Edmund Burke
Michael Trick (Eds.)

LNCS 3616

Practice and Theory of Automated Timetabling V

5th International Conference, PATAT 2004
Pittsburgh, PA, USA, August 2004
Revised Selected Papers

Springer

Edmund Burke Michael Trick (Eds.)

Practice and Theory of Automated Timetabling V

5th International Conference, PATAT 2004
Pittsburgh, PA, USA, August 18-20, 2004
Revised Selected Papers

Springer

Volume Editors

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

Michael Trick
Carnegie Mellon University
Tepper School of Business
Pittsburgh, PA 15213, USA
E-mail: trick@cmu.edu

Library of Congress Control Number: 2005936696

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

ISSN 0302-9743
ISBN-10 3-540-30705-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-30705-1 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11593577 06/3142 5 4 3 2 1 0

Preface

This volume contains a selection of papers from the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004) held in Pittsburgh, USA, August 18–20, 2004. Indeed, as we write this preface, in the Summer of 2005, we note that we are about one month away from the tenth anniversary of the very first PATAT conference in Edinburgh. Since those very early days, the conference series has gone from strength to strength and this volume represents the latest in a series of five rigorously refereed volumes which showcase a broad spectrum of ground-breaking timetabling research across a very wide range of timetabling problems and applications.

Timetabling is an area that unites a number of disparate fields and which cuts across a number of diverse academic disciplines. While the most obvious instances of timetabling occur in educational institutions, timetabling also appears in sports applications, transportation planning, project scheduling, and many other fields. Viewing timetabling as a unifying theme enables researchers from these various areas to learn from each other and to extend their own research and practice in new and innovative ways. This volume continues the trend of the conference series to extend the definition of timetabling beyond its educational roots. In this volume, seven of the 19 papers involve domains other than education. Of course, educational timetabling remains at the core of timetabling research, and the papers in this volume represent the full range of this area including exam timetabling, room scheduling, and class rostering.

There are a number of particularly interesting aspects to the research presented in this volume. First, the variety of techniques being used to address these problems is striking. In this book, there are papers exploring optimization, constraint programming, evolutionary algorithms, tabu search, fuzzy approaches, and many other exact and heuristic methodologies. In many ways, timetabling is an ideal testbed for algorithmic approaches. The strength of timetabling in this regard revolves around a number of characteristics. First, timetabling problems are difficult, even for small instances. It is not necessary to have 10,000 students and 1,000 courses to lead to hard instances: even problems one-hundredth the size can be difficult. But it is exactly problems of that size that are of practical interest. Problems of practical interest are neither too large to be possibly solved, nor too small to be trivial. They are "just right": challenging, but possible. Furthermore, there are a lot of data available, and much of those data are available to academics. Finally, there are a number of different problem types available, allowing for a rich field of problems to be addressed. Taken together, these characteristics make timetabling an ideal domain for research on algorithms, and this volume demonstrates this richness through the variety of novel timetabling approaches that are explored and discussed.

Second, it is important to note how grounded in practice these papers are. Most of the papers begin with a real-world problem to solve. It is this interplay between real practice and theory that gives timetabling its richness. These papers are generally not about theoretical issues but are based on the need to create real timetables. This gives an immediacy to this work that is uncommon in much research.

The downside of this real-world aspect is a lack of standardization, leading to many papers solving only slightly different problems. The third interesting aspect of this volume is a growing interest in standardizing problem definitions and creating robust, flexible definitions of general timetabling problems. This trend is most obvious in the “General Issues” papers, but occurs in many other papers in the volume. While grounding the work in practice, there is a growing interest in generalizations.

Overall, we think this volume shows timetabling as a broad, important field with a rich set of practical models, and a robust and growing set of solution approaches. We thank the authors for their contributions, and are confident in the continuing success of the PATAT conference series.

Conference Series

The Meeting in Pittsburgh was the fifth in the PATAT series of international conferences. The first four conferences were held in Edinburgh (1995), Toronto (1997), Constance (2000), and Gent (2002). Selected papers from these four conferences appeared in the Springer *Lecture Notes in Computer Science* series. The full references are:

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

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

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

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

The sixth conference will be held in Brno, Czech Republic, August/September 2006. See http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml for information on the conference series.

Acknowledgements

We are very grateful to a large number of people for the success of the Pittsburgh conference and for their efforts in helping to put together this volume. We would like to acknowledge the financial support from the Tepper School of Business, Carnegie Mellon; the Carnegie Bosch Institute, Carnegie Mellon; and the Aladdin Center, Carnegie Mellon. Their generosity helped to give the conference the special atmosphere that made it such a memorable occasion. A particular thank you also goes to Cathy Burstein, who was invaluable in handling the local organization and registration, and the program could not have occurred without her efforts.

The papers that appear in this volume were carefully and thoroughly refereed. Many thanks go to the members of the Programme Committee who spent a significant amount of time ensuring the quality of the conference program itself and, particularly, of the selected papers that appear in this volume. Their hard work plays a major role in ensuring the success and high standards that have come to characterize the conference. We are also grateful to the staff at Springer for their help and encouragement and to Jan van Leeuwen, who, as an editor of the *Lecture Notes in Computer Science* series, has always given us valuable support and advice since the very beginning of the conference series back in 1995.

We would like to offer a very special thank you to Piers Maddox, our copy editor. The very high formatting and typesetting standards of this volume are entirely due to him. Special thanks should also go to Emma-Jayne Dann for all her hard work in supporting the administration that underpinned the editorial process for this book.

We are, of course, also very grateful to the authors and delegates at the conference who contributed so much towards making it such an enjoyable event. Finally, we would like to thank all the people on the Steering Committee for their hard work in organizing the entire series of PATAT conferences.

We are looking forward to the next conference and to seeing you in Brno in the Summer of 2006.

July 2005

Edmund Burke
Michael Trick

5th International Conference on the Practice and Theory of Automated Timetabling Programme Committee

Edmund Burke (Co-chair):	University of Nottingham, UK
Michael Trick (Co-chair):	Carnegie Mellon University, USA

Jonathan Bard	University of Texas, USA
Viktor Bardadym	Noveon Inc., Belgium
Cynthia Barnhart	MIT, USA
James Bean	University of Michigan, USA
Patrice Boizumault	University of Caen, France
Peter Brucker	University of Osnabrueck, Germany
Michael Carter	University of Toronto, Canada
Peter Cowling	University of Bradford, UK
Patrick De Causmaecker	KaHo St.-Lieven, Gent, Belgium
Kathryn Dowsland	Gower Optimal Algorithms Ltd.
Andreas Drexl	University of Kiel, Germany
Moshe Dror	University of Arizona, USA
Wilhelm Erben	FH Konstanz - University of Applied Sciences, Germany
Jacques A. Ferland	University of Montreal, Canada
Michel Gendreau	Centre de Recherche sur les Transports, Montreal, Canada
Alain Hertz	École Polytechnique de Montréal, Canada
Jeffrey Kingston	University of Sydney, Australia
Raymond Kwan	University of Leeds, UK
Gilbert Laporte	Universite de Montreal, Canada
Vahid Lotfi	University of Michigan-Flint, USA
Anuj Mehrotra	University of Miami, USA
Amnon Meisels	Ben-Gurion University, Beer-Sheva, Israel
George Nemhauser	Georgia Tech, USA
Thiruthlall Nepal	Durban Institute of Technology, South Africa
James Newall	eventMap Ltd., UK
Ben Paechter	Napier University, Edinburgh, UK
Gilles Pesant	École Polytechnique de Montréal, Canada
Sanja Petrovic	University of Nottingham, UK
Jean-Yves Potvin	Université de Montréal, Canada
Hana Rudova	Masaryk University, Czech Republic
Andrea Schaerf	Università di Udine, Italy
Jan Schreuder	University of Twente, The Netherlands

Stephen Smith	Carnegie Mellon University, USA
Jonathan Thompson	Cardiff University, UK
Paolo Toth	University of Bologna, Italy
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

5th International Conference on the Practice and Theory of Automated Timetabling Organizers

Michael Trick:	Carnegie Mellon University, USA
Cathy Burstein:	Carnegie Mellon University, USA

International Series of Conferences on the Practice and Theory of Automated Timetabling Steering Committee

Edmund Burke (Chair)	University of Nottingham, UK
Ben Paechter (Treasurer)	Napier University, UK
Victor Bardadym	Noveon Inc., Belgium
Patrick De Causmaecker	KaHo St.-Lieven, Gent, Belgium
Wilhelm Erben	FH Konstanz, University of Applied Sciences, Germany
Jeffrey Kingston	University of Sydney, Australia
Amnon Meisels	Ben-Gurion University, Beer-Sheva, Israel
Michael Trick	Carnegie Mellon University, USA
Dominque de Werra	EPF-Lausanne, Switzerland
George White	University of Ottawa, Canada

Table of Contents

General Issues

Transport Timetabling

University Course Timetabling

School Timetabling

Project Scheduling

Examination Timetabling

General Issues

Learning User Preferences in Distributed Calendar Scheduling

Jean Oh and Stephen F. Smith

School of Computer Science, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh,
PA 15213, USA
{jeanoh, sfs}@cs.cmu.edu

Abstract. Within the field of software agents, there has been increasing interest in automating the process of calendar scheduling in recent years. Calendar (or meeting) scheduling is an example of a timetabling domain that is most naturally formulated and solved as a continuous, distributed problem. Fundamentally, it involves reconciliation of a given user's scheduling preferences with those of others that the user needs to meet with, and hence techniques for eliciting and reasoning about a user's preferences are crucial to finding good solutions. In this paper, we present work aimed at learning a user's time preference for scheduling a meeting. We adopt a passive machine learning approach that observes the user engaging in a series of meeting scheduling episodes with other meeting participants and infers the user's true preference model from accumulated data. After describing our basic modeling assumptions and approach to learning user preferences, we report the results obtained in an initial set of proof of principle experiments. In these experiments, we use a set of automated CMRADAR calendar scheduling agents to simulate meeting scheduling among a set of users, and use information generated during these interactions as training data for each user's learner. The learned model of a given user is then evaluated with respect to how well it satisfies that user's true preference model on a separate set of meeting scheduling tasks. The results show that each learned model is statistically indistinguishable from the true model in their performance with strong confidence, and that the learned model is also significantly better than a random choice model.

1 Introduction

One vision of research in the field of intelligent software agents is the realization of personal computer assistants. A personal computer assistant is a software agent that is integrated into a user's computing environment and pro-actively accomplishes various tasks in support of high-level user goals. Like a human assistant, such a personal computer assistant would do such things as process email, schedule meetings, service information requests, organize events, and so on; autonomously interacting with other personal computer assistants as necessary to carry out a given task and in each case recognizing if and when it is

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 3–16, 2005.

appropriate to engage the user in the process. One fundamental aspect of a personal computer assistant is that it is enduring and self-improving. It is expected to persist indefinitely, and learn over time to make decisions that better reflect user constraints and preferences.

Toward this goal of creating personal computer assistants, there has been increasing interest in automating the process of scheduling meetings and managing user calendars. Calendar scheduling can be seen as a kind of timetabling problem—the objective is to assign time slots to meetings in such a way that the constraints and preferences of meeting requests and prospective attendees are best satisfied. However, the problem of calendar scheduling differs from typical timetabling problems in a couple of important respects:

- *Continuous, dynamic problem.* Calendar scheduling is an ongoing endeavor. At any point in time, there are some number of meetings booked, and new requests are serviced continuously in an incremental manner. It is generally desirable to maintain stability in assignments over time, although invariably it will be necessary to bump previously scheduled meetings, and the busier individuals are, the more frequently such tradeoffs will need to be considered.
- *Distributed decision-making.* Although one can consider centralized approaches to the calendar scheduling problem, this requires all individuals involved to share their calendars and this is not a realistic assumption in many circumstances. In these cases, protocols for negotiating time slots that are mutually acceptable to prospective attendees must be devised. Note that in some situations it may be necessary to settle on a subset of attendees, and/or coordinate with additional resource brokers (e.g., room booking agents).

Like other timetabling domains, calendar scheduling preferences will vary from user to user, and one strong prerequisite of any calendar scheduling solution is an ability to incorporate user-specific preferences. Preferences can range from simple static time of day (or day of week) preferences, to more complex dynamic preferences (such as scheduling meetings back-to-back or retaining free time in proximity to important deadlines), to preferences of which meeting(s) to bump in over-constrained situations. Typical timetabling solutions require users to directly specify their preferences as input to the problem solving process. However, the fact that calendar scheduling is an ongoing, continuous process suggests the possibility of automatically acquiring this knowledge over time through observation of meeting scheduling episodes.

Our recent research in the calendar scheduling domain has led to development of CMRADAR, a distributed calendar scheduling system [9]. Each CMRADAR scheduling agent accepts requests for meetings from its user, and interacts autonomously with the CMRADAR agents of other users to determine and confirm a mutually agreeable meeting time. If the action of scheduling a given meeting pre-empts a previously scheduled meeting, then affected CMRADAR agents coordinate to reschedule the bumped meeting. CMRADAR meeting scheduling protocols support a range of negotiation strategies, allowing the amount of information exchanged (e.g., number of options, preference values) and the assumptions made about organizational structure to be varied. Scheduling options are proposed and evaluated by CMRADAR agents based on how well they satisfy underlying user preferences. CMRADAR uses a quantitative preference representation, (i.e., a user preference is specified as a utility function), and assigns a value indicative of degree of satisfaction to a given scheduling option. Since such a specification of preference is somewhat unnatural for the user whom a given CMRADAR agent is assisting, we consider the possibility of automating the acquisition of user preferences. Another advantage of learning preferences automatically is that statistical observations can reveal certain meaningful patterns that are often missed by the human users.

Our general hypothesis is that it is possible for a software agent to learn a user's meeting time scheduling preferences by observing the user engage in a series of meeting scheduling episodes with other meeting participants. In this paper we describe an initial proof of principle experiment. We make specific assumptions about the types of information that can be observed during a meeting scheduling episode and the organizational setting in which meeting scheduling takes place, and with these assumptions we investigate an approach to learning the user's true preference model. In particular, we assume that a learning software agent has access to the following information when observing the user schedule meetings:

1. the user's current calendar,
2. incoming and outgoing meeting requests (initiator, proposed time slots),
3. user replies (accept or refuse) and
4. confirmed meeting time slots.

We further assume that meeting scheduling takes place within a hierarchical organization, that the learning agent has knowledge of the respective ranks of various meeting participants in the organization, and that users in the organization use a common negotiation strategy when scheduling meetings that favors the preferences of higher ranked individuals. Our specific hypothesis is that under these assumptions, accumulation of the above meeting information over some number of user scheduling episodes is sufficient to enable the agent to learn the user's true meeting time preference. To test this hypothesis we use a set of CMRADAR scheduling agents to simulate meeting scheduling under the above assumptions and generate training data for the learning agent. We then evaluate the ability of the learning agent to learn the true preference model of a given CMRADAR agent.

2 Related Work

There has been growing interest in creating software agents that can assist the users with daily routine tasks. In the particular problem of calendar scheduling there exist several commercial software calendar programs that support some form of basic meeting scheduling protocol. For instance, Microsoft Outlook Exchange Server provides capabilities such as finding intersections of free time slots of all attendees, and sending out meeting requests via automatically generated email messages, etc. Most commercial software rely on the protocol in which a centralized server has access to every individual calendar. In general meeting scheduling often takes place in more distributed settings where the members of an organization are not obliged to use a specific calendar software but elect to use their own choice of calendar programs. More fundamentally, these tools are limited in the fact that the user's scheduling preferences and habits are ignored.

Sen et al. [10] applied voting theory to their distributed scheduling agent system which tries to compromise in the event of conflicting user preferences during the negotiation process. The preferences are associated with utility values similar to our approach, but the user is responsible for specifying the quantitative data manually. Calendar Apprentice (CAP) [5], [8] used the decision tree learning method to learn user preference rules. Blum [2] improved CAP's performance by applying Winnow and Weighted-Majority based algorithms. CAP suggests specific values for the attributes of meeting such as duration and time slot. For example, the time of day preference rule suggests a time slot for a given type of meeting. If the suggested time slot is already taken by another meeting the closest available time slot is suggested. On the other hand, CMRADAR learns a utility function to evaluate different alternatives. PCalM [1] is another system that learns an evaluation function, in this case using large margin method [6] and Naive Bayesian approach with additional active learning strategy. However, no experimental results have been reported with this system in the literature.

