–130 (2010)
- Ben Paechter, Luca Maria Gambardella, and Olivia Rossi-Doria, The First International Timetabling Competition, <http://www.idsia.ch/Files/ttcomp2002/> (2002)
- Gerhard Post, Jeffrey H. Kingston, Samad Ahmadi, Sophia Daskalaki, Christos Gogos, Jari Kyngäs, Cimmo Nurmi, Haroldo Santos, Ben Rorije and Andrea Schaerf, An XML format for benchmarks in high school timetabling II, *Annals of Operations Research* (online), <http://10.1007/s10479-011-1012-2>, 2011
- Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, and David Ranson, An XML format for benchmarks in high school timetabling, *Annals of Operations Research* 194, 385–397, 2012
- Gerhard Post, Benchmarking project for High School Timetabling, <http://www.utwente.nl/ctit/hst/> (2011)
- Matias Sørensen, Simon Kristiansen, and Thomas R. Stidsen, International timetabling competition 2011: an adaptive large neighborhood search algorithm, *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*, Son, Norway, August 2012

---

# An Evolutionary Algorithm for High School Timetabling

Jonathan Domrös · Jörg Homberger

**Abstract** An Evolutionary Algorithm for high school timetabling problems, which is used for the Third International Timetabling Competition (ITC2011), is presented. The solver is based on two ideas: Firstly, an indirect representation of timetables is used. Each coded solution consists of a permutation of sub-events. Secondly, the evolutionary search is controlled by the population concept of the (1+1)-Evolution Strategy.

**Keywords** High school timetabling · International timetabling competition · Evolutionary algorithm

## 1 Introduction

Evolutionary Algorithms are metaheuristics which are often based on a coding of decision variables or solutions (e.g., Miettinen et al., 1999). They have been used successfully for (high) school timetabling so far (e.g., Burke and Newall, 1999; Bufe et al., 2001; Beligiannis et al., 2009; Raghavjee and Pillay, 2010). However, most approaches have been developed and evaluated for high school timetabling problems with a smaller number of constraints than the problems used for the ITC2011.

The high school timetabling problems of the ITC2011 are described in Post et al. (2012). In order to describe our algorithm, especially our coding, it is important to know, that the problems take up to three different kinds of problem decisions into account: (1) split-decisions (for a given event the number of sub-events and the duration of each sub-event have to be calculated); (2) time-decisions (for a sub-event a starting time has to be assigned); and (3) resource-decisions (for a sub-event one or more event resources have to be assigned).

## 2 Overview of the Evolutionary Algorithm

An overview of the developed Evolutionary Algorithm is shown in Fig. 1.

- 
- (1) **preprocessing**: calculate all feasible sub-events;
  - (2) **initialization**: calculate one coded timetable  $\mathbf{C}$  randomly on the basis of (1);
  - (3) **decoding**: construct a timetable  $T$  by decoding  $\mathbf{C}$ ;
  - (4) **evaluation**: calculate the infeasibility value and the objective value for  $T$ ;  
     WHILE (NOT a given computational time limit has been reached)
  - (5)     **mutation**: calculate a new coded timetable  $\mathbf{C}^*$  by mutation of  $\mathbf{C}$ ;
  - (6)     **decoding**: construct a new timetable  $T^*$  by decoding  $\mathbf{C}^*$ ;
  - (7)     **evaluation**: calculate the infeasibility value and the objective value for  $T^*$ ;
  - (8)     **replacement**: IF ( $T^*$  is better than  $T$ ) THEN ( $\mathbf{C} := \mathbf{C}^*$ ;  $T := T^*$ );
  - (9) **output**: the timetable  $T$ ;
- 

**Fig. 1** The Evolutionary Algorithm for High School Timetabling

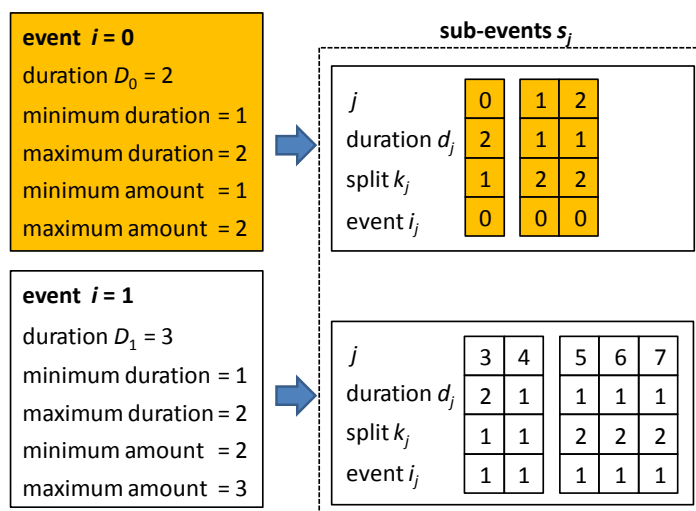
As shown in Fig. 1, the population concept of the (1+1)-Evolution Strategy, which is simply a randomized hill-climbing method, is used in order to control the search (e.g., Beyer and Schwefel, 2002; Mester and Bräysy, 2004). That means, in each iteration one solution or timetable  $T^*$  is calculated by mutation of the current best solution  $T$ . The latter is replaced by  $T^*$ , if  $T^*$  is better than the current best solution  $T$  (for the evaluation of timetables see Post et al., 2012). The mutation does not directly work on timetables. Instead, the mutation varies a coded version  $C$  of the current best timetable  $T$ . The coding of solutions and the steps of the algorithm are explained in the next sections.