3 Basic Modeling Assumptions

For purposes of this paper, we adopt the following basic modeling assumptions:

- We define a calendar to be a sequence of time slots of equal duration over some horizon. Let C_u be the calendar of user u, and $C_u(t)$ refer to time slot t of C_u.
- A meeting request $Req_{i,A,T}$ is initiated by an initiator i, and designates a set of attendees A and a set of one or more proposed time slots T.
- A reply or response to meeting request r by user u is designated $Resp_{u,r}$ and specifies a value of either *accept* or *refuse* for each time slot $t \in T$.
- A Scheduled Meeting $M_{i,A,t}$ similarly designates an Initiator i, a set of attendees A and a specific time slot t. For all $u \in A$, $C_u(t) = M_{i,A,t}$.
- A given time slot $C_u(t)$ of user u's calendar will either contain a scheduled meeting M or is *available*. Only one meeting may be scheduled at a given time slot. Hence for any new meeting M' to be scheduled at a given time slot t, either $C_u(t) = \textit{available}$ or the currently scheduled meeting $M_{I,A,t} : u \in A$ must be bumped.
- A *static* user meeting time preference model is expressed as a utility curve over some sequence of time slots. A static time of day (TOD) preference is a utility curve over a sequence of TOD time slots (e.g., the sequence of slots 7AM, 8AM, ..., 6PM). A time of week (TOW) preference model would be specified similarly (albeit using a larger sequence of TOW time slots). The value associated with a given time slot t in a preference utility curve

ranges from 1.0 (most preferred) to -1.0 (most negatively preferred), with 0 designating a tolerable lower bound on acceptability in peer to peer negotiation circumstances. Let $\mathit{Pref}_u(t)$ designate the preference value for time slot t by user u. When scheduling a meeting, Pref_u determines the relative desirability of different *available* time slots.

- We assume a hierarchical organization with n levels. A given individual i in the organization has a rank R_i, where $1 \leq R_i \leq n$. Individuals with higher rank reside at higher levels in the organization.

From the standpoint of the learning agent, the goal is to observe user u in the process of scheduling meetings and to acquire u's preference curve Pref_u. We assume that the learner sees each meeting request $\mathit{Req}_{i,A,T}$ involving u, each $\mathit{Resp}_{u,r}$ involving u, and receives confirmation of each scheduled meeting $M_{i,A,t}$ involving u. The learner also has access to u's current calendar at all times. Finally, the learner has knowledge of the ranks of all individuals in the organization, and assumes that all individuals use a common strategy for negotiating meeting times in which the preferences of higher ranked individuals are favored.

4 Approach

We take a statistical approach to learning a static TOD preference curve for a given user from observed meeting scheduling data. During a series of meeting scheduling episodes the learning agent observes the user's actions relative to the set of time slots in C_u. As meeting scheduling proceeds, the user proposes various time slots to initiate new meetings, accepts or refuses the time slots proposed by other users, and receives confirmations of mutually agreed upon meeting times. Conceptually, our approach views such actions and results as *noisy* examples of the user's underlying preference model (both positive and negative). In order to turn these observed data into a meaningful characterization of the user's time of day preference, we map the collected set of scheduling actions and results into "votes" for each time slot. The key point of our approach concerns how to weight these votes to minimize the noise.

In more detail, the following information is collected as the user engages in meeting scheduling:

- TOD time slots proposed by the user when initiating a meeting. In this case, proposed time slots provide active positive evidence of the user's true preference. Accordingly, we define $InitPropCt_u(t)$ to accumulate this positive evidence for different TOD time slots (e.g., 7AM, 8AM). The potential obscuring factor (or source of noise) in these data, however, is the density of C_u; if the most preferred time slots are already occupied, then less preferred time slots will necessarily be proposed. Taking this fact into account, each time the user proposes a given TOD time slot t when initiating a meeting, the following computation is performed:

$$\mathit{InitPropCt}_u(t) = \mathit{InitPropCt}_u(t) + (1 - \mathit{Density}_{C_u})$$

where

$$Density_{C_u} = \frac{OccupiedSlots_{C_u}}{TotalSlots_{C_u}} .$$

In other words, the evidence for a given proposed TOD time slot is discounted by the current density of C_u.

- TOD time slots that are available but refused by the user when responding to a meeting request. In this case, the user's response provides active negative evidence for the time slot(s) in question. We define $RefusedCt_u(t)$ to accumulate this negative evidence for various time slots. Each time the user refuses a proposed TOD time slot t that is actually available in C_u, the following computation is performed:

$$RefusedCt_u(t) = RefusedCt_u(t) + 1 .$$

- TOD Time slots of confirmed meeting times. These time slots also can provide passive positive evidence of the user's true preference (since the user has agreed to this time slot). As in the case of those time slots proposed by the user when initiating a meeting, $Density_{C_u}$ can be an obscuring factor and must be discounted. However, here there is also a second source of noise relating to the relative ranks of meeting attendees in the organization. Taking into account the fact that all users employ a common negotiation protocol that favors higher ranked individuals, we assume that the user will tend to reveal more truthful preferences when negotiating with lower ranked individuals. To account for this, evidence relating to confirmed meeting times is differentiated by the rank of the meeting initiator. Specifically, we define a matrix $ConfirmedCt_u(r,t)$, and each time a TOD time slot t proposed by an individual of rank r is confirmed as a meeting time, the following update is performed:

$$ConfirmedCt_u(r,t) = ConfirmedCt_u(r,t) + (1 - Density_{C_u}) .$$

Using the above computations, we collect "votes" for each TOD time slot. We then use the weighted k-nearest neighbor (KNN) algorithm [4] to consolidate this data and smooth the curve. KNN was initially proposed by Fix and Hodges [7]. It is a popular statistical approach which has been used heavily in the pattern recognition research and also in the text categorization. The basic idea here is to predict the utility value for a given TOD time slot using k similar data points in the training set. Here similarity is defined as a combination of both distance between TOD time slots and the distance between a meeting initiator's rank and the user's rank in the organization. The influence of a given data point on another is discounted as a function of its distance.

In more detail, the learned user preference model is computed according to the following four-step procedure:

1. *Integrate TOD time slot values in $ConfirmedCt_u$.* Taking the user's rank R_u into account, weighted KNN is applied to average the values accumulated for

each TOD time slot (i.e., each column in the matrix). KNN is applied asymmetrically in this case using rank distance as a similarity metric. Specifically, values accumulated for meetings initiated by individuals of rank $R_i > R_u$ are increasingly discounted as rank distance increases, based on the above stated intuition that meetings initiated by higher ranked individuals give less information. On the other hand, values for meetings initiated by individuals of rank $R_i \leq R_u$ are given full weight, since the user's preference will dominate in this case. The result of this smoothing step is a flattened vector of Confirmed meeting votes, designated $\mathit{FlatConfirmedCt}_u$.

2. *Combine collected data.* Compose final "votes" for each TOD time slot as follows:

$$TS_u = w_1 \times \mathit{InitPropCt}_u + w_2 \times \mathit{FlatConfirmedCt}_u - w_3 \times \mathit{RefusedCt}_u \,.$$

3. *Smooth adjacent time slot data.* Weighted KNN is applied again, this time to the consolidated TOD time slot vector TS_u. Following the assumption that the actual (true) user preference will tend to be continuous, each TOD time slot value is averaged with the values of the k neighboring TOD time slots, discounted by TOD distance.
4. *Normalize final values.* Finally, the values in TS_u are normalized to the range $[-1, 1]$ to produce the learned preference utility curve.

In the experiment described below, the above procedure is invoked after observing a fixed number of meeting scheduling episodes. Note, however, that the algorithm could be applied to compute (and recompute) the utility curve dynamically as more data points are accumulated over time.

5 Evaluation

Our evaluation contrasts the performance of three preference models:

- *true model,* the preference model used to simulate meeting scheduling and generate the training data used by learner;
- *learned model,* the preference model generated by the learning agent;
- *random model,* a preference model that randomly assigns utility values to time slots.

Each of these models is evaluated with respect to the true model: i.e., how well each model approximates the scheduling outcomes produced by the true model. Each model is applied to schedule a common (new) sequence of meetings, and in each case the final resulting calendar is evaluated with respect to how well it satisfies the true user preference model. More precisely, the quality of the resulting schedule is determined as

$$Q = \sum_{m \in MTGS_u} \frac{\mathit{Pref}_u(\mathit{TimeSlot}(m))}{|MTGS_u|} \,,$$

where $MTGS_u$ is the set of meetings in C_u and $\mathit{TimeSlot}(m)$ is the time slot in which meeting m is scheduled.

6 Experimental Design

We use a set of CMRADAR agents to simulate meeting scheduling in a four person organization. In our experiment, this CMRADAR simulation serves two purposes. First, it is used to generate training data for learning a given user's preference model. Second, it is used to evaluate the performance of a learned user preference model during the test phase of the experiment. The simulation is configured as follows:

- Each CMRADAR "user" (agent) is given a unique preference curve. Experiments were run with two distinct sets of preference curves: (1) a *simple* configuration consisting of a combination of morning, afternoon, strictly morning, and strictly afternoon preferences, and (2) a *complex* configuration consisting of four randomly specified preference curves.
- We assume a three-tiered organizational structure with User A > User B > Users $C1, C2$.
- A common negotiation protocol that favors preferences of higher ranked individuals is utilized by all CMRADAR agents. Details of this negotiation protocol are given in the Appendix.

For purposes of the experiment user calendars are assumed to contain a total of 60 time slots per week; specifically a five-day work week with each work day containing 12 one-hour time slots from 7AM to 7PM. The target is to learn a daily TOD preference: i.e. a utility curve over the sequence of TOD time slots from 7AM to 7PM.

For the training (learning) phase of each experiment, 60 meeting requests to be scheduled over a two-week horizon were randomly generated. 50 meetings were designated for week one and 10 for week two to ensure that some number of scheduling decisions must be made in the context of high density calendars. Each generated meeting involved 2, 3 or 4 of the individuals in the organization and the initiator was randomly assigned. Starting with empty calendars for all agents, the CMRADAR system is used to automatically schedule these 60 meetings. All meeting request messages, meeting response messages and meeting confirmation messages together with the users' evolving calendars are used along the way as training data for the learning algorithm summarized in section 4. Note that CMRADAR is a distributed system and each CMRADAR agent represents a different user in the organization. Hence, there are four different preference learners operating in this experiment, each associated with a specific CMRADAR agent (or user). Of course each learner only has access to the information that is local to its user.

For the test phase of each experiment, 20 new meetings to be scheduled in a one week interval were randomly generated as before. Each of four learned models was evaluated separately, by substituting one learned model for the corresponding true model in the CMRADAR system, and then running the system along with the true preference models for other CMRADAR users. For each revised configuration, the sequence of 20 new meetings was scheduled, starting again with empty calendars for all agents. By limiting the test case to 20 meetings, we ensure calendars that in the worst case are only $\frac{1}{3}$ full, and hence scheduled times are likely to be more reflective of user preference.

As indicated above, separate experiments were run for both simple and complex configurations of true preference models. For each configuration, 10 replications of the above training and test procedure were run using different meeting sets. The test results obtained using each different data set were averaged together to determine overall performance of the learned model. For comparison, we also computed average results obtained with the true and random models across all data sets.

7 Results

We analyze the results of both experiments relative to two basic hypotheses:

1. The performance of the learned model is comparable to the performance of the true model (i.e., the two models produce results that are statistically indistinguishable).
2. The performance of the learned model is indistinguishable from the performance of the random model.

Table 1 shows the experimental results obtained for the simple preference model configuration. It shows, for each of the four users, the quality of the final schedule produced by the learned model during the test phase from the standpoint of how well it satisfies the true preference model of each respective user (designated $Q_{Learned}$). The quality of the final schedules produced by the true model (Q_{True}) and the random model (Q_{Random}) are also shown, as well as a p-test evaluation of our basic hypotheses.

For all users, we see that there is sufficient evidence to reject the hypothesis that $Q_{Learned}$ is indistinguishable from Q_{Random}. In other words, $Q_{Learned}$ performs significantly better than Q_{Random} for all users. We note that the difference between $Q_{Learned}$ and Q_{Random} is smaller for users at lower ranks in the organization than it is for users at higher ranks (and this can be seen for the difference between Q_{True} and Q_{Random} as well). This reflects the fact that a common negotiation strategy favoring the preferences of higher ranked individuals is used, which reduces the overall influence of the preferences of lower ranked individuals.

Alternatively, it is not possible to reject the hypothesis that $Q_{Learned}$ is indistinguishable from Q_{True} for any user. To provide evidence in support of this hypothesis [3], confidence intervals for each pair of models were also computed. These are shown in Table 2. We can see with 95% confidence that there is strong evidence for $Q_{True} = Q_{Learned}$ for all users except user A. In each case, the confidence interval is found to contain zero and to be very small relative to the two scores. In the case of user A, $Q_{Learned}$ is actually found to be significantly *higher* than Q_{True}. The fact that $Q_{Learned} > Q_{True}$ is due to the fact that the learned preference model is stricter than the true model, which leads to increased pressure toward more preferred time slots in the true preference

Table 1. Performance results for *Simple* preference model configuration

User	Q_{True}	$Q_{Learned}$	Q_{Random}	$Q_{True} = Q_{Learned}$ (p-value)	$Q_{Learned} = Q_{Random}$ (p-value)
A	0.882	0.889	0.200	Cannot reject $(0.712 > 0.05)$	Very strong reject $(0.0000 < 0.01)$
B	0.936	0.938	0.864	Cannot reject $(0.534 > 0.05)$	Very strong reject $(0.0068 < 0.01)$
C1	0.822	0.807	0.720	Cannot reject $(0.347 > 0.05)$	Strong reject $(0.020 < 0.5)$
C2	0.791	0.792	0.726	Cannot reject $(0.505 > 0.05)$	Weak reject $(0.061 < 0.1)$

Table 2. 95% Confidence intervals for *Simple* preference model configuration

User	$Q_{True} - Q_{Learned}$ interval	Width	$Q_{True} = Q_{Learned}$
A	$(-0.028768, -0.006232)$	0.022536	Cannot accept (but $Q_{Learned}$ is better)
B	$(-0.0070086, 0.0028886)$	0.0098971	Accept
C1	$(-0.00086878, 0.031358)$	0.032226	Accept
C2	$(-0.011393, 0.010319)$	0.021713	Accept

Table 3. Performance results for *Complex* preference model configuration

User	Q_{True}	$Q_{Learned}$	Q_{Random}	$Q_{True} = Q_{Learned}$ (p-value)	$Q_{Learned} = Q_{Random}$ (p-value)
A	0.826	0.792	0.533	Cannot reject $(0.191 > 0.05)$	Very strong reject $(0.0000 < 0.01)$
B	0.757	0.759	0.647	Cannot reject $(0.52 > 0.05)$	Very strong reject $(0.0069 < 0.01)$
C1	0.607	0.602	0.510	Cannot reject $(0.462 > 0.05)$	Strong reject $(0.032 < 0.5)$
C2	0.634	0.640	0.501	Cannot reject $(0.551 > 0.05)$	Very strong reject $(0.0025 < 0.01)$

model. We would expect these Q values to balance out with additional training data. Figure 1 shows the true and learned preference models for each user from one of the simple preference configuration runs to give a graphical sense of their correspondence.

Table 3 shows the experimental results for the complex preference model configuration, and Figure 2 shows plots of learned and true models for each user

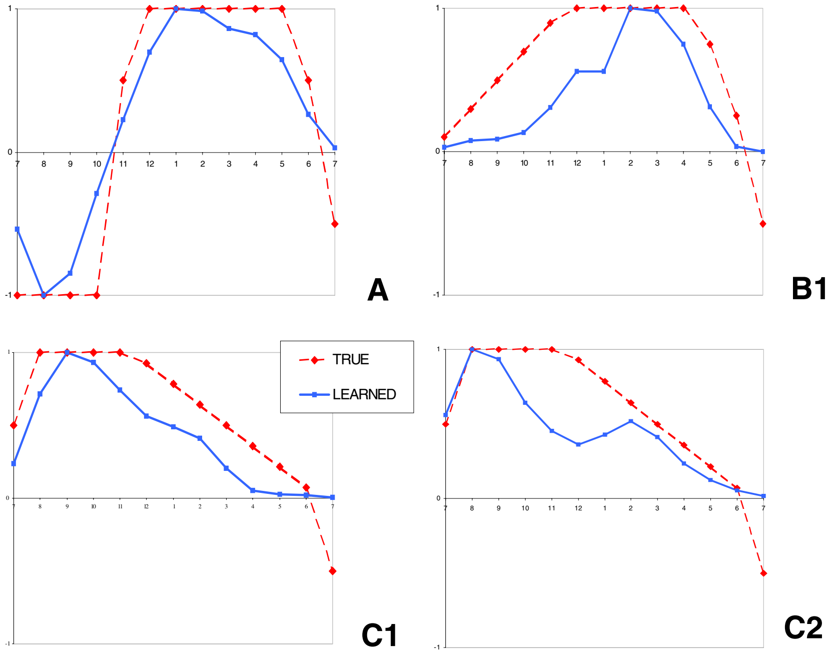


Fig. 1. True (dashed) and learned (solid) preference utility curves for *Simple* preference experiment

Table 4. 95% Confidence interval for *Complex* preference model configuration

User	Q_{True} - $Q_{Learned}$ interval	Width	$Q_{True} = Q_{Learned}$
A	(0.016814,0.052084)	0.03527	Cannot accept
B	(-0.01343,0.0091317)	0.022562	Accept
C1	(-0.011222,0.020452)	0.031674	Accept
C2	(-0.018553,0.0062601)	0.024814	Accept

from one of the experimental runs. As in the first experiment, the performance of the learned preference model is found to be better than that of the random model for all individuals in the organization (i.e., the hypothesis $Q_{Learned} = Q_{Random}$ is rejected in all cases). Again, it is the case that the hypothesis $Q_{Learned} = Q_{True}$ cannot be rejected for any user and computation of confidence intervals provides strong evidence that the hypothesis can be accepted. Table 4 shows the 95% confidence intervals for the complex model. For users B, C1 and C2, the results indicate that there is no significant difference between Q_{True} and $Q_{Learned}$. Not only does the confidence interval of the difference contain zero in each case, but

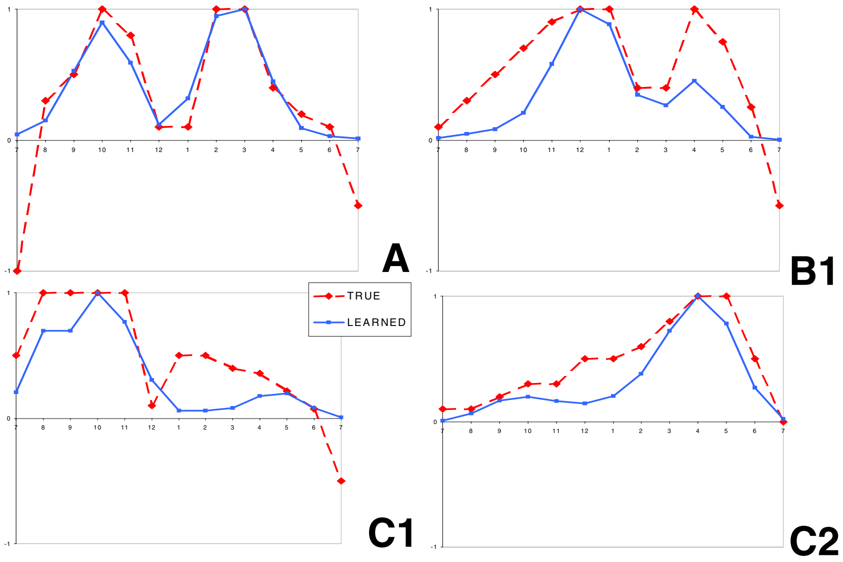


Fig. 2. True (dashed) and learned (solid) preference utility curves for the *Complex* preference experiment

it is also very small relative to the two scores. (e.g., for user B, the confidence interval of the difference between Q_{True} and $Q_{Learned}$ is $\frac{0.0225}{0.757} = 3\%$ of Q_{True}). In the case of user A, the confidence interval failed to cover zero. However, even though the difference between the two scores is not really zero, it is no more than $\frac{0.035}{0.826} = 4\%$ of Q_{True} with 95% confidence, indicating that performance of A's learned model very closely approximates that of A's true model.

8 Conclusion and Future Work

These two experiments provide initial evidence of the ability to learn static user preference models for meeting scheduling through observation. Our ongoing work is investigating the implications for user preference learning of other sets of assumptions about meeting scheduling protocols and organizational structures, as well as extension of user preference learning techniques to incorporate more complex, dynamic preferences (e.g., a preference for scheduling meetings back-to-back, a preference to bump less important meetings). Currently, CMRADAR utilizes a passive learning approach only, but we plan to integrate our system with an intelligent user interface to enable active learning by collecting user's feedback. Since the user's feedback provides the true answers, the agent's learning curve will be expedited. Knowing when and in what occasions the user should be interrupted is also another interesting learning task. Rule based strategies such as those proposed in CAP are more appropriate for capturing preferences more sensitively tuned to specific meeting types, i.e. implicitly recurring meetings. To tune our model to such customized meeting types, we anticipate extending our system to incorporate additional features in the distance metric, such as the subject of the meeting. If the number of features is large it will be more efficient to dynamically compute the utility values for alternative options, dynamically because the utility values will be customized according to the attributes of the given meeting by selectively choosing a subset of the data collected to date. For instance, if the given meeting is with person A and the subject of the meeting is Lunch, the system will use only the lunch meetings with person A to evaluate possible options, providing more customized evaluation criteria.

Acknowledgements

The authors thank Jay Modi and Manuela Veloso for the many stimulating discussions we have had on various aspects of calendar scheduling throughout the RADAR project. This research was sponsored in part by the Department of Defense Advanced Research Projects Agency (DARPA) under contract #NBCHC030029.

References

1. Berry, P., Gervasio, M., Uribe, T. E., Myers, K., Nitz, K.: A Personalized Calendar Assistant. AAAI Spring Symposium Series. (March, 2004)
2. Blum, A.: Empirical Support for Winnow and Weighted-Majority Based Algorithms: Results on a Calendar Scheduling Domain. Machine Learning **26** (1997) 5–23
3. Cohen, P. R.: Hypothesis Testing and Estimation. Chapter 4. MIT Press, Camridge, MA (1995)
4. Cost, S., Salzberg, S.: A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. Machine Learning **10** (1993) 57–78
5. Dent, C. L., Boticario, J., McDermott, J. P., Mitchell, T. M., Zabowski, D.: A Personal Learning Apprentice. In: Proceedings of AAAI-92 (1992) 96–103
6. Fiechter, C.-N., Rogers, S.: Learning Subjective Functions with Large Margins. In: ICML 2000 (2000) 287–294
7. Fix, E., Hodges, J. L.: Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties. Technical Report Project 21-49-004 4. USAF School of Aviation Medicine, Randolf Field, TX (1951)
8. Mitchell, T. M., Caruana, R., Freitag, D., McDermott, J. P., Zabowski, D.: Experience with a Learning Personal Assistant. Commun. ACM (1994) 80–91
9. Modi, P. J., Veloso, M., Smith, S. F., Oh, J.: CMRadar: A Personal Assistant Agent for Calendar Management. In: 6th Int. Workshop on Agent-Oriented Information Systems (AOIS) (2004) 134–148
10. Sen, S., Haynes, T., Arora, N.: Satisfying User Preferences While Negotiating Meetings. *Int. J. Human-Computer Studies* **47** (1997) 407–427

Appendix

Negotiation Protocols for CMRADAR Simulation

Within the CMRADAR simulation used in the experiment, each CMRADAR agent uses the following common negotiation protocol:

- *Initiator.* Initiator issues meeting request message and proposes a set of n options (time slots) that best suit its own preferences. In other words, the initiator proposes the n most preferred options (In the experimental runs $n = 3$.)
- *Attendees.* Each attendee responds to the meeting request as follows:
  - If one or more options are available, then evaluate and returned the *combined* preference value for each (see below).
  - If there are no available options, *bump* the least important pre-emptable meeting, and return the combined preference value for this newly freed time slot.
- *Initiator.* Collect all attendee responses. If there is an agreeable option then confirm it. Otherwise, repeat above steps with n new options.

The above protocol implements a common policy of favoring the preferences of higher ranked individuals by communicating and combining the preference values of initiator and attendee. More precisely, Let

- $Pref_i(t)$ = the value assigned by Initiator i to time slot t and communicated to attendee A
- $Pref_a(t)$ = the value assigned by attendee a to time slot t
- $R_i and R_a$ = the ranks of i and a respectively.

In evaluating a time slot proposed to attendee a by initiator i, a computes the following combined preference value:

$$Pref_{Comb}(t) = W_i \times Pref_i(t) + W_a \times Pref_a(t)\,,$$

where

$$W_i = 0.5 + 0.5 * \frac{R_i - R_a}{R_{Max} - R_{Min}}$$

and

$$W_a = 1.0 - W_i\,.$$

Semantic Components for Timetabling

Nele Custers, Patrick De Causmaecker,
Peter Demeester, and Greet Vanden Berghe

KaHo Sint-Lieven, Information Technology,
Gebroeders Desmetstraat 1, 9000 Gent, Belgium
{nele.custers, patrick.decausmaecker, peter.demeester,
greet.vandenberghe}@kahosl.be

Abstract. Automated timetabling is a research domain that has occupied many researchers over the last 50 years. Several algorithms have proven to be applicable to timetabling but they are nearly all designed to address specific problems. The framework presented in this paper is a step towards a generic semi-automatic timetabling tool. The basis of the framework is an ontology for timetabling that we designed after having investigated different types of timetabling problem.

A first step towards solving general problems consists of mapping their data representation to the ontology. That is carried out in such a way that existing data that is already available in databases can further be used throughout the application. In the next step, the tool assists in determining the constraints and objectives of the problem. The semantic components have three sources of information: meta-data about the database, domain knowledge about timetabling problems and external, non-domain-specific knowledge.

1 Introduction

From the early definitions of timetabling problems (e.g. [9], see [10], [17], [18] for an overview) on, it has been clear that timetabling was a complicated problem deeply entangled in the real world. From then onwards several researchers have tried to model a variety of real-world cases [22], [32], [33], [39], [50]. An equally large variety of solving techniques have been tried out and experimented with (for example tabu search [24], [61], genetic algorithms [16], [26], [59], constraint logic programming [19], [29], [39], simulated annealing [14], [55], evolutionary algorithms [54]). Quite often a single methodology was not sufficient. In most real-world cases only a hybrid approach would lead to satisfactory solutions [12], [42], [60]. Both modelling and techniques have seen significant progress over the years. On the modelling side, specific languages were created [11], [47], [48], [51], and development frameworks have been built [23], [28], [38]. Those frameworks play an in-between role while offering modelling support as well as libraries of solution approaches [23], [28]. Methodologies, on their side, have evolved from problem-specific [27] to more general [3], [4]. The evolution is in line with the more general trend in optimisation technology, especially in the field of metaheuristics [52]. The complexity and importance of those real-world applications

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 17–33, 2005.

together made it an excellent field for research in optimisation technology. They both ignited new technological ideas and offered a test bed for new developments. Application domains such as university course and exam timetabling [35], [37], sport timetabling [58], employee timetabling [36] were studied by cases. Sometimes even more specialised domains—such as nurse rostering (see [15] for a survey)—effectively grew into important research subjects with their own research communities. This strong real-world entanglement and the fruitful years of intensive research naturally lead to tools that support the development of timetabling systems. A number of efforts in these directions have been reported [11], [28], [48], [51]. Recent years have seen a boost in the development of methodologies for integration [2], [34]. Again, the timetabling domain, with its advanced status of knowledge, can serve as a test bed. In this paper we present a set of integration tools that we have developed with this goal in mind.

Languages as in [11], [48], [51] allow the description of timetabling problems to which solution methods can be applied. Frameworks as presented in [23], [28] support the development of timetabling systems using libraries and data structures. The approach of the present paper is to try to automatically read an existing system and to semi-automatically extract the essential concepts. We furthermore provide tools that allow users or operators to define their constraints. The result is a system that is linked to a database but that uses an ontology which allows different components such as solvers and editors to smoothly interact. We are thus one step further away from the programming level of the languages and framework-focused approaches.

While studying different timetabling problems, it is possible to identify a common set of characteristics. In Figure 1, we have grouped four different timetabling (or scheduling) examples with varying terminology such as lecture, game, qualification, operation, etc. As can be seen from these examples, timetabling always involves assigning activities to timeslots (and locations, people, etc), subject to constraints. Constraints that one developer calls soft may be hard for another developer. In [62], Wren defines timetabling as “the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives.” Hard constraints are those that must be satisfied at all time. Soft constraints should preferably be satisfied but violations will affect the quality of the solution. Reducing the number of violated soft constraints is a common objective in timetabling. Often, the objective of timetabling problems is expressed in terms of a real-valued function, called the objective function, that reflects a degree of constraint violation.

We have developed a framework for timetabling applications with a central “time-tabling” domain ontology [21]. Ontologies enable information sharing between researchers in a specific domain. They contain at least some machine-interpretable definitions of domain concepts and their relationships. The timetabling ontology is based on the general OZONE [53] scheduling ontology. We used the DAML+OIL [20] ontology language, which is founded on web standards such as XML [8] and RDF [40], to express the timetabling ontology. RDF provides a format to describe data using XML as a serialisation syntax. All these

Table 1. Examples of possible timetabling (scheduling) problems with some sample hard and soft constraints

	School timetabling	Sports timetabling	Nurse rostering	Job shop scheduling
Scheduling object	Session (lecture)	Game	Shift (assignment)	Operation (of a job)
Problem dimension	Time	Sports day	Time	Start time of operation
Attributes of scheduling object	Lecture (group) Student (group)	Home team Visitor team	Shift type Qualification	Machine Job
Examples of hard constraints	Lecture @place/time≤ 1 Student @place/time≤ 1 Full curriculum No sessions on holidays	Double round robin (A at B and B at A) No repeaters (A and B, followed immediately by B at A) Not more than three consecutive home or away games for any team	Nurse qualifications Coverage constraints	Tcol(s) Precedence constraints Capacity Resources Stock
Examples of soft constraints	Capacity constraints Room facilities Precedence lectures Lecture preferences Student preferences	Play home match if local bank holiday Broadcasting preferences	Contract constraints: overtime workload, consecutive shifts, . . . Personal preferences Patterns	Due date Machine constraints
Goal: minimise	Violated soft constraints	Distance travelled	Violated soft constraints	Tardiness Processing time

languages and standards fit in the Semantic Web [6] initiative of the World Wide Web Consortium (W3C). More detailed information about the use of these languages in timetabling applications is available in [21].

The developed timetabling ontology allows the application of one single approach to solve a multitude of different problems. The ontology acts as an intermediate language between the data layer (in this case an already existing database that contains specific information about the timetabling problem at hand) and a calculation component. In the case that the problem data is stored in a problem-specific database, the only requirement is a mapping (a kind of semantic translation) of the database schema to the ontology. The mapping can be performed semi-automatically (user assistance is still required) by the "semantic mapping component" presented in Section 3. We do not aim at fully automatic mapping but input from a domain expert will be required during the mapping process. A calculation component, which is aware of the ontology, can compute a solution with the data extracted from the data layer. Of course, a general solution framework will not be competitive with made-to-measure algorithms for specific problems. Optimal performance across different timetabling domains is however not our main concern. We rather aim at providing satisfactory decision support for a variety of timetabling problems.