### 3 Coding of solutions and preprocessing

Solutions are coded by a permutation of eligible sub-events or, more precisely, a permutation of the indices of eligible sub-events. The aim of preprocessing is to calculate the set of eligible sub-events, taking split events constraints, distribute split events constraints, and link events constraints into account. Therefore, for each given event  $i$  and its given duration  $D_i$  all feasible splits are calculated. A split of an event  $i$  is a division into one or more sub-events with individual durations. A split is feasible, if the split events constraints and the distribute split events constraints are fulfilled for  $i$ . The number of feasible splits of event  $i$  is denoted by  $n_i$ . Each sub-event  $s_j = (i_j, k_j, d_j)$  can be described by a triple, which consists of the corresponding event  $i_j$ , the corresponding split  $k_j$  ( $k_j = 1, \dots, n_i$ ), and a sub-event duration  $d_j$ . The total duration of all sub-events, which corresponds to the same split of an event  $i$  is equal to  $D_i$ . Fig. 2 describes the preprocessing procedure by an example. The example takes the split events constraints into account, i.e., a minimum and a maximum duration for sub-events, and a minimum and a maximum amount of sub-events have to be considered, when eligible sub-events are calculated.

Link events constraints could reduce the set of feasible splits. In case that two events  $i$  and  $l$  must be executed in parallel, the corresponding splits must be “compatible”, i.e., for each sub-event  $s_m$  of event  $i$  exists one sub-event  $s_n$  of event  $l$ , such that  $d_m = d_n$ . Thus, each feasible split of  $i$  ( $l$ ) with no compatible split of  $l$  ( $i$ ) is deleted from the set of feasible splits.

As mentioned, each coded solution is just a permutation of the indices of the calculated eligible sub-events. In our example of Fig. 2 each coded solution is a permutation of the indices 0, ..., 7.



**Fig. 2** Preprocessing procedure

## 4 Initialization and mutation

At the beginning of the search (step 2) one coded solution  $C^*$  is constructed randomly, i.e., a random permutation of the indices of the eligible sub-events is calculated. During mutation (step 5) this permutation is varied randomly, i.e., two indices are randomly chosen and then swapped. The swap-mutation is often suggested for permutation based Evolutionary Algorithms (e.g., Gottlieb, 2000).

## 5 Decoding

The decoding procedure aims to construct a feasible timetable  $T$  on the basis of a coded solution  $C$ . A timetable  $T$  consists of sub-events, which are extended by starting times and event resources. The decoding method is very complex, since it considers all constraints of the problem at hand in order to find a feasible solution. In the following, we just present some of the basic ideas of the procedure.

To construct a timetable  $T$ , three steps are executed within each iteration of the iterative decoding procedure: (i) One eligible sub-event  $s_j = (i_j, k_j, d_j)$  is selected; (ii) the split  $k_j$  is checked to be “practicable”; (iii) in the positive case ( $k_j$  is a practicable split), an insertion heuristic is executed in order to assign a starting time and all required event resources to  $s_j$ .

The order to select sub-events in step (i) is based on the order of indices in the coded solution  $C$ . If the index  $j$  of sub-event  $s_j$  is ordered more left than the index  $q$  of sub-event  $s_q$ ,  $s_j$  is selected before  $s_q$  and thus has a higher priority to be inserted into the timetable  $T$ .

A split  $k_j$  is defined as “practicable” in step (ii), if no sub-event of the corresponding event  $i_j$  has been inserted into  $T$  or if only sub-events of split  $k_j$  have been inserted into  $T$  so far. The check of practicability is necessary, since for each event only sub-events of the same split can be inserted into a timetable.

The insertion heuristic of step (iii) checks, if at least one assignment of a starting time and of event resources for  $s_j$  exists, such that all hard constraints are satisfied. In a positive case, the heuristic inserts  $s_j$  into  $T$ , i.e., assigns a starting time and event resources such that no hard constraint is injured (time and resource decisions are made). If  $s_j$  is the first sub-event of the event  $i_j$ , which is inserted

into  $T$ , the corresponding split  $k_j$  is selected for  $i_j$  (split-decision is made). In the case, that no such feasible assignment of time and resources exists, the sub-event  $s_j$  is not inserted into  $T$ .

There is one exception from the rule that in each iteration of the decoding procedure only one sub-event ( $s_j$ ) is selected and inserted into  $T$ . In case of link events constraints, where one or more events exist, which should be placed in parallel to  $i_j$ , sub-events of these events are also selected in step (i) and inserted together with  $s_j$  in step (iii).

It should be remarked, that sub-events of an event are not inserted at all, if the insertion would violate hard constraints. However, not inserting an event violates assign times constraints. The decoding method could be improved by executing an additional construction step (at the end of decoding) which inserts all these events, which were not inserted so far, in order to reduce the infeasibility value.