In Figure 1, a schematic overview of the tool is presented. All depicted parts will be discussed in the following sections. We use an existing real-world university database to test the semantic application. The database contains all data concerning rooms, students, teachers,... of our institute and is used to manually create the timetables. We also tested the application on two simplified databases containing data on nurse rostering and games.

This paper focuses on the semantic part of the developed application and not on the calculation component. In Section 2, we discuss other recent contributions to semantic mapping. The semantic mapping component for timetabling is introduced in Section 3. In Section 4, we describe the user support for defining the timetabling problem. Section 5 describes implementation issues, while Section 6 gives directions for future research. Section 7 concludes.

2 Semantic Mapping Tools

The ontology mapping tool described by Prasad et al. [49] uses explicit information. It must be provided by "exemplars" that describe the meaning of the concepts in both ontologies. By using a text classifier, a model is built for each ontology. Afterwards, the models of both ontologies are compared and the concepts with the highest similarity scores are considered for mapping. Two algorithms (heuristic and Bayesian) have been developed to finalise the mappings. The ontology mapping tool GLUE [25] applies a multi-strategy learning approach with a set of learners of which the predictions are combined. Domain constraints and general heuristics improve the accuracy of the matching.

Anchor-PROMPT [45] and PROMPT [46] are tools for merging ontologies. The activity of mapping ontology concepts is an important step in the merging

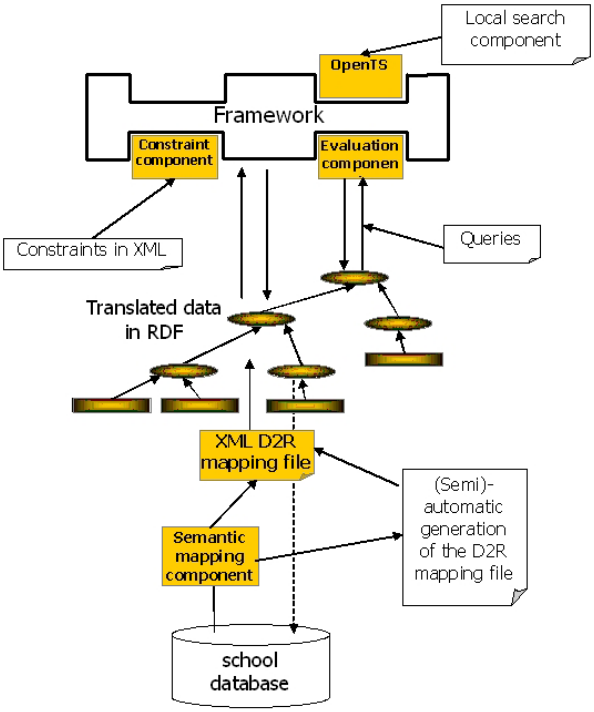


Fig. 1. Schematic overview of the tool

process. The tool described in this paper has a similar aim since it tries to map the concepts in a database schema to an ontology. It differs in the sense that one ontology is considered fixed and will probably only cover a subset of the concepts in the database. PROMPT explores the ontologies to locate candidate terms for merging. It uses both syntactic and semantic information and also feedback from the user. Anchor-PROMPT first searches pairs of related terms in different ontologies ("anchors"), which are identified by the user or automatically generated by the system. Starting from these anchors, Anchor-PROMPT searches a new pair of terms on the path between anchors.

KAON Reverse [56] is the tool that approaches our needs for a semantic mapping tool best. It allows to export data from a database to an ontology. As a prototype, KAON Reverse's functionality is too basic to fulfil the requirements of the timetabling mapping tool.

Missikof et al. [44] developed a software environment with the OntoLearn tool as a core. It can build and valuate domain ontologies. The software environment acquires new domain concepts by exploring available documents and related Web sites. WordNet (see Section 3) is one of the resources used for the semantic interpretation of the corpus. Missikof et al. state that capturing kindship relations is clearly important for an ontology based Web application.

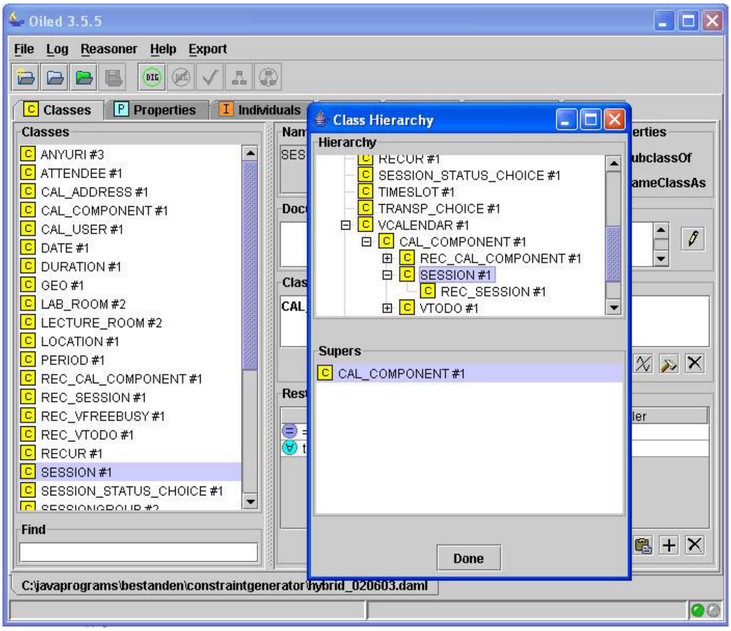
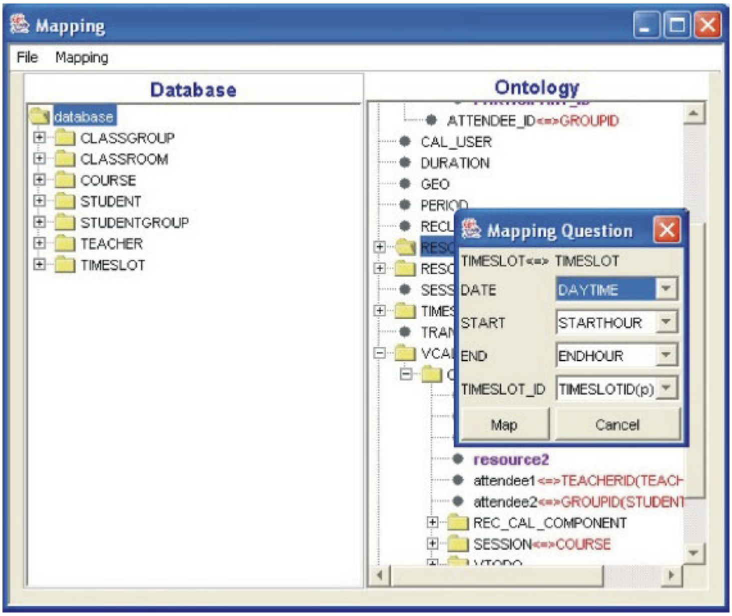


Fig. 2. The timetabling ontology presented in OilEd

3 Semantic Mapping Component

The purpose of the semantic mapping component is to "map" the specific problem data of a relational database to the timetabling ontology. The structure and content of the database will always be problem dependent, but the target ontology is always the same. In Figure 2, the timetabling ontology is represented in an ontology editor called OilEd [5]. The complete ontology can be found at http://ingenieur.kahosl.be/projecten/cofftea/2003/03/hybrid.daml. The ontology is rather general since different problem-specific databases have to be mapped to it. The developed tool will help to discover and map data that is directly available in the database to the ontology. One of the key concepts in the ontology is the "SESSION" concept. It corresponds to what is called the "scheduling object" in Figure 1. The concept contains properties such as location, timeslot, date, attendee and SESSION_ID, that are useful for describing concepts in detail. There are also constraints attached to these properties. One of the constraints is that there is exactly one unique SESSION_ID per SESSION concept.

Each relational database consists of relations (tables) with attributes. There are two types of special attribute: primary keys and foreign keys. One condition for the mapping component is that there are no composite primary keys in the relations. The second part of the semantic mapping component consists of the ontology for timetabling problems and is shown in the right part of the



TIMESLOT<=> TIMESLOT
DATE DAYTIME
START STARTHOUR
END ENDHOUR
TIMESLOT_ID TIMESLOTID(p)

Fig. 3. GUI of the mapping component

user interface (see Figure 3). The ontology includes classes with properties. We can identify two special property types: identifier and reference properties. The first step in the process consists of mapping a relation from the database to a class in the ontology. Next, the attributes of the relation are mapped to the corresponding properties of the class. The correspondence between the database and the ontology terminology is illustrated in Figure 4.

The semantic mapping component uses three kinds of information source. First of all, meta data about the database offers useful knowledge. We mark the primary key of each relation and annotate each foreign key with the relation whose candidate key matches. Domain knowledge about timetabling problems is a second source of information. Timetabling concerns resources that need to be scheduled in a time frame. Apart from these problem-specific information sources, WordNet [1] adds external information to the mapping component. WordNet is the result of a research project at Princeton University that has attempted to model the lexical knowledge of a native English speaker. Information in WordNet is organised around logical groupings called synsets. Each synset consists of a list of synonymous word forms and semantic pointers that describe relationships between the current synset and other synsets. These semantic pointers can be of a number of different types including: "Hyponym/Hypernym (Is-a, Has-a)" and "Meronym/Holonym (Part-of/Has-part)".

Figure 5 describes the procedure for mapping the database to the ontology. In the first step, the central timetabling object is identified by the mapping process.

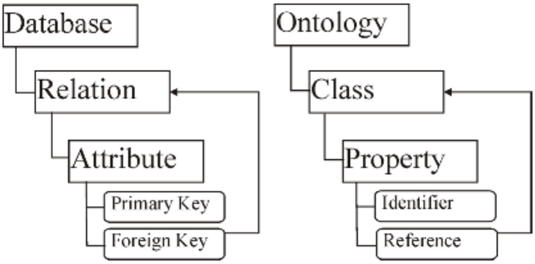


Fig. 4. Terminology

That object will be liable to the local search component and must be mapped to the class "Session" in the ontology. A first possible way to recognise this central timetabling object is its number of foreign keys. The central timetabling object of the database will often be the relation with the largest number of foreign keys because it involves a lot of resources. A lecture, for example, is assigned to a teacher, students, a room, etc. The second rule for finding the central timetabling object indicates that it is a kind of activity or event. That characteristic can easily be checked using WordNet. If this still does not identify the central timetabling object, user assistance is required.

Once the relation in the database that represents the central timetabling object is found, mapping its attributes can start. A first rule for mapping attributes is that the identifier of the class in the ontology will be mapped to the primary key of the corresponding relation. For example, we managed to recognise "shift" (a task to be carried out within a specified time period) in nurse rostering as the central scheduling object. Therefore, "shift" will be mapped to the ontology class "Session". The ontology class "Session" contains an identifier "SessionID". We will map the primary key of the relation "shift" to that identifier. Other attributes of the relation are mapped to the most similar property of the ontology. We apply a slightly altered version of the recursive algorithm for computing string similarity [30]. Attributes can also be mapped to properties that are not similar, by using knowledge about the English language. For example, "room" can be mapped to "location" because "room" is a kind of "location" according to WordNet. Similarly, "nurse" is a possible dimension of the nurse rostering search space because it is a kind of "professional" and it is an attribute of the central timetabling object. Once a foreign key of the relation can be mapped to a reference property of the ontology, we can also map the corresponding relation to the reference class. Subsequently, we can map attributes of the relation to properties of the class, etc.

The Semantic Mapping Component also determines the dimensions of the solution space. In school timetabling, for example, sessions can be moved in time and space (rooms). In the sport timetabling problem that we studied [57], one single dimension (time) suffices (see also Figure 1). The result of the mapping process is a semi-automatically generated XML file that is in an appropriate format to be used by the D2R mapping tool [7]. That tool allows translation

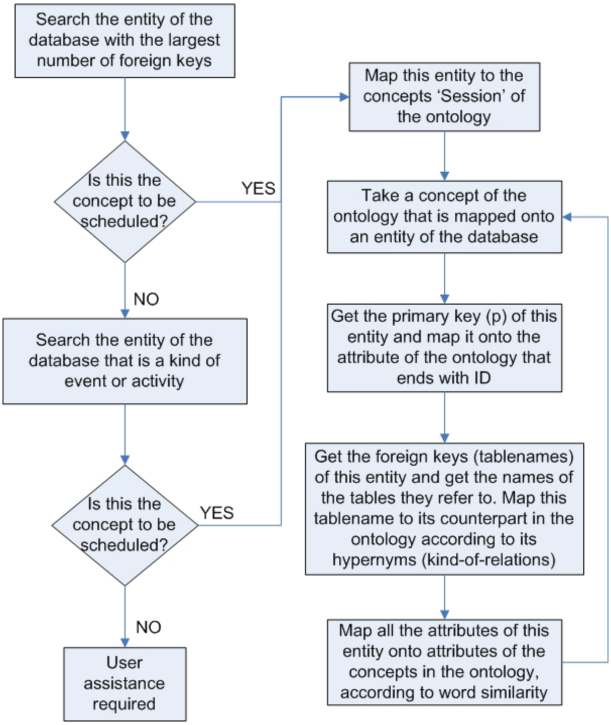

Search the entity of the database with the largest number of foreign keys
Is this the concept to be scheduled?
YES
NO
Search the entity of the database that is a kind of event or activity
Is this the concept to be scheduled?
YES
NO
User assistance required
Map this entity to the concepts 'Session' of the ontology
Take a concept of the ontology that is mapped onto an entity of the database
Get the primary key (p) of this entity and map it onto the attribute of the ontology that ends with ID
Get the foreign keys (tablenames) of this entity and get the names of the tables they refer to. Map this tablename to its counterpart in the ontology according to its hypernyms (kind-of-relations)
Map all the attributes of this entity onto attributes of the concepts in the ontology, according to word similarity

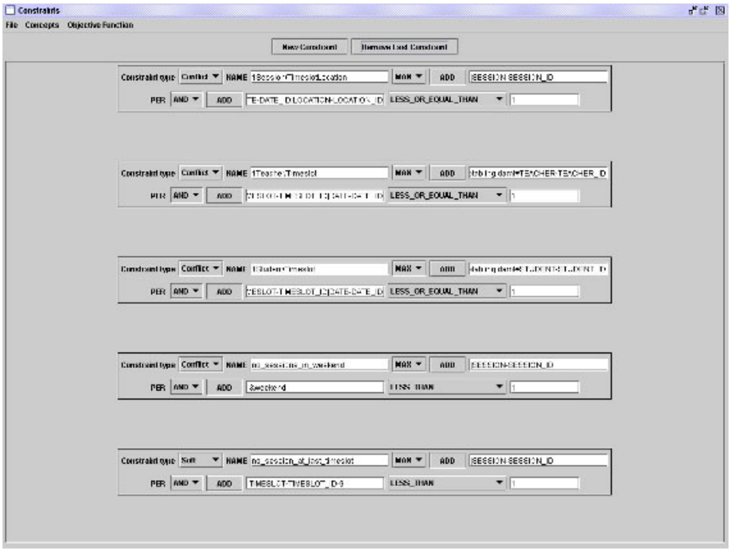
Fig. 5. Mapping rules

of data from a relational database to an RDF file. Normally, the D2R XML file that describes how to map data from the database to the RDF file is manually written, but our mapping tool is now able to semi-automatically construct the "translation" file. The resulting RDF data file, which contains the problem specific data in terms of the timetabling ontology, will be used in the computational part of the framework and in the graphical semantic constraint generator.

4 Timetabling Characteristics

4.1 Constraints

Timetabling problems are not completely characterised by the RDF data file alone. Some additional semantic components are required. We have developed a semantic tool—the graphical semantic constraint generator (GSCG)—to assist in defining the timetabling problem with concepts from the user's domain. The GSCG (Figure 6) builds on the results of the mapping component and enables the user to specify the constraints. It stores the constraint description in separate XML files. We opt to save the constraints directly in XML, since we experienced



Constraint type Conflict NAME 1Session/TimeslotLocation MAX ADD SESSION-SESSION_ID
PER AND ADD TE-DATE_ID|LOCATION-LOCATION_ID LESS_OR_EQUAL_THAN 1

Constraint type Conflict NAME 1Teacher/Timeslot MAX ADD tabling.daml#TEACHER-TEACHER_D
PER AND ADD [illegible] LESS_OR_EQUAL_THAN 1

Constraint type Conflict NAME 1Student/Timeslot MAX ADD tabling.daml#STUDENT-STUDENT_ID
PER AND ADD ESLOT-TIMESLOT_ID|DATE-DATE_ID LESS_OR_EQUAL_THAN 1

Constraint type Conflict NAME no_sessions_in_weekend MAX ADD SESSION-SESSION_ID
PER AND ADD &weekend LESS_THAN 1

Constraint type Soft NAME no_session_at_last_timeslot MAX ADD SESSION-SESSION_ID
PER AND ADD TIMESLOT-TIMESLOT_ID-9 LESS_THAN 1

Fig. 6. GUI for constructing constraints

that it was hard to express the constraints in DAML+OIL. Such difficulties are also indicated in [41], [43].

Constraints are expressed as follows:

number of concept1 PER concept2 is [LESS THAN|EQUAL|etc] concept3 .

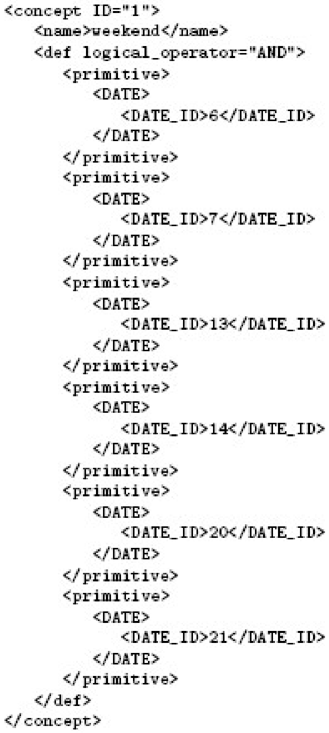
For example,

$$\texttt{(number of sessions) per (timeslot and location)} \leq \texttt{1}$$

means that there cannot be more than one session per timeslot in a location (room). Both hard and soft constraints can be expressed in the same form. In order to define comprehensive constraints, we sometimes need information that is not available in the ontology. “Weekend” is an example of a concept that is commonly used in timetabling constraints but not available in the timetabling ontology nor in the database. The XML code in Figure 7 clearly demonstrates that the existing (primitive) concept “Date” is used to define the concept “Weekend”. It is actually the translated primary key of the date concept (`DATE_ID`) which is used. In the example, we consider a period of 3 weeks (21 days) consisting of 3 weekends. Every weekend is defined as consisting of a Saturday and a Sunday. Of course, it is also possible to define a “Weekend” that starts on Friday and ends on Monday.

4.2 Objective Function

Once the mapping has been completed and the constraints are known, the timetabling problem is pretty well defined. The local search component, however,



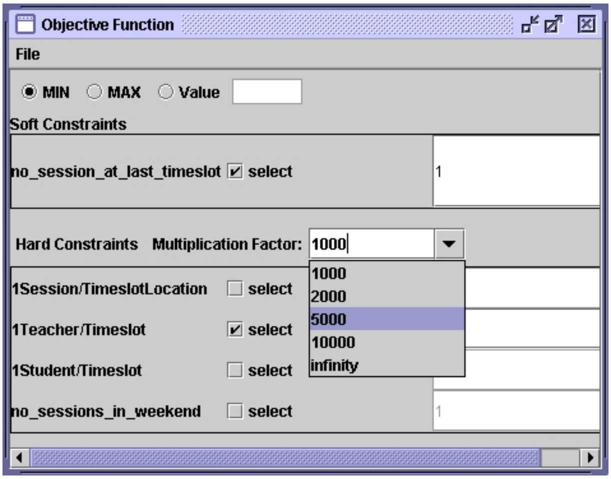
```
<concept ID="1">
   <name>weekend</name>
   <def logical_operator="AND">
      <primitive>
         <DATE>
            <DATE_ID>6</DATE_ID>
         </DATE>
      </primitive>
      <primitive>
         <DATE>
            <DATE_ID>7</DATE_ID>
         </DATE>
      </primitive>
      <primitive>
         <DATE>
            <DATE_ID>13</DATE_ID>
         </DATE>
      </primitive>
      <primitive>
         <DATE>
            <DATE_ID>14</DATE_ID>
         </DATE>
      </primitive>
      <primitive>
         <DATE>
            <DATE_ID>20</DATE_ID>
         </DATE>
      </primitive>
      <primitive>
         <DATE>
            <DATE_ID>21</DATE_ID>
         </DATE>
      </primitive>
   </def>
</concept>
```

Fig. 7. Example of XML file that expresses the “Weekend” concept

still requires an objective function that evaluates constraint violations. A final component (Figure 8) supports the user when defining this function. The component can be regarded as a tuning instrument for the application that allows the user to express the relative importance of the constraints.

5 Implementation Issues

We succeeded to map most of the data fields in the real-world university database that we used as a test case (see Section 1). The mapping failed where stored procedures were used. There is insufficient semantic information available for our system to map such items to their ontology counterparts. Instead, we had to write extra Java code to obtain these data. We, for example, need the number of students that attend a session in order to decide whether the capacity of the room is sufficient. Since a session can be attended by different student groups, the total number of attendees is to be computed through a stored procedure that sums the students in all the groups. In order to solve that problem, we executed the stored procedure from a Java client and imported the obtained information in the already translated mapped data file.



● MIN ○ MAX ○ Value

Soft Constraints

no_session_at_last_timeslot ☑ select 1

Hard Constraints Multiplication Factor: 1000

1Session/TimeslotLocation □ select

1Teacher/Timeslot ☑ select

1Student/Timeslot □ select

no_sessions_in_weekend □ select 1

Fig. 8. GUI for constructing the objective function

6 Future Research

The basic calculation component currently handles a specific type of constraint. We are investigating an extension of this component that allows expressing and evaluating general constraints. Specifically, we will look at the numbering technique [13] that was developed for a shift assignment system to allow for an exact definition of concepts such as minimal contiguity or maximum interleave. We want to investigate if the numbering technique can be extended to define and evaluate all possible timetabling problems. A component that is based on the numbering could perfectly well be used with a local search component. Currently, we are experimenting with the OpenTS framework [31] for optimisation and search.

7 Conclusions

It is the aim of this research to apply specific timetabling knowledge for solving any kind of timetabling problem within a generic framework. Although this is a demonstration project, we have achieved to support experienced planners in integrating their problem (represented in a database) into the timetabling framework. Given the following information:

- the ontology,
- the mapped problem data,
- the dimensions of the solution space,
- the objective function,
- and a list of hard and soft constraints,

a local search component can be called to solve any kind of timetabling problem. We carried out qualitative experiments on a number of timetabling problems (as described in Section 1). The results are promising but, as discussed in Section 5, not all the real-world data can be mapped to the ontology. We believe that semantic components such as these presented in this paper, will become very valuable when it comes to re-using timetabling software.

References

1. Al-Halimi, R. et al.: WordNet—An Electronic Lexical Database. Bradford Books, MIT Press, Cambridge, MA (1998)
2. Anicic, N., Marjanovic, Z.: Integration of Business Applications Using Semantic Web Technologies. First Int. Conf. on Interoperability of Enterprise Software and Applications, Geneva (2005)
3. Barták, R.: A Generalized Framework for Constraint Planning, Technical Report No 97/9. Department of Theoretical Computer Science, Charles University, Prague (June 1997)
4. Barták, R.: Modelling Planning and Scheduling Problems with Time and Resources, in Recent Advances in Computers, Computing and Communications. WSEAS Press, Rethymnon, Greece (2002) 104–109
5. Bechhofer, S., Horrocks, I., Goble, C., Stevens, R.: OilEd: a Reason-able Ontology Editor for the Semantic Web. In: Proceedings of KI2001, Joint German/Austrian Conference on Artificial Intelligence (Vienna). Lecture Notes in Artificial Intelligence, Vol. 2174. Springer, Berlin (2001) 396–408
6. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web, Sci. Am. (May, 2001)
7. Bizer, C.: D2R MAP—A Database to RDF Mapping Language. In: Proc. WWW (Budapest) (2003)
8. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E.: Extensible Markup Language (XML) 1.0 (2nd edn). http://www.w3.org/TR/REC-xml. W3C Recommendation (2000)
9. Broder, S.: Final Examination Scheduling. Commun. ACM. **7** (1964) 494–498
10. Burke, E. K., Elliman, D. G., Ford, P. H., Weare, R. F.: Examination Timetabling in British Universities—A Survey. In: Burke, E. K., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 76–90
11. Burke, E. K., Kingston, J. H., Pepper, P. A.: A Standard Data Format for Timetabling Instances. In: Burke, E., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 213–222
12. Burke, E. K., Cowling, P., De Causmaecker, P., Vanden Berghe, G.: A Memetic Approach to the Nurse Rostering Problem. Appl. Intell. (Special Issue on Simulated Evolution and Learning) **15** (2001) 199–214
13. Burke., E. K., De Causmaecker, P., Petrovic, S., Vanden Berghe, G.: Fitness Evaluation for Nurse Scheduling Problems. In: Proc. Congress on Evolutionary Computation, (CEC2001, Seoul) IEEE Press, Piscataway, NJ (2001) 1139–1146
14. Burke, E. K., Bykov, Y., Newall, J. P., Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Trans. Oper. Eng. **36** (2004) 509–528

15. Burke, E. K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The State of the Art of Nurse Rostering. J. Scheduling **7** (2004) 441–499
16. Carrasco, M. P., Pato, M. V.: A Multiobjective Genetic Algorithm for the Class/Teacher Timetabling Problem. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 3–17
17. Carter, M. W., Laporte, G.: Recent Development in Practical Examination Timetabling. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 3–21
18. Carter, M. W., Laporte, G.: Recent Development in Practical Course Timetabling. In: Burke, E., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 3–19
19. Chan, P., Weil, G.: Cyclical Staff Scheduling Using Constraint Logic Programming. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 159–175
20. Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A.: DAML+OIL Reference Description. W3C Note 18 (December, 2001) http://www.w3.org/TR/daml+oil-reference
21. De Causmaecker, P., Demeester, P., Lu, Y., Vanden Berghe, G.: Using Web Standards for Timetabling. In: Burke, E., De Causmaecker, P. (eds.): PATAT 2002—Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling 238–257
22. De Causmaecker, P., Vanden Berghe, G.: Relaxation of Coverage Constraints in Hospital Personnel Rostering. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 129–147
23. Di Gaspero, L., Schaerf, A.: EasyLocal++: An Object-Oriented Framework for the Flexible Design of Local Search Algorithms. Softw. Pract. Exp. **33** (2003) 733–765
24. Di Gaspero, L., Schaerf, A.: Multi-Neighbourhood Local Search with Application to Course Timetabling. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 262–275
25. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to Map Between Ontologies on the Semantic Web. In: 11th Int. WWW Conf. (Hawaii, 2002) 662–673
26. Erben, W.: A Grouping Genetic Algorithm for Graph Coloring and Exam Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 132–156
27. Foxley, E., Lockyer, K.: The Construction of Examination Timetables by Computer. Comput. J. **11** (1968) 264–268
28. Gröbner, M., Wilke, P., Büttcher, S.: A Standard Framework for Timetabling Problems. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 24–38

29. Guéret, C., Jussien, N., Boizumault, P., Prins, C.: Building University Timetables Using Constraint Logic Programming. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 130–145
30. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Press Syndicate of the University of Cambridge, ISBN 0521585198 (June, 1997)
31. Harder, R.: OpenTS—Java Tabu Search http://www-124.ibm.com/developerworks/oss/coin/OpenTS/index.html
32. Henz, M., Würtz, J.: Using Oz for College Timetabling. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 162–177
33. Franses, P., Post, G.: Personnel Scheduling in Laboratories. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 113–119
34. Izza, S., Vincent, L., Burlat, P.: Ontology-Based Approach for Application Integration. In: 1st Int. Conf. on Interoperability of Enterprise Software and Applications (Geneva, 2005)
35. Kaplansky, E., Kendall, G., Meisels, A., Hussin, N.: Distributed Examination Timetabling. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 511–516
36. Kaplansky, E., Meisels, A.: Negotiation Among Scheduling Agents for Distributed Timetabling. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 517–520
37. Kendall, G., Hussin, N. M.: Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at University Technology MARA. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 199–217
38. Kingston, J. H., Yin-Sun Lynn, B.: A Software Architecture for Timetabling Construction. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 309–321
39. Lajos, G.: Complete University Modular Timetabling Using Constraint Logic Programming. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 146–161
40. Lassila, O., Swick, R. R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation (1999) http://www.w3.org/TR/REC-rdf-syntax
41. Lockheed Martin: DAML Language Lessons Learned. http://ubot.lockheedmartin.com/ubot/lessons/language.html
42. Merlot, L. T. G., Boland, N., Hughes, B. D., Stuckey, P. J.: A Hybrid Algorithm for the Examination Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 207–231
43. Miller, L., FitzPatrick, G., Brickley, D.: SkiCal and iCalendar in DAML+OIL: A Case Study. http://www.ilrt.bris.ac.uk/discovery/2002/03/skical-daml/

44. Misskof, M., Navigli, R., Velardi, P.: Integrated Approach to Web Ontology Learning and Engineering. IEEE Comput. **35** (2002) 60–63
45. Noy, N. F., Musen, M. A.: Anchor-PROMPT: Using Non-Local Context for Semantic Matching. Workshop on Ontologies and Information Sharing at the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI-2001, Seattle) (2001)
46. Noy, N. F., Musen, M. A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. Proc. 7th Natl Conf. on Artificial Intelligence (AAAI-2000, Austin) (2000) 450–455
47. Okada, M., Okada, M.: Prolog-Based System for Nursing Staff Scheduling Implemented on a Personal Computer. Comput. Biomed. Res. **21** (1988) 53–63
48. Özcan, E.: Towards an XML Based Standard for Timetabling Problems: TTML. In: Proc. 1st Multidisciplinary Int. Conf. on Scheduling: Theory and Applications (MISTA'03, Nottingham) (2003) 163–185
49. Prasad, S., Peng, Y., Finin, T.: Using Explicit Information To Map Between Two Ontologies. Proc. Workshop on Ontologies in Agent Systems, 1st Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (Bologna, 2002)
50. Rankin, R. C.: Automatic Timetabling in Practice. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 266–279
51. Reis, L. P., Oliveira, E.: A Language for Specifying Complete Timetabling Problems. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 322–341
52. Resende, M. G. C., Pinho de Sousa, J.: Metaheuristics: Computer Decision-Making. Kluwer, Dordrecht (2004)
53. Smith, S. F., Becker, M. A.: An Ontology for Constructing Scheduling Systems. In: Working Notes AAAI Spring Symposium on Ontological Engineering (Stanford, March, 1997) 120–129
54. Socha, K., Knowles, J., Sampels, M.: A MAX–MIN Ant System for the University Timetabling Problem. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.): Proc. ANTS'02—From Ant Colonies to Artificial Ants. 3rd Int. Workshop on Ant Algorithms. Lecture Notes in Computer Science, Vol. 2463. Springer, Berlin (2002) 1–13
55. Sosnowska, D., Rolim, J.: Fleet Scheduling Optimization: A Simulated Annealing Approach. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 227–241
56. Stojanovic, N., Stojanovic, L., Volz, R.: A Reverse Engineering Approach for Migrating Data-Intensive Web Sites to the Semantic Web. Intelligent Information Processing IFIP 17th World Computer Congress: TC12 Stream on Intelligent Information Processing (Montreal, 2002) 141–154
57. Trick, M. A.: Challenge Traveling Tournament Instances. http://mat.gsia.cmu.edu/TOURN/
58. Trick, M. A.: Scheduling Court Constrained Sports Tournaments. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 371–376

59. Ueda, H., Ouchi, D., Takahashi, K., Miyahara, T.: A Co-evolving Timeslot/Room Assignment Genetic Algorithm Technique for University Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 48–63
60. White, G. M., Zhang, J.: Generating Complete University Timetables by Combining Tabu Search with Constraint Logic. In: Burke, E., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 187–198
61. White, G. M., Xie, B. S.: Examination Timetables and Tabu Search With Longer-Term Memory. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 85–103
62. Wren, A.: Scheduling, Timetabling and Rostering—A Special Relationship? In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 46–75

An Open Interactive Timetabling Tool

Sylvain Piechowiak, Jingxua Ma, and René Mandiau

LAMIH-CNRS, University of Valenciennes, France
sylvain.piechowiak@univ-valenciennes.fr

Abstract. This article deals with the analysis and design of an interactive decision support system for timetable management. This tool will be able to take hierarchical data organization into account and to maintain coherence of the constraints on this data. Our research which has led to the creation of the VT tool[1] has two aims. The first aim is to provide an open, generic tool which can be developed in many different ways. In order to achieve this aim, we have used an object-oriented approach and we have defined object classes for the modelling of the timetabling problem. The second aim is to analyse the needs in timetable manipulation and to provide a generic organization so that the tool can be used in many situations. To achieve this aim, both user-based and automated techniques are used.