## 6 Conclusion

The developed Evolutionary Algorithm is suitable to solve the problems of the ITC2011. Our solver was a finalist in ITC2011. However, the solutions generated in round 1 are not a match to the existing best known or optimal solutions.

## References

- Beligiannis GN, Moschopoulos CN, Likothanassis SD (2009) A genetic algorithm approach to school timetabling. *Journal of the Operational Research Society* 60(1), 23–42
- Beyer HG, Schwefel HP (2002) Evolution strategies: a comprehensive introduction. *Journal Natural Computing* 1(1), 3–52
- Bufe M, Fischer T, Gubbels H, Hacker C, Hasprich O, Scheibel C, Weicker K, Weicker N, Wenig M, Wolfangel C (2001) Automated solution of a highly constrained school timetabling problem – preliminary results. In: *Proc. of the Evolutionary Computing Workshops on Applications of Evolutionary Computing*, Springer, pp 431–440
- Burke EK, Newall JP (1999) A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation* 3(1), 63–74
- Gottlieb J (2000) Permutation-based evolutionary algorithms for multidimensional knapsack problems. In: *Proceedings of the 2000 ACM Symposium on Applied Computing – Vol. 1*, ACM New York, NY, USA, pp 408–414
- Mester D, Bräysy O (2007) Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers and Operations Research* 34, 2964–2975
- Miettinen K, Mäkelä MM, Neittaanmäki P, Périaux J (eds), *Evolutionary algorithms in engineering and computer science*. Chichester et al., 1999
- Post G, Gaspero LD, Kingston JH, McCollum B, Schaerf A (2012) The third international timetabling competition. In: *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*, Son, Norway, August 2012
- Raghavjee R, Pillay N (2010) An informed genetic algorithm for the high school timetabling problem. In: *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pp 408–412

---

# International Timetabling Competition 2011: An Adaptive Large Neighborhood Search algorithm

Matias Sørensen · Simon Kristiansen ·  
Thomas R. Stidsen

**Keywords** Adaptive Large Neighborhood Search · High School Timetabling ·  
International Timetabling Competition 2011 · XHSTT

## 1 Introduction

An algorithm based on Adaptive Large Neighborhood Search (ALNS) for solving the generalized High School Timetabling problem in XHSTT-format (Post et al (2012a)) is presented. This algorithm was among the finalists of round 2 of the International Timetabling Competition 2011 (ITC2011). For problem description and results we refer to Post et al (2012b).

## 2 Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search was first developed as a metaheuristic for the class of Vehicle Routing Problems (Pisinger and Ropke (2005); Ropke and Pisinger (2006)). It has been applied for few other problem classes as well, including Project Scheduling (Muller (2009, 2010)), Lot-sizing (Muller and Spoorendonk (2011)), Optimal Statistic Median Problem (Katterbauer et al (2012)).

Recently we have developed a framework based on ALNS for solving combinatorial optimization problems (written in C# 4.0). This framework is part of the commercial product Lectio<sup>1</sup>, where it is used to solve various practical timetabling problems, see Kristiansen et al (2011); Sørensen and Stidsen (2012) and Kristiansen and Stidsen (2012).

The pseudo code for a general ALNS algorithm is given in Algorithm 1.

---

Matias Sørensen [mss@dtu.dk], Simon Kristiansen, Thomas R. Stidsen  
Operations Research, DTU Management Engineering  
Technical University of Denmark

<sup>1</sup> <http://www.lectio.dk>  
Cloud-based administration system for high schools. Developed by MaCom A/S, Vesterbro-  
gade 48 1., 1620 Copenhagen V, Denmark

---

**Algorithm 1** Adaptive Large Neighborhood Search
 

---

```

1: candidate solution  $x$ , remove-methods  $\Omega^-$ , insert-methods  $\Omega^+$ 
2:  $x_{\text{best}} = x$ 
3: while stop-criterion not met do
4:    $x' = x$ 
5:   RemoveStrategy: select  $q$  as some quantity to be removed
6:   AdaptiveStrategy: select remove-method  $r \in \Omega^-$  and insert-method  $i \in \Omega^+$ 
7:   remove requests from  $x'$  using  $r(q)$ 
8:   insert requests into  $x'$  using  $i$ 
9:   AdaptiveStrategy: update performance indicators
10:  if  $c(x') \leq c(x_{\text{best}})$  then
11:     $x_{\text{best}} = x'$ 
12:  end if
13:  AcceptStrategy: set candidate solution  $x$  to either  $x'$ ,  $x_{\text{best}}$  or  $x$  itself
14: end while
15: return  $x_{\text{best}}$ 

```

---

The main points of the algorithm are described below in general terms.

- In each iteration, a remove and insertion method is chosen and applied to the candidate solution. The combination of these methods defines the neighborhood of the algorithm, hence there exists  $|\Omega^-| \cdot |\Omega^+|$  different neighborhoods.
- **RemoveStrategy**: Governs the selection of  $q$ . This has major influence on how much computational time each iteration requires.
- **AdaptiveStrategy**: Responsible for selecting remove and insertion methods in each iteration, and updating their respective performance indicators of these method by some metric.
- **AcceptStrategy**: Determines which solution to use as candidate solution for next iteration. This could in principle be any known solution, but is usually selected as either the current candidate solution  $x$  itself, the newly produced solution  $x'$ , or the current best solution  $x_{\text{best}}$ .