1 Introduction

Every year, the task of the university educational managers is to organize timetables for the various courses or branches, whilst trying, as far as possible, to meet the "human" constraints of the teachers and students, along with the pedagogical constraints imposed by teaching progression and the "physical" constraints linked to material resources (rooms, equipment, etc).

Up until now, managers have explored two different ways: graphical tools and fully automated tools.

The graphical tools (like tabler or dedicated tools) are easy to use. They are powerful to draw pretty timetables. Unfortunately these tools give very poor help especially to find conflicts in timetables. They give no help to resolve these conflicts.

The fully automated tools [3], [10], [20], [21] are able to find a timetable. But generally, these tools require powerful machines. They are better feet the needs of secondary schools. Their timetables are built on a weekly basis. The development of a timetable for a year only duplicates the weekly structure. These automated generation tools are also well suited to the examinations [5], [11] or conferences [8].

Fully automated tools are not efficient when the constraints cannot lead to a valid solution (impossibility of building a clash-free timetable). In these situations, the tools do not provide any support in explaining the causes for the

[1] VT for visual timetabling: http://visual.timetabling.free.fr

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 34–50, 2005.

lack of solution. Nothing is given to determine which constraints must be relaxed to bring about a solution. The quality of these timetables also depends on the exhaustiveness of the constraints. In a university, it is impossible to collect and to formalize all this information. Expertise of timetablers is the key.

Besides, more advanced techniques have been explored by the constraint programming community for the design of timetables [15]. These techniques allow users to give some guidelines to the system and obtain quicker or more adequate solution. However, this approach is still focused on systems. We have chosen a different point of view since our work is focused on the end user.

The tool required must above all be interactive, adaptable and open, and it must have ergonomic qualities [14].

We will describe first the current organization at the University of Valenciennes and Hainaut-Cambrésis (UVHC) and more specifically at the Institute of Sciences and Techniques of Valenciennes (ISTV). We will then present the `VT` tool, written in DELPHI object-oriented language and we will describe the different objects defined. Finally, we will give some obtained results using `VT` and we present perspectives to get the highest level of genericity for this kind of tool and to make it designed for a multi-user configuration.

2 Modelling of the Timetable Problem

In an abstract view, the timetable problem consists in distributing over time pedagogical activities which require resources (teachers, groups, rooms and equipment). In order to respect the usual vocabulary, these activities are called teaching modules and each occurrence of an activity is called a session. The sessions have a duration. The timetable problem therefore requires the modelling of the activities, the resources and the time.

2.1 Modelling of the Pedagogical Activity

The pedagogical activity is modelled using one entity *teaching module.*

A teaching module is characterized by a name, a *subject*, a total duration, a default duration (the duration suggested by default for each teaching module session) and a set of resources. Experience in the field led us to separate the resources into two sets: imposed resources and necessary resources. The distinction made between these two types of resource makes it possible to set certain constraints as soon as the teaching module is described.

In addition to these characteristics, each teaching module must be situated in relation to the others. For example, the E2 teaching module may only take place once the E1 has been completely finished.

2.2 Modelling of Resources

The resources considered are the physical entities necessary for the preparation of timetables: rooms, teachers, groups, students and equipment. In order to take

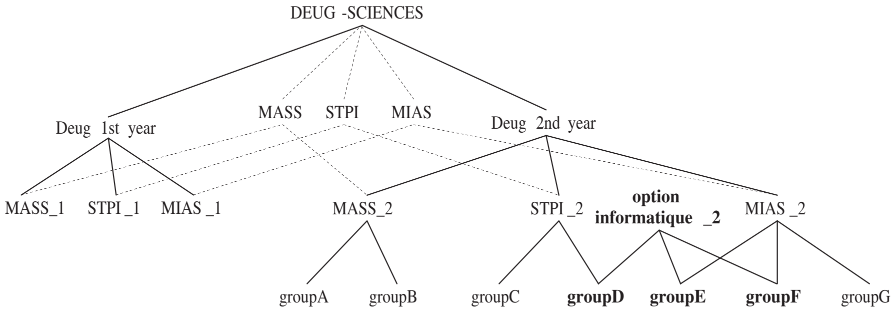


Fig. 1. Example of group hierarchy in the ISTV

in most of the configurations imaginable, the resources are organized according to a hierarchical structure. Each resource R has daughter resources and can be the daughter of several other resources. Figure 1 shows an example.

Each reference to a resource R also concerns all the daughters of R, as well as all its "granddaughters", down to the lowest level of the hierarchy. Moreover, as soon as a session concerning R is placed, this limits the degrees of freedom of all its mother resources.

In the ISTV, this organization of the resources into hierarchical structures is used for the groups and for the equipment.

Five categories of resources are considered: the rooms, the teachers, the groups, the students and the equipment. Each resource is described with data to identify it (like its name) and data to characterize it (like capacity, speciality, etc).

2.3 Modelling of Time

Two fields are used in order to represent time: the instant and the duration. The duration of each session is distinguished from the duration between the sessions. It should be noted that this representation of time is more flexible than that generally chosen in secondary schools. In classical timetabling problems each day is divided into a set of time slots which are fixed [4]. In these cases, each session can only take place in one or several consecutive slots. Here a session can begin at each time in a day.

The granularity of time makes it possible to link representations of time with various degrees of sharpness. At the lowest level, we find the basic temporal representation (for example 15 minutes).

In order to model time, the following entities are defined: date, time, duration, slot and calendar.

A date refers to a triplet (day, month, year). Using this triplet, the value associated with it on the day axis is defined. A time is a whole number included between the minimum value $HMin$ (8h00) and the maximum value $HMax$ (19h00). These two numbers correspond to times in relation to a date.

A duration is a number included between $DMin$ and $Dmax = Hmax - HMin$. $DMin$ represents the smallest temporal unit available.

A time slot refers to a temporal interval during a day. It is characterized by a couple (H, D) in which H represents the starting time of the slot and D its duration.

A calendar is a set of dates. Each date is associated with a state (available or unavailable).

Each teaching module is also associated with a calendar to specify when it can be planned during the year. For example we can decide that a teaching module must be planned on Friday and another from September 1st to December 15th.

2.4 Sessions, Schedules and Bookings

A session corresponds to a temporal event of a teaching module on a given date, during a specific slot. The characteristics of a session are: its teaching module, its date, its slot, its equipment and its room. The schedule of a resource is the set of sessions in which this resource appears. In this way, it is possible to consider the schedule for a room in the same way as that of a teacher.

A booking corresponds to an option placed on the occupation of a resource. In relation to a session, a booking is not associated with a teaching module. Bookings give flexibility to the use of the tool, especially during the timetable creation stage performed by the various managers. For example, a room can be booked by a manager for a group, without knowing exactly which teaching module will take place in it. The other managers are aware of the booking and will avoid using the room. If they do use the room, the room manager will have to find an alternative by suggesting another room.

2.5 Description of Constraints

The constraints to be respected can be classified into two groups: physical constraints (also called hard constraints) and preference constraints (or soft constraints) which are linked to the pedagogical quality of the timetables. The physical constraints make it possible to be sure that any particular timetabling problem has a solution (resolution) whereas the preference constraints are generally taken into account when a solution is being improved (optimization).

Physical Constraints. These constraints cannot be violated, otherwise clashing situations would arise. There is a physical resource clash between two sessions s1 and s2 if these sessions have a common resource for a non-null duration.

The physical constraints we have studied in `VT` are the following:

- No resource r may be concerned by two different sessions s_1 and s_2 at the same slot (time and date): $\forall s_1 \forall s_2 \forall r[((s_1 \neq s_2 \wedge r \in resources(s_1) \wedge r \in resources(s_2)))] \Rightarrow [slotDateTime(s_1) \cap slotDateTime(s_2) = \emptyset]$
  where $resources(s_i)$ is the set of all the resources affected by s_i and

$slotDateTime(s_i)$ indicates date, time and duration of s_i.
As soon as a resource appears in a session, all its daughter resources are also concerned by the same session recursively:
$\forall s \forall r(r \in resources(s) \Rightarrow (\forall x(x \in subresources(s) \Rightarrow x \in resources(s))))$
In the same way, as soon as the resource appears in a session, none of its mother resources may be occupied at the same time (because all of these resources have resources in common). These are constraints linked to the hierarchical structure.

- It is forbidden to put more students than there are places in a room.
- The total duration of the sessions in a teaching module may not exceed the volume planned.
- The calendars are respected for all the resources (sessions may only be placed during resource “working” days).

These constraints are part of VT: they cannot be modified by the users but they can be parametrized or can be ignored. Thus, in all the orders available to the user, only actions which respect the constraints can be performed. For example, when a user wants to allocate a room to a session, only the rooms which respect all the constraints (free, suitable size and type) are suggested. This way of proceeding makes it possible, amongst other things, to avoid clashes appearing. We will discuss this further in the paragraph concerning the treatment of clashes. The user can decide to use the automatic timetabler and then to modify the result.

Preference Constraints. Preference constraints may be violated: in this case, the timetable obtained is of a lower quality. Typically, these constraints are used to express what a “good” timetable should be for the students and for the teachers.

Two preference constraints have been studied in VT. The first one is used to limit the resources moving between rooms. The second one allocates to a session the room for which the capacity is near to the number of students. For example, in VT we avoid allocating a very large room to a small set of students.

3 Visualization of the Timetables and Points of View

The two types of view most commonly used are: the weekly graphical view and the yearly graphical view. In both cases, visualizing a timetable consists in representing sessions of one or several resources on a plan. On every view of a timetable, two axes are represented: the resource axis and the time axis. It should be noted that there are no restrictions concerning the resources which appear on the resource axis. In other words, it is possible to visualize on one screen the timetables of several groups, teachers, rooms and elements of equipment.

This has two major advantages. Firstly, during the modification phase, the user is informed on one screen about the availability or occupation of the resources he/she is dealing with. Secondly, it enables the user to understand why the tool refuses to place a session in the slot he/she has chosen (role of explanation).

The handling orders are given via screens. They are invariably available in each view. The orders allow the user to "navigate" around the timetables and modify them whilst controlling clashes (and avoiding them as far as possible). In order to make the user's task easier, the constraints are taken into account in a dynamic fashion. Thus, for example, when a session is being moved, the tool begins by finding the moves allowed and then suggests them to the user. This approach limits the user's search space to the areas in which sessions can be placed without creating clashes.

We defined the notion of points of view in a previous research on co-operative reasoning and its application to interactive diagnosis [16]. The basic idea consists in representing a model according to different angles of vision, called points of view. Each point of view only concerns a part of the model, but the set of points of view covers the entire model. The reasoning performed using each point of view is propagated to the other points of view, and this guarantees the coherence of the set of points of view.

The notion of point of view is used here to represent various elements of information for the same indicator (time, resource) without overloading the screen. The points of view retained here are: the resource point of view (teacher, group, room and equipment), the teaching module point of view and a specific point of view in which the user chooses the information he/she wishes to see. In this last point of view, the user can see for example the names of the rooms, their capacity and their occupation rate as well as the names of the teachers who use the rooms in question. A button enables the user to switch from one point of view to another whilst keeping the general appearance of his/her view along with the same orders (addition, removal, etc).

Figure 2 shows part of the schedule for the $DESS\_ICHM$ class and its subgroups $ICHM1$ and $ICHM2$, from the point of view of the teachers and the rooms for a period chosen by the user. It is a yearly view where each week is represented by one line. Each line has as many sub-lines as resources selected (here the resources are the groups: $DESS\_ICHM$, $ICHM1$ and $ICHM2$). The name of the teacher for each visualized session is displayed (teacher's point of view).

3.1 Action Possible Using the Graphic Interface

We have defined three types of user: the designers (pedagogical managers), the analysers (administrative managers) and the others (teachers, students: only for seeing the timetables). They have specific needs which lead to the definition of specific action classes. This action classes are described in [17].

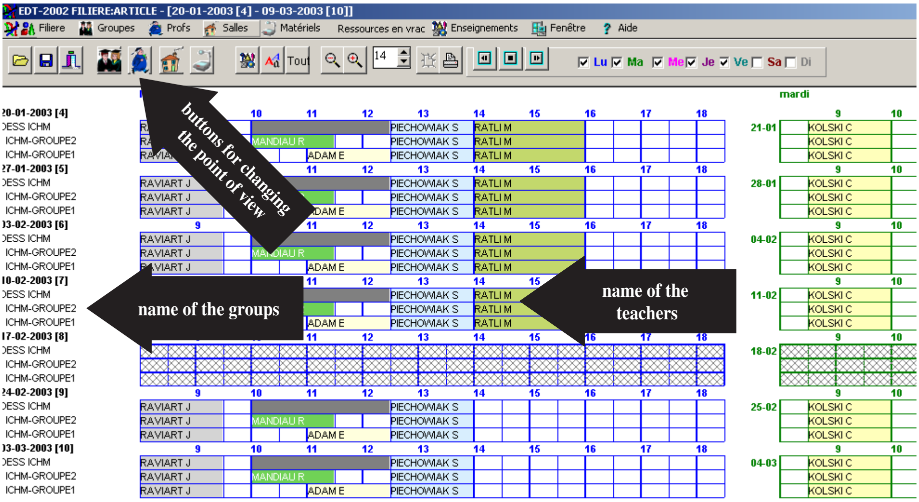


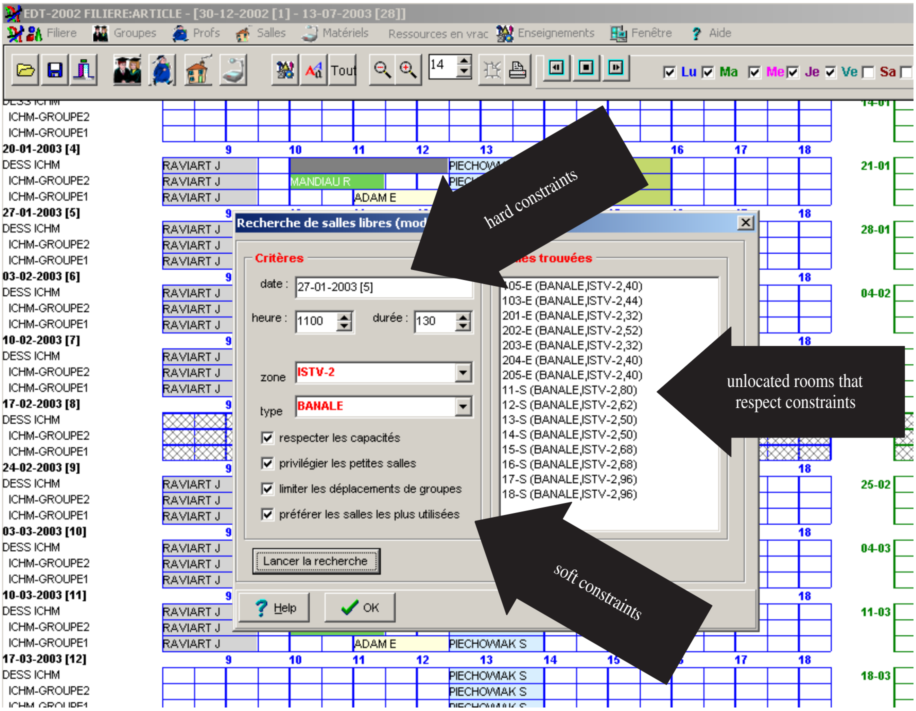
Fig. 2. A timetable seen from the teacher point of view

3.2 Allocation of Resources in an Existing Schedule: Example of the ROOMS

The allocation of rooms is performed by the room manager once the timetables have been created by the educational managers. The rooms are divided into several types: lecture theatres, ordinary rooms, specialized rooms. They are characterized by their capacity as well as by their geographical location. The location is limited to proximity zones because the important thing is to know whether one room is close to another or not, more than the actual distance between the two.