### 3 Algorithm setup for ITC2011

Here we describe our implementation of a ALNS algorithm for the XHSTT format. The choice of ALNS strategies are briefly mentioned below. More details will be available in the full paper.

- **RemoveStrategy**: The remove and insertion methods deal with sub-events.  $q$  is defined as the sum of the duration of the sub-events which are removed from the solution. We select  $q$  as a random number, bounded by a percentage of the total duration of all instance events.
- **AdaptiveStrategy**: We have chosen a metric essentially based on two parameters for each method; The number of times the method was part of an iteration which yielded a better solution than the current one, and the

relative gap between the current solution and the resulting solution from applying the method.

- **AcceptStrategy:** An acceptance criteria borrowed from Simulated Annealing (SA) is used, with the following additional property: If no new best solution has been found in a number of iterations, the temperature is increased by a factor, and the candidate solution is set to the best known solution. The intention is to allow more diverse exploring of the area around the best known solution, in case the algorithm gets 'stuck'.

Let a *move* be a small perturbation on a solution. The following moves are used in this implementation: Move  $M_{se,t}$  denotes the assigning of sub-event  $se$  to time  $t$ .  $M_{r,er,se}$  denotes the assigning of resource  $r$  to event resource  $er$  on sub-event  $se$ . Furthermore we also implement the corresponding unassign-moves, denoted  $M_{se,t}^-$  and  $M_{r,er,se}^-$ , respectively.

Using these moves a total of 9 insertion methods (all more or less based on the greedy principle, e.g. regret heuristics (Potvin and Rousseau (1993); Sørensen and Stidsen (2012)), and 14 remove methods (all based on some element of relatedness and an element of randomness) are implemented. These methods are divided into three categories, based on what they (un-)assign: Only times, only resources, or both times and resources.

An example of a remove method is the following, which removes sub-events from non-preferred times: Given an XHSTT instance, and a solution  $S$  to this instance. Find all tuples  $\langle se, t \rangle$  of  $S$ , where sub-event  $se$  is assigned time  $t$ , and  $t$  is not a preferred time for sub-event  $se$  (see *Prefer times constraints*, Kingston (2010)). Let the set of these tuples be denoted  $U$ . Select randomly a subset of these tuples  $\bar{U} \subseteq U$  such that the sum of the duration of all sub-events of the tuples in  $\bar{U}$  equals  $q$ . Perform an unassign time move  $M_{se,t}^-$  for each of the tuples in  $\bar{U}$ .

An example of an insertion method is the following: Let  $\Delta(M) \in \mathbb{R}$  be the profit of performing move  $M$  on the solution at hand  $S$ . Select  $M_{\text{best}} = \arg \min_{se,t} (\Delta(M_{se,t}))$ , and if  $\Delta(M_{\text{best}}) \leq 0$ , apply  $M_{\text{best}}$  to  $S$  and repeat, otherwise stop. This is a greedy method which assigns times to sub-events, until no profitable move can be found.

In the full paper all insert/remove methods will be described in detail.

The final algorithm contains 9 free parameters, which were tuned for best performance using the *irace* package (see López-Ibáñez et al (2011); Birattari (2005)).

#### 4 Final remarks

This paper documents how Adaptive Large Neighborhood Search can be applied to problems in XHSTT format.

The proposed algorithm was applied to all instances in archive XHSTT-ITC2011, and showed competitive results in most cases (comparing to the best known solutions at that point in time).



ALNS has not been used much in the field of timetabling, but we see no reason to believe that ALNS should not perform well on other (related) problems in this field.

**Acknowledgements** Thank you goes to Michael Herold for fruitful discussions concerning ALNS strategies. Thank you to Manuel López-Ibáñez for help using the `irace` package. And finally thank you to David Pisinger for advice on ALNS implementation.

## References

- Birattari M (2005) The Problem of Tuning Metaheuristics as seen from a Machine Learning Perspective, vol 292 *Dissertations in Artificial Intelligence - Infix*, 1st edn. Springer
- Katterbauer K, Oguz C, Salman S (2012) Hybrid adaptive large neighborhood search for the optimal statistic median problem. *Computers & Operations Research* 39(11):2679 – 2687
- Kingston JH (2010) The hseval high school timetable evaluator. URL <http://it.usyd.edu.au/~jeff/hseval.cgi>
- Kristiansen S, Stidsen TR (2012) Adaptive large neighborhood search for student sectioning at danish high schools. In: *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*
- Kristiansen S, Sørensen M, Stidsen T, Herold M (2011) Adaptive large neighborhood search for the consultation timetabling problem, to appear
- López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M (2011) The irace package: Iterated racing for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, Université Libre de Bruxelles, IRIDIA, Av F. D. Roosevelt 50, CP 194/6 1050 Bruxelles, Belgium, <http://iridia.ulb.ac.be/irace>
- Muller L (2009) An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In: *MIC 2009: The VIII Metaheuristics International Conference*
- Muller L (2010) An adaptive large neighborhood search algorithm for the multi-mode resource-constrained project scheduling problem. Tech. rep., Department of Management Engineering, Technical University of Denmark Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
- Muller L, Spoorendonk S (2011) A hybrid adaptive large neighborhood search algorithm applied to a lot-sizing problem. *European Journal of Operational Research* Volume 218(Issue 3):614–623
- Pisinger D, Ropke S (2005) A general heuristic for vehicle routing problems. *Computers & Operations Research* 34:2403–2435
- Post G, Ahmadi S, Daskalaki S, Kingston J, Kyngas J, Nurmi C, Ranson D (2012a) An xml format for benchmarks in high school timetabling. *Annals of Operations Research* 194:385–397
- Post G, Gaspero LD, Kingston JH, McCollum B, Schaerf A (2012b) The third international timetabling competition. In: *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*, Son, Norway
- Potvin JY, Rousseau JM (1993) A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research* 66(3):331 – 340
- Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40:455–472
- Sørensen M, Stidsen TR (2012) High school timetabling: Modeling and solving a large number of cases in denmark. In: *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*

---

## A SA-ILS approach for the High School Timetabling Problem

Fonseca, G.H.G., Santos, H.G., Toffolo,  
T.A.M., Brito, S.S., Souza, M.J.F.

Received: date / Accepted: date

**Abstract** This work presents a heuristic approach proposed by one of the finalists of the Third International Timetabling Competition (ITC2011). The KHE school timetabling engine is used to generate an initial solution and then Simulated Annealing (SA) and Iterated Local Search (ILS) perform local search around this solution.

**Keywords** Simulated Annealing · Iterated Local Search · High School Timetabling Problem · Third International Timetabling Competition

### 1 Introduction

The diversity of School Timetabling models encountered around the world motivated the definition of an XML format to express different entities and constraints considered when building timetables[6]. The format evolved and a public repository[1] with a rich set of instances was built. To stimulate the research in this area, the Third International Timetabling Competition (ITC2011) occurred in 2012. This paper presents one of the finalists' solvers in this competition.

### 2 Solution Approach

Our approach uses the KHE school timetabling engine [2] to generate initial solutions and the metaheuristics Simulated Annealing and Iterated Local Search to perform local search around this solution. Since the constructive method is described in high level of details in[2], only the local search procedures will be detailed in the next sections.

---

Computing Department  
Federal University of Ouro Preto  
Ouro Preto, Minas Gerais, Brazil 35400-000

## 2.1 Local Search

Seven neighborhood structures were used in our local search approach:

- Event Swap (ES): two events  $e_1$  and  $e_2$  have their timeslots  $t_1$  and  $t_2$  swapped;
- Event Move (EM): an event  $e_1$  is moved from timeslot  $t_1$  to another timeslot  $t_2$ ;
- Event Block Move (EBM): like ES, but when moving adjacent events with different duration keeps these events adjacent;
- Resource Swap (RS): two events  $e_1$  and  $e_2$  have their assigned resources  $r_1$  and  $r_2$  swapped, resources  $r_1$  and  $r_2$  should play the same role (e.g. both have to be teachers);
- Resource Move (RM): an event  $e_1$  has his assigned resource  $r_1$  replaced by a new resource  $r_2$ .
- Permute Resources (PR): given a resource  $r_1$ , up to  $n_{PR}$  events assigned to  $r_1$  have their timeslots permuted; the events are chosen at random and the parameter  $n_{PR}$  is set to 7 so that all the permutations can be computed in a short amount of time.
- Kempe Move (KM): two times  $t_1$  and  $t_2$  are fixed and one seeks the best path at the bipartite conflict graph containing all events in  $t_1$  and  $t_2$ ; arcs are build from conflicting events which are in different timeslots and their cost is the cost of swapping the timeslots of these two events;

All local search methods can apply any move on the proposed neighborhoods (except PR, which is used only in the perturbation phase of ILS, returning the best different neighbor). If the instance requires assignment of resources (i.e. there exists at least one *AssignResourceConstraint* constraint), the kind of neighborhood is chosen based on the following probabilities: ES = 0.20, EM = 0.38, EBM = 0.10, RS = 0.20, RM = 0.10 and KM = 0.02, otherwise, the neighborhood RS and RM are not used and the odds are: ES = 0.40, EM = 0.38, EBS = 0.20 and KM = 0.02. These values were empirically adjusted.

### 2.1.1 Simulated Annealing Implementation

Proposed by [3], the metaheuristic Simulated Annealing is a probabilistic method based on an analogy to thermodynamics simulating the cooling of a set of heated atoms. This technique starts its search from any initial solution. The main procedure consists of a loop that randomly generates, at each iteration, one neighbor  $s'$  of the current solution  $s$ . Movements are probabilistically selected considering a temperature  $T$  and the cost variation of the movement  $\Delta$ .

The developed implementation of Simulated Annealing is described in Algorithm 1. Parameters used were  $\alpha = 0.97$ ,  $T_0 = 1$ ,  $SAMax = 10,000$  and  $SAreheats = 5$ . The method *selectNeighborhood()* just chooses a neighborhood structure according to the neighborhood probabilities previously defined.