Up to now, the manual allocation of rooms has been carried out without too many problems thanks to the experience acquired over the past 20 years by the various managers. Moreover, the allocation of rooms rarely leads to clashes which bring about considerable modifications in the timetables. This can be explained mainly by the applied approach. Before the educational managers even begin developing the timetables, they meet together to pinpoint the critical teaching modules (i.e. those which need critical resources—rooms or equipment). These teaching modules are placed on the various timetables in such a way as to satisfy all requirements and avoid clashes. For example, the lecture theatres are allocated to the various courses so as to cover global needs. This is done using the booking notion explained previously. Thus, for example, for a specific group, a lecture theatre is reserved for four half-days every week from 15th September to 30th May.

When the rooms are actually allocated to sessions, the user must choose a room from all the rooms so that the constraints are respected: the room must be free and have a sufficient capacity to accommodate the students concerned by the session. Extra constraints (preference constraints) must also be taken


EDT-2002 FILIERE:ARTICLE - [30-12-2002 [1] - 13-07-2003 [28]]
Filiere
Groupes
Profs
Salles
Matériels
Ressources en vrac
Enseignements
Fenêtre
Aide
Tout
14
☑ Lu ☑ Ma ☑ Me ☑ Je ☑ Ve □ Sa □
hard constraints
unlocated rooms that respect constraints
soft constraints

Recherche de salles libres (mod

Critères

date : 27-01-2003 [5]

heure : 1100 durée : 130

zone ISTV-2

type BANALE

☑ respecter les capacités

☑ privilégier les petites salles

☑ limiter les déplacements de groupes

☑ préférer les salles les plus utilisées

…les trouvées

105-E (BANALE,ISTV-2,40)
103-E (BANALE,ISTV-2,44)
201-E (BANALE,ISTV-2,32)
202-E (BANALE,ISTV-2,52)
203-E (BANALE,ISTV-2,32)
204-E (BANALE,ISTV-2,40)
205-E (BANALE,ISTV-2,40)
11-S (BANALE,ISTV-2,80)
12-S (BANALE,ISTV-2,62)
13-S (BANALE,ISTV-2,50)
14-S (BANALE,ISTV-2,50)
15-S (BANALE,ISTV-2,68)
16-S (BANALE,ISTV-2,68)
17-S (BANALE,ISTV-2,96)
18-S (BANALE,ISTV-2,96)

Fig. 3. Dialogue window for the allocation of rooms with consideration of preference constraints

into account not only for the quality of the timetables but also in order to anticipate needs which are unknown when the timetables are developed (for example, organization of work meetings or seminars).

We suggest helping the user to decide which room to allocate to which session in the following manner. Firstly, a list is drawn up of the free rooms which have a sufficient capacity to accommodate the group of students. Then these rooms are classified using a function which depends on preference constraints or heuristics. Thus the user can either trust the tool and choose the first room in the classified list, or choose another room on the list for reasons which are not known to the tool. The parameters of the function used to classify the rooms in the list can be chosen by the user who can decide to apply all of the preference constraints or just some of them.

The preference constraints currently in use were defined in collaboration with the room managers: regularity of room allocation, limitation of group movement between rooms and anticipation of unknown events.

The user can decide whether to take preference constraints into account or whether to ignore them, as is shown in the dialogue window in Figure 3. In addition, this user can decide to use the automatic allocation module. This module is based on intelligent backtracking algorithms like *backjumping* and *forward checking* (we know that these exact methods are not the best for a full automated approach, but they are sufficient in our situation) [19].

4 Treatment of Clashes

In the section concerning constraints, we defined the notion of clashes: a clash is characterized by the sharing of resources between several sessions at the same moment in time:

$$\begin{aligned}\forall s_1 \forall s_2, clash(s_1, s_2) \Leftrightarrow ((s_1 \neq s_2) \wedge (overlapped(date(s_1), date(s_2))) \\ \wedge (resources(s_1) \cap resources(s_2) \neq \emptyset))\,.\end{aligned}$$

The basic idea of our work is to design an interactive decision support tool which will allow the planning of sessions on a schedule whilst guaranteeing that there are no clashes. Nevertheless, several reasons oblige us firstly to accept that there will be clashes and secondly to resolve them.

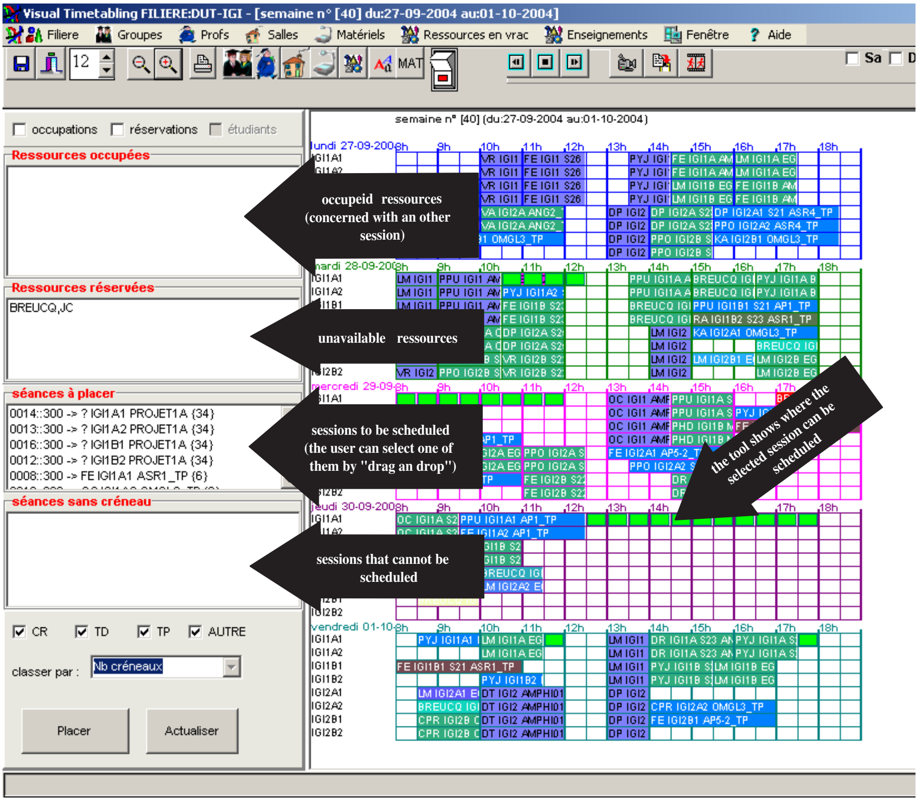
Firstly, during our on-site tests with the users, it appeared that during the development of timetables, some users prefer to have the possibility of placing clashing sessions temporarily and then correcting these clashes, rather than only being able to consider non-clashing situations.

Secondly, in reality, the timetables are created locally for the branches, without knowing the schedules for other branches. When all the schedules for all these branches are grouped together, clashes generally appear because of teachers who work in several branches for example.

Finally, our tests revealed that even for users who wish to avoid clashes when placing sessions, it is important for them to be able to be informed of the reasons for the impossibility of placing certain sessions. For example, when the user clicks with the mouse on a grid in order to place a session, the tool begins by searching for a list of sessions which may be placed without creating clashes. If there is nothing in this list, it is better to indicate why that is so. Two explanations can be given. The first is that all the sessions have been placed. The second explanation is that for all the sessions which still have to be placed, there is systematically one of the resources which is already used for another session at the same time. In this case, it is best to give to the user the list of sessions which hinder the placing; the user can then decide whether to rethink the choice of place or whether to return to sessions placed beforehand.

4.1 First Approach: Avoiding Clashes

The method which consists in avoiding clashes is well adapted when the timetable is available and the user wishes to modify it slightly, or when it is available on paper and the user wishes to enter it on a computer. Avoiding clashes then comes down to avoiding data entry errors. The users who appreciate this method are the room managers because they allocate the rooms to the sessions, once the


Visual Timetabling FILIERE:DUT-IGI - [semaine n° [40] du:27-09-2004 au:01-10-2004]

semaine n° [40] (du:27-09-2004 au:01-10-2004)

□ occupations □ réservations □ étudiants

Ressources occupées

Ressources réservées

BREUCQ,JC

séances à placer

0014::300 -> ? IGI1A1 PROJET1A {34}
0013::300 -> ? IGI1A2 PROJET1A {34}
0016::300 -> ? IGI1B1 PROJET1A {34}
0012::300 -> ? IGI1B2 PROJET1A {34}
0008::300 -> FE IGI1A1 ASR1_TP {6}

séances sans créneau

☑ CR ☑ TD ☑ TP ☑ AUTRE

classer par : Nb créneaux

occupeid ressources (concerned with an other session)

unavailable ressources

sessions to be scheduled (the user can select one of them by "drag an drop")

sessions that cannot be scheduled

the tool shows where the selected session can be scheduled

Fig. 4. A weekly view for sessions scheduling

sessions have already been scheduled. Figure 4 shows a weekly view of session placement. On the left is a list of the sessions that should be placed in this week. To the right, the user can see where to place the session he/she has selected.

This method is also advantageous for the creation of timetables because it constantly limits the space of the sessions which can be placed in a given slot. In this case, this method alone is not sufficient. We noted that when wanting to place a session in a slot, the user is put off if the session he/she thinks he/she can place is not on the list of possible sessions. To solve this problem, the user would like to be given the reasons for this absence (teacher, room, group or one of the descendants or ascendants already occupied). Another way to tackle this problem is to accept clashes and to resolve them afterwards. The danger of this method is that there could be a lot of clashes which would be impossible to resolve without starting the work again from the beginning.

By allowing the manipulation of incomplete sessions (no teacher, no room), a different approach is adopted which consists in creating a timetable without resources (or the fewest resources possible) and then allocating the resources

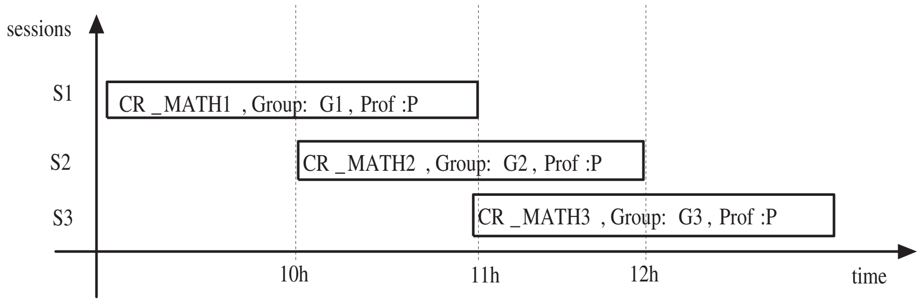

sessions
S1
CR _MATH1 , Group: G1 , Prof :P
S2
CR _MATH2 , Group: G2 , Prof :P
S3
CR _MATH3 , Group: G3 , Prof :P
10h
11h
12h
time

Fig. 5. Example of clash with groups

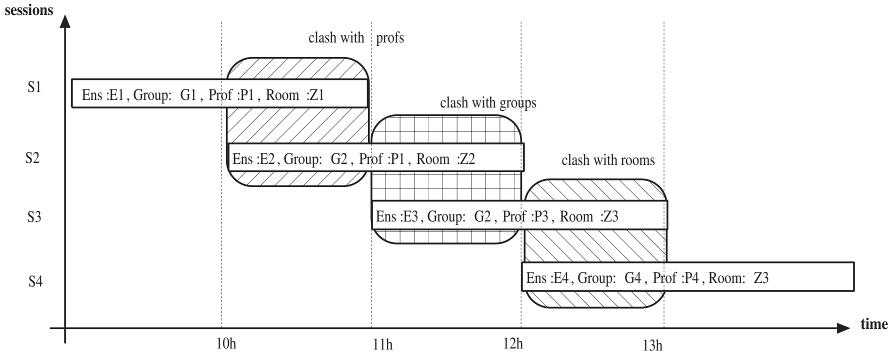

sessions
clash with profs
S1
Ens :E1 , Group: G1 , Prof :P1 , Room :Z1
clash with groups
S2
Ens :E2 , Group: G2 , Prof :P1 , Room :Z2
clash with rooms
S3
Ens :E3 , Group: G2 , Prof :P3 , Room :Z3
S4
Ens :E4 , Group: G4 , Prof :P4 , Room: Z3
10h
11h
12h
13h
time

Fig. 6. Example of a multiple clash

afterwards. The problem with this approach is that the constraints are not integrated early enough. At the ISTV, it is this approach which is used, since the rooms are allocated once the timetables have been created.

These two approaches can be compared to the prospective and retrospective approaches which are well known in the field of constraint programming [19]. It is shown that the approaches which involve forward checking through constraints are more efficient than those which check if constraints have been respected after the choices have been made.

4.2 Second Approach: Interactive Resolution of Clashes

A clash can be resolved in several different ways.

The first way is to delete certain clashing sessions and to request the user to place them elsewhere on the schedule. This method is only viable if the dependencies between clashes are taken into account and if the number of clashes is reasonable. For example, let us consider the case shown in figure 5 in which the sessions $S1$, $S2$ and $S3$ are scheduled on the same day in the same slot.

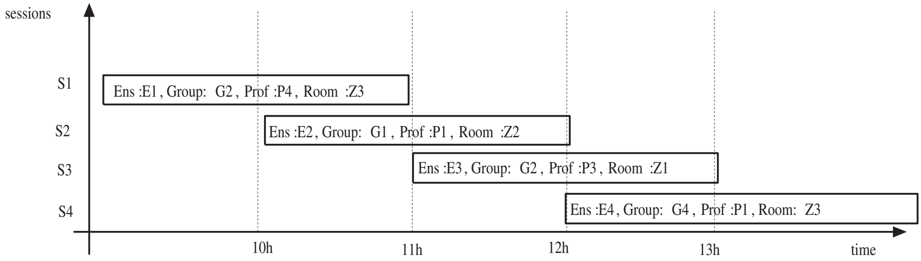

sessions
S1
S2
S3
S4
Ens :E1, Group: G2, Prof :P4, Room :Z3
Ens :E2, Group: G1, Prof :P1, Room :Z2
Ens :E3, Group: G2, Prof :P3, Room :Z1
Ens :E4, Group: G4, Prof :P1, Room: Z3
10h
11h
12h
13h
time

Fig. 7. The clashes have been resolved

In this case, two clashes are linked: $S1$ with $S2$ and $S2$ with $S3$. The clashes can be resolved by deleting the three sessions and trying to place them elsewhere, by deleting $S1$ and $S3$ and placing them elsewhere, or by deleting $S2$ only and by placing it in another slot. The choice of solution can depend on various criteria. It is possible to choose the solution which modifies the fewest sessions (just $S2$ in the example). It is also possible to choose the solution which preserves the quality of the timetables.