**Algorithm 1:** Developed implementation of Simulated Annealing

---

**Input:**  $f(\cdot), N(\cdot), \alpha, S_{Amax}, T_0, S_{Areheats}, s, timeout$   
**Output:** Best solution  $s^*$  found.  
 $s^* \leftarrow s; IterT \leftarrow 0; T \leftarrow T_0; reheats \leftarrow 0;$   
**while**  $reheats < S_{Areheats}$  and  $elapsedTime < timeout$  **do**

**while**  $IterT < S_{Amax}$  **do**

$IterT \leftarrow IterT + 1;$   
 $k \leftarrow selectNeighborhood();$   
Generate a random neighbor  $s' \in N_k(s);$   
 $\Delta = f(s') - f(s);$   
**if**  $\Delta < 0$  **then**

$s \leftarrow s';$   
**if**  $f(s') < f(s^*)$  **then**  $s^* \leftarrow s';$

**else**

Take  $x \in [0, 1];$   
**if**  $x < e^{-\Delta/T}$  **then**  $s \leftarrow s';$

$T \leftarrow \alpha \times T;$   
 $IterT \leftarrow 0;$   
**if**  $T < 0.1$  **then**

$reheats \leftarrow reheats + 1;$   
 $T \leftarrow T_0$

### 2.1.2 Iterated Local Search

The method Iterated Local Search (ILS)[4] is based on the idea that a local search procedure can achieve better results by optimizing different solutions generated through disturbances on the local optimum solution.

Our ILS algorithm starts from an initial solution  $s_0$  obtained by the Simulated Annealing procedure and makes disturbances of size  $p_{size}$  under  $s_0$  followed by a descent method. A disturbance is the unconditional acceptance of a neighbor generated by neighborhoods PR or KM, both with 0.5 of probability.

The descent phase uses a Randomic Non Ascendent Method, which accepts only neighbors if they are better than or match the current solution. Tested moves are excluded from the neighborhood and return only when an improvement to the current best solution is reached. The local search phase ends when there is no remaining neighbor to be explored.

The local search produces a solution  $s'$  which will be accepted if it is better than the best solution  $s^*$  found. In such case, the disturbance size  $p_{size}$  gets back to the initial size  $p_0$ . If the iteration  $Iter$  reaches a limit  $Iter_{max}$ , the disturbance size is incremented. Yet if the disturbance size reaches a bound  $p_{max}$ , it goes back to the initial size  $p_0$ . Algorithm 2 presents the developed implementation of ILS. The considered parameters are  $ILS_{max} = 10,000$ ,  $p_0 = 1$ ,  $p_{max} = 10$  and  $MaxIter_p = 10$ .

**Algorithm 2:** Developed implementation of ILS

---

**Input:**  $f(\cdot), N(\cdot), ILS_{max}, p_0, p_{max}, MaxIter_p, s, timeout$   
**Output:** Best solution  $s^*$  found.  
 $s \leftarrow descentPhase(s); s^* \leftarrow s;$   
 $p_{size} \leftarrow p_0; Iter \leftarrow 0;$   
**for**  $i \leftarrow 0$  **until**  $ILS_{max}$  **do**  
    **if**  $elapsedTime \leq timeout$  **then**  
        **for**  $j \leftarrow 0$  **until**  $p_{size}$  **do**  
             $s \leftarrow s_p \in N(s);$   
             $s' \leftarrow descentPhase(s);$   
            **if**  $f(s') < f(s^*)$  **then**  
                 $s \leftarrow s'; s^* \leftarrow s';$   
                 $Iter \leftarrow 0; p_{size} \leftarrow p_0;$   
            **else**  
                 $s \leftarrow s^*;$   
                 $Iter \leftarrow Iter + 1;$   
            **if**  $Iter = MaxIter$  **then**  
                 $p_{size} \leftarrow p_{size} + p_0;$   
                **if**  $p_{size} \geq p_{max}$  **then**  $p_{size} \leftarrow p_0;$   
    **return**  $s^*;$

---

### 3 Concluding Remarks

This paper presented a SA-ILS approach proposed by one the competition finalists of the Third ITC2011. Even though final results will only appear in the organizer's paper [5] we are confident that our method produced good results: we are finalists of the competition and our method improved several best known solutions for the competition instance set. Possible improvements are the development of an augmented set of (larger) neighborhoods and a proper experimental study to fine tune parameter selection. This almost surely will improve these results.

### References

1. Benchmarking project for (high) school timetabling (2012). URL <http://www.utwente.nl/ctit/hstt/>
2. Kingston, J.H.: A software library for school timetabling (2012). Available at <http://sydney.edu.au/engineering/it/~jeff/khe/>, May 2012
3. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* **220**, 671–680 (1983)
4. Lourenco, H.R., Martin, O.C., Stutzle, T.: Iterated local search. In: F. Glover, G. Kochenberger (eds.) *Handbook of Metaheuristics*, chap. 11. Kluwer Academic Publishers, Boston (2003)
5. Post, G., Di Gaspero, L., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. In: *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling*
6. Post, G., Kingston, J., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., Schaerf, A.: XHSTT: an XML archive for high school timetabling problems in different countries. *Annals of Operations Research* pp. 1–7

---

# HySST: Hyper-heuristic Search Strategies and Timetabling

Ahmed Kheiri · Ender Özcan ·  
Andrew J. Parkes

Received: date / Accepted: date

## 1 Introduction

High school timetabling (HST) is a well-known real-world combinatorial optimisation problem. It requires the scheduling of events and resources, such as courses, classes of students, teachers, rooms and more within a fixed number of time slots subject to a set of constraints. In a standard fashion, constraints are separated into ‘*hard*’<sup>1</sup> and *soft*. The hard constraints must be satisfied in order to achieve *feasibility*, whereas the soft constraints represent preferences and a solution for a given problem; solutions are expected to satisfy all hard constraints and as many soft constraints as possible. The HST problem is known to be NP-complete [2] even in simplified forms. For a recent survey of HST see [4]. Also, see [5] for a description of the specific HST version studied here, and also of the third international timetabling competition, ITC2011. Briefly, the ITC2011 problem instances contain 15 types of constraints and a candidate solution is evaluated in terms of two components: feasibility and preferences. The evaluation function computes the weighted hard and soft constraint violations for a given solution as *infeasibility* and *objective* values, respectively. For the comparison of algorithms, a solution is considered to be better than another if it has a smaller infeasibility value, or an equal infeasibility value and a smaller objective value.

Our approach to solving the HST is based on development of a hyper-heuristic (see [1] for a recent survey) in order to intelligently control the application of a range of neighbourhood move operators. Hyper-heuristics explore the space of (meta-)heuristics, as opposed to directly searching the space of solutions, and generally, split into one of two types: *selection* hyper-heuristics that select between existing low-level heuristics, and *generation*

---

University of Nottingham, School of Computer Science  
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK  
{ axk, exo, ajp }@cs.nott.ac.uk

<sup>1</sup> In the competition, such constraints are not strictly hard but are simply much more heavily penalised than the ‘soft’ constraints.

**Algorithm HySST\_Solver\_ITC2011**


---

```

S = create_initial_solution(); // takes tinit time
tremaining = toverall - tinit
Sbest = S;
while (tremaining notExceeded)
  Sstage_best = S;
  Sstage_start = S;
  while (tstage notExceeded) // stage entry
    LLH = SelectRandomlyFrom(MutationalHeuristics);
    S' = ApplyHeuristic(LLH, S);
    if (S' isBetterThan Sbest) then Sbest = S';
    if (S' isBetterThan Sstage_best) then Sstage_best = S';
    S = MoveAcceptance(S, S',  $\epsilon$ ); // can accept with factor  $(1 + \epsilon)$  worse
  if (Sstage_best isNotBetterThan Sstage_start) then
    S = ApplyHillClimbing(S); // uses the hill climbing heuristics
return Sbest;

```

---

**Fig. 1** Pseudocode of the implemented hyper-heuristic. Here  $\epsilon$  is a small (tunable) parameter that controls the level of worsening moves that are acceptable.

hyper-heuristics that build the decision procedures used with the heuristics. A selection hyper-heuristic generally includes both heuristic selection and move acceptance within a single point search framework (that is, without the use of populations of solutions): At each iteration, a candidate new solution is produced by selecting and applying a heuristic (neighbourhood operator) from a set of low level heuristics; A ‘move acceptance’ component then decides whether or not the candidate should replace the incumbent solution.

In this work, we develop and exploit a generalised selective hyper-heuristic (though envisage that future work could well exploit generative methods). We build on a previous study [3] that demonstrated the effectiveness of a generalised version of the iterated local search approach. Specifically, our hyper-heuristic uses a structured and staged application of multiple perturbative and hill climbing operators as opposed to simple selection from a single pool of all operators.

## 2 Hyper-heuristic Search for High School Timetabling

The search algorithm is implemented as a time contract algorithm and completes its execution in  $t_{overall}$  time. It consists of an initial construction phase followed by an extended improvement; see Figure 1 for the pseudocode. The initial construction of a complete solution is performed using the general solver implemented by Jeff Kingston in the KHE library<sup>2</sup>. The improvement phase uses the remaining time left ( $t_{remaining}$ ) after the construction of the initial solution which takes  $t_{init}$  time. Subsequently, a hyper-heuristic is used to control and mix a set of 11 low-level domain-specific heuristics and that are (mostly)

<sup>2</sup> <http://sydney.edu.au/engineering/it/~jeff/khe/>

fairly simple moves such as moving a task to a different resource, or swaps of events. They are divided into two sets; 9 mutational operators that do a randomised move, and 2 hill climbing operators that search their neighbourhoods for better solutions. Note that the construction phase often gives a solution in which hard constraints are violated, and so the improvement phase also needs to improve the hard constraints. The hyper-heuristic divides the search into stages where each stage takes a prefixed amount of time ( $t_{stage}$ ). In each stage, firstly and repeatedly, one of the 9 mutational heuristics is applied and the move is accepted if it is improving or is not worsening by more than a small amount. If no improvement is achieved during a stage, a hill climbing phase is applied using the 2 hill climbing heuristics. Unlike most of the mutational operators, these 2 hill-climbing heuristics are capable of making quite large changes to a solution. The hill climbing phase is itself also slightly non-standard. One of the operators is designed using neighbourhood structures based on ejection chains while the other operator is a type of first improvement hill climbing operator. Both hill climbing operators attempt to make moves which respect a particular constraint type while hoping to improve upon the other types of constraint violations but might have a net worsening of the objective, however, then such worsening moves are rejected. A hill climbing step is always non-worsening and so can be repeatedly applied in standard fashion until a local minimum is reached. A notable difference from standard methods (such as in memetic algorithms) is that we found that performance is better if the hill climbing is not applied if the mutational operators managed to improve the best solution. We suspect that excessive use of the hill climbing somehow gives over-optimised local solutions that afterwards lead to restricted movement within the search space.

In summary, we have developed and applied a hyper-heuristic to intelligently and effectively exploit a suite of neighbourhood move operators. Since the aim of hyper-heuristics is to separate adaptive search control from the details of the specific domain, we naturally also envisage our hyper-heuristic could be applied to other PATAT-related problems, such as to the timetabling of university examinations and courses.

## References

1. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* (to appear)
2. Even, S., Itai, A., Shamir, A.: On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal on Computing* **5**(4), 691–703 (1976)
3. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis* **12**(1), 3–23 (2008)
4. Pillay, N.: An overview of school timetabling research. In: *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, pp. 321–335 (2010)
5. Post, G., Gaspero, L.D., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. In: *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)* (2012)



# Index

## List of Authors

### A

Al-Betar, Mohammed Azmi  
Alefragis, Panayiotis  
Ângelo Machado Toffolo,  
Túlio  
Asaju Laaro, Bolaji  
Ásgeirsson, Eyjólfur Ingi  
Awadallah, Mohammed A.

### B

Bar, Doron  
Bellenguez-Morineau, Odile  
Boeres, Maria  
Bolaji, Asaju La'Aro  
Brac De La Perriere, Louise  
Burke, Edmund K.

### C

Capua, Renatha  
Carpio, Martin  
Castillo-Salazar, J. Arturo  
Ceschia, Sara  
Chiarandini, Marco  
Cohen-Zamir, Ilana  
Conniss, Richard  
Curtois, Tim

### D

De Causmaecker, Patrick  
Di Gaspero, Luca  
Domrös, Jonathan  
Dwyer, Rick

### E

Elhag, Anas

### F

Fagerberg, Rolf

### G

Gambini Santos, Haroldo  
Goerigk, Marc  
Gogos, Christos  
Gomes, Rafael  
Gonzalez-Rubio, Ruben  
Gualandi, Stefano  
Gukhool, Balkrishna Sharma  
Gunawan, Aldy  
Günther, Maik

### H

Hamilton-Bryce, Ryan  
Henrique Godim Da  
Fonseca, George  
Hombberger, Jörg  
Housos, Efthymios  
Hurink, Johann

### J

Jablonski, Stefan  
Jamilson Freitas Souza,  
Marccone  
Jouglet, Antoine

### K

Kendall, Graham  
Khader, Ahamad Tajudin  
Kheiri, Ahmed  
Kingston, Dr. Jeffrey H.  
Kloster, Oddvar  
Kristiansen, Simon  
Kyngäs, Jari  
Kyngäs, Nico

### L

Lamorgese, Leonardo  
Landa-Silva, Dario  
Lapègue, Tanguy  
Larsen, Jesper  
Lau, Hoong Chuin  
Lusby, Richard

### M

Mannino, Carlo  
Marecek, Jakub  
Martin, Simon  
Martins, Simone  
McCollum, Barry  
McMullan, Paul  
Mühlenthaler, Moritz  
Müller, Tomáš

### N

Nace, Alexandre  
Nace, Dritan  
Nissen, Volker  
Nu, Tha  
Nurmi, Kimmo

### O

Ouelhadj, Djamila  
Outteryck, Christophe  
Özcan, Ender

### P

Parkes, Andrew J.  
Pesant, Gilles  
Petrovic, Sanja  
Pillay, Nelishia  
Post, Gerhard  
Prot, Damien  
Puga, Héctor J.

**Q**

Qu, Rong

**R**

Range, Troels Martin

Rangel, Maria

Ribas, Sabir

Ribeiro, Celso

Riise, Atle

Rocha, Wallace

Rudová, Hana

**S**

Salassa, Fabio

Santos, Haroldo Gambini

Schaerf, Andrea

Schutten, Marco

Shah, Viral

Sigurardóttir, Guríur Lilla

Smet, Pieter

Soria-Alcaraz, Jorge Alberto

Sotelo-Figueroa, Marco

Souza Brito, Samuel

Stidsen, Thomas K.

Swan, Jerry

Sørensen, Matias

**T**

Tao, Baiyun

Thomas, J. Joshua

Toffolo, Túlio

**U**

Uijland, Suzanne

Urrutia, Sebastián

**V**

Valouxis, Christos

van der Veen, Egbert

van Veldhoven, Sophie

Vancroonenburg, Wim

Vanden Berghe, Greet

**W**

Wanka, Rolf

Westphal, Stephan

Wright, Mike

**Z**

Zeising, Michael

Zhu, Chun Bao



# **PATAT 2012**

9th International Conference on the Practice and Theory of Automated Timetabling  
Son, Norway, Tues. 28<sup>th</sup> - Fri. 31<sup>st</sup> August 2012