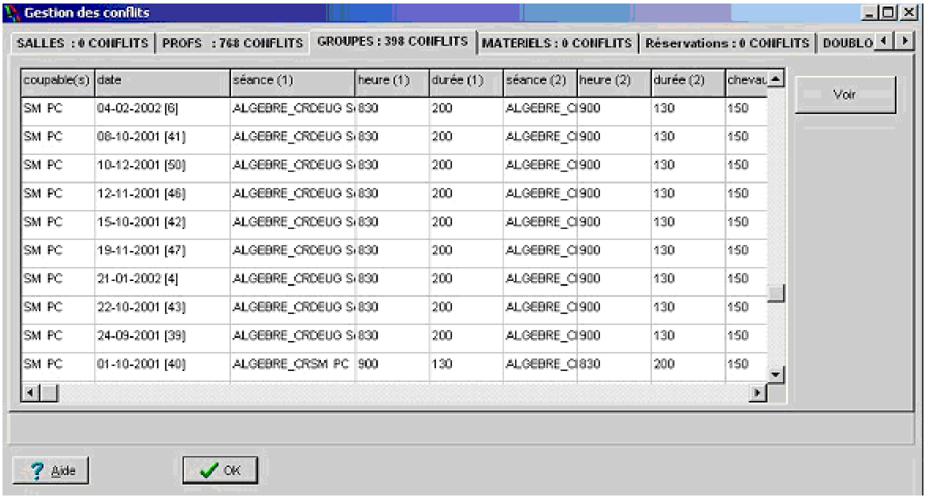
The second way consists in grouping together the resources involved in different sessions. The teaching modules may be shared by several groups in different branches. When the data relating to the different branches is merged, these teaching modules are declared several times. This can cause clashes with the teachers, for example, because for these shared teaching modules, the same teacher gives a lesson to different groups at the same time. The resolution of these clashes is easy: the groups in the two teaching modules have to be merged.

The third way of resolving clashes is to reduce the duration of the sessions so that the time interval associated to the clashes is eliminated. In the previous example, the two clashes take place in the time intervals [10.00–11.00] and [11.00–12.00]. There are several possibilities : reduce session $S1$ to the [09.00–10.00] interval and session $S3$ to the [12.00–13.00] interval or reduce session $S1$ to the [09.00–10.30] interval, session $S2$ to the [10.30–11.30] interval and session $S3$ to the [11.30–13.00] interval.

The fourth way is to eliminate the “cause” of the clash, that is the teacher, the group or the room. Once the clash has been eliminated, an attempt can be made to allocate the sessions to other teachers, groups or rooms.

This method can be extended so that it is capable of exchanging the resources for various sessions. For example, the teachers of two teaching modules can be interchanged. More complicated exchanges can also be envisaged, involving more than two teachers. However, searching for these exchanges with an aim to resolving clashes is a laborious process. In Figure 6, an example of a multiple clash is shown.

By performing the following exchanges: $prof(S1) \leftrightarrows prof(S4)$, $group(S1) \leftrightarrows group(S2)$ and $room(S1) \leftrightarrows room(S3)$, the three clashes are resolved and the configuration shown in Figure 7 is obtained.



coupable(s)	date	séance (1)	heure (1)	durée (1)	séance (2)	heure (2)	durée (2)	chevau
SM PC	04-02-2002 [6]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	08-10-2001 [41]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	10-12-2001 [50]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	12-11-2001 [46]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	15-10-2001 [42]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	19-11-2001 [47]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	21-01-2002 [4]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	22-10-2001 [43]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	24-09-2001 [39]	ALGEBRE_CRDEUG S	830	200	ALGEBRE_C	900	130	150
SM PC	01-10-2001 [40]	ALGEBRE_CRSM PC	900	130	ALGEBRE_C	830	200	150

Fig. 8. Window displaying the clashes detected

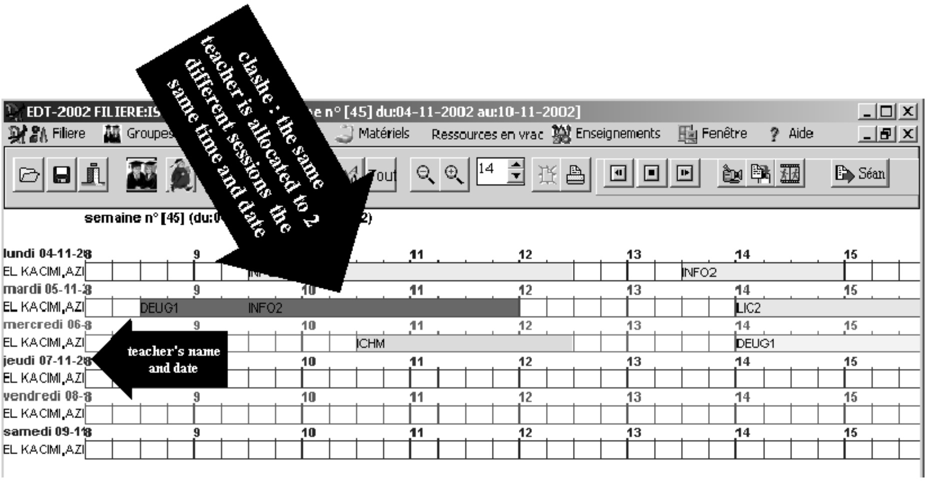
Trials were carried out in real situations. We have worked with the manually made timetables for the academic year 1999–2000. The search for clashes amongst the 2500 sessions took three seconds using a PC with a Pentium III 500Mz processor, 256M ram. This search revealed 350 group clashes, 50 teacher clashes and 30 room clashes.

It should be noted that this test was performed using real data from the manually-made timetables used for the university year, which should in theory have been clash-free! This mean that clashes was not detected during the conception phase of timetables: they have been treated during the year by teachers themselves.

Figure 8 shows how the clashes are displayed (group clashes in the case of the figure). There is a nine-column chart (the ninth column is hidden because of the size of the window). Each line represents a clash. For example, the first line shows a clash involving the *SMPC* group on 04-02-2002. In fact there are two overlapping sessions. The first one, *ALG_CR_DEUG*, starts at 08.30 and lasts for 2 hours, while the second, *ALG_CR_DEUG*, starts at 09.00 and lasts for 1.30 hours. The overlap is therefore 1.30 hours. By using the *See* button, the user sees the schedule for this group for the entire week which shows up these two sessions. The user can then decide how to correct the clash (move the sessions, eliminate them, etc).

Figure 9 shows the view a user can have when he/she has decided to request a graphical view of a clash (the first clash in our example). The two clashing sessions are shown up by a colour. It is necessary to modify (eliminate, move or reduce) the duration of one of the two sessions to resolve the clash.

The user can view the schedule for a set of resources, showing the teachers as well as the groups, rooms and equipment. This type of view is particularly


clashe : the same teacher is allocated to 2 different sessions the same time and date

teacher's name and date

Fig. 9. Visualization of one of the clashes

well-suited to understanding why the choice of a place or move of session is impossible. For example, let us suppose that the graphical view shown concerns the timetable of a group G1 and that the user wishes to move a session S for this group. By clicking on the session, the user can choose the instruction move in the day. VT calculates the list of slots corresponding to the possible moves. If the user does not find the slot he/she had been thinking of *a priori* on this list, the tool enables him/her to understand the reasons why by providing the occupation of each resource involved in the session.

4.3 Semi-automatic Resolution of Clashes

The problem of resolving clashes can be tackled in a generic way, but experience has led us to agree with the view of Foulds and Johnson [**?**]: the clashes generally concern few sessions and can be resolved by backtracking a little way.

Clashes are detected according to each type of resource (teacher, group, room and equipment) and are classified according to different criteria. These criteria were defined on the basis of the experience of the various educational managers.

- The clash represents a *double*. Two sessions are a double (session duplication) if they are sessions in the same teaching module (so they have the same teachers and the same groups, but not necessarily the same equipment or the same rooms) or if they take place on the same day, at the same time and for the same duration. This problem is solved by deleting one of the sessions. If the total volume of sessions of this teaching module is not equal to the total volume planned, the user will have to place the deleted session elsewhere.
- Sharing of a teaching module over several groups. For example, conferences are organized for two courses represented by G1 and G2. The managers

of these courses create their timetables. Then, when all the timetables are collated for the allocation of rooms and detection of clashes, the conferences are detected as being clashes because they share the same teacher, on the same date, at the same time and for the same duration. Experience shows that these cases of teaching modules shared over several courses is done with small groups and according to the level (first, second or third cycle). The problem is solved by deleting one of the two teaching modules and adding the teaching module groups from the deleted module to the groups of the remaining module.

- Resolution of clashes by giving preference to moving sessions involving small groups and short sessions (in duration).

5 Conclusion and Future Work

In this article, the problem of timetable management has been looked at from a viewpoint centred on the user, and not, as in many other research projects, centred on the problem (or on the algorithms available). This led us to design the VT decision support tool which helps the user when faced with solving a problem. The tool has been tested successfully at the University of Valenciennes and Hainaut-Cambrésis in large-scale real situations.

VT is intended for different user profiles, going from the designers of timetables to users who have varied degrees of computing skills. It makes it possible to manage resources organized into a hierarchy (groups, teachers, rooms and equipment). It guarantees data coherence in a dynamic way. When clashing situations appear, various resolution approaches are suggested.

The aim was to design a generic interactive decision support tool for timetable problems on the basis of a real case study which was truly representative of the problem. An object-oriented approach was chosen and the generic classes have been defined like in [22].

Work is now continuing in a main direction: the aim will be to help the user to express constraints and to take these constraints into account in an interactive manner. For this, we intend to revise the work performed on constraint programming to provide support in the design and generating of timetables. In this field, most existing work deals with automated generation. On the other hand, very few research projects focus on support in the description of timetables [6], [2], [12] or [18]. We feel that an interesting project would be to combine the automated and interactive generation of timetables. This direction was explored successfully in [1] and [9] which deal with other problems based on scheduling. For example, we envisage performing an automatic allocation of rooms after the timetables have been created interactively with the users (thus making it easier to take into account the pedagogical constraints which are difficult to express and formulate). A project similar to [7] or [13] based on approaches from the constraint programming field has already been started on this subject.

References

1. Abdennadher, S., Schlenker, H.: INTERDIP, an Interactive Constraint Based Nurse Scheduler. Proc. PACLP'99 (London)
2. Burke, E. K., Kingston, J. H., Pepper, P. A.:
3. Boizumault, P., Delon, Y., Peredy, L.: Constraint Logic Programming for Examination Timetabling. J. Logic Program. **26** (1996) 217–233
4. Carter, M. W.: A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 64–84
5. Carter, M. W., Laporte, G., Chinneck, J. W.: A General Examination Scheduling System. Interfaces **24** (1994) 109–120
6. Cooper, T. B., Kingston, J.: A Program for Constructing High
7. Deris, S., Omatu, S., Ohta, H.: Timetable Planning Using the Constraint-Based Reasoning. Comput. Oper. Res. **27** (2000) 819–840
8. Eglese, R. W., Rand, G. K.: Conference Seminar Timetabling. J. Oper. Res. Soc. **38** (1987) 591–598
9. Goltz, H. J., Matzke, D.: Combined Interactive and Automatic Timetabling. Proc. PACLP'99 (London)
10. Goltz, H. J.: On Methods of Constraint-Based Timetabling. Proc. PACLP'00 (Manchester)
11. Johnson, D. G.: Timetabling University Examinations. J. Oper. Res. Soc. **41** (1990) 39–47
12. Kingston, J. H.: Modelling Timetabling Problems with STTL. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 433–445
13. Legierski, W.: Search Strategy for Constraint-Based Class-Teacher Timetabling. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 247–261
14. McCollum, B., Ahmadi, S., Burke, E., Barone, R., Cheng, P., Cowling, P.: A Review of Existing Interfaces of Automated Examination and Lecture Scheduling Systems. In: Burke, E., De Causmaecker, P. (eds.): PATAT 2002—Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling 262–264
15. Müller, T., Bartak, R.: Interactive Timetabling. In: Proc. ERCIM Workshop on Constraints (Prague, 2001)
16. Piechowiak, S., Jouglet, D., Vanderhaegen, F.: A Multi-Point of View for Phone Network System Diagnosis. Proc. ICSSSE'98 (Beijing)
17. Piechowiak, S., Kolski, C.: Analyse et Conception d'un Outil Interactif d'Aide à la Gestion des Emplois du Temps Basé sur les Points de Vue (in French). J. Human–Computer Interaction, in press
18. Reis, L. P., Oliveira, E.: A Language for Specifying Complete Timetabling Problems. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 322–341
19. Tsang, E.: Foundations of Constraint Satisfaction. Computation in Cognitive Science Series. Academic, New York (1993)

20. Willemen, R.: School Timetable Construction: Algorithms and Complexity. Ph.D. Thesis. Technishe Universiteit, Eindhoven, The Nederlands (2002)
21. Yoshikawa, M., Kaneko, K., Nomura, Y., Watanabe, M.: A Constraint-Based Approach to High-School Timetabling Problems: A Case Study. Proc. 1995 Int. Joint Conf. on AI (IJCAI'95, Montreal)
22. Zervoudakis, K., Stamatopoulos, P.: In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 28–47

Distributed Choice Function Hyper-heuristics for Timetabling and Scheduling

Prapa Rattadilok, Andy Gaw, and Raymond S.K. Kwan

School of Computing, University of Leeds, UK
{prapa, gaw, rsk}@comp.leeds.ac.uk

Abstract. This paper investigates an emerging class of search algorithms, in which high-level domain independent heuristics, called hyper-heuristics, iteratively select and execute a set of application specific but simple search moves, called low-level heuristics, working toward achieving improved or even optimal solutions. Parallel architectures have been designed and evaluated. Results based on a university timetabling problem show an important relationship between performance, algorithm software and hardware implementation.

1 Introduction

Hyper-heuristics are a powerful emerging search technology [6]. Search algorithms are often constructed from a collection of simple neighbourhood moves referred to as low-level heuristics. Rather than hard-wiring such simple moves, hyper-heuristics employ a domain independent driver that iteratively makes dynamic decisions on which simple move(s) should be executed next. The selected heuristics can be knowledge-poor heuristics like simple add, drop and swap moves or complete algorithms more akin to meta-heuristics. Cowling et al. [15], [16] and Soubeiga [27] proposed a choice-function-based hyper-heuristic driver, which has components designed for search intensification and diversification and incorporates some simple learning capability. Selection is made based on three factors: the previous performance of the heuristic, the performance of successive pairs of heuristics and the time since the heuristic was last used.

Hyper-heuristics have been successfully used to solve scheduling problems and have also been shown to provide a useful tool in rapid prototyping of optimised systems [6], [14], [15], [16]. Burke et al. [9] and Petrovic and Qu [22] show the success of a hyper-heuristic approach that uses case-based reasoning, while the later paper of Burke et al. [7] uses tabu search to address this specific type of problem.

Burke et al. [9] use case-based reasoning, by taking the benefits of using relevant cases from memory and adapt them to fit new situations to apply in heuristic selection. The selected heuristics for the problem in hand are selected based on the best (or reasonably good) heuristics used from solving similar problems. The result shows that after a training session, retrieval accuracy increases. Petrovic and Qu [22] studied the importance of features and their weightings in

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 51–67, 2005.

the knowledge discovery process. They found that, in retrieval, the feature selection of each source case is more important than the feature weight, when the features are elaborate enough. The number of features also affects system performance. Too many features cause a slow-down because of the confusion within the system in finding a good source case for prediction when the source cases and the training cases are too close to one another. For the system to perform very well, a large training set is needed. Another approach involves the use of a learning classifier system that associates the problem states with the heuristics. The approach tries to find powerful combinations of heuristics, it has also been successfully used to tackle a bin packing problem [24], [25] and an examination timetabling problem [28].

Burke et al. [7] combine tabu search with heuristic selection, by maintaining a tabu variable length list of low level heuristics, which excludes some low-performance heuristics from the selection for a certain duration. The results show that for small/medium size problems it performs as well as or better than tailor-made algorithms, while better results are achieved from an Ant algorithm for bigger instances. Burke et al. [8] adapt the single objective hyper-heuristic approach in [7] to multi-objective optimisation. The paper compares the performance of different objective selection methods (random or roulette wheel) and different design of tabu lists (one for every objective or one for each objective). The experiment results show that the approach performs well on a wide range of problems with narrow gaps between the best and worst solutions obtained.

Kendall and Mohd Hussin [20], [21] show more recent work on tabu-based hyper-heuristics. Similar to [7], hyper-heuristics select a non-tabu low-level heuristic to apply at any iteration. Kendall and Mohd Hussin [20] studied the tabu criteria based on CPU time, the tabu element was based on changes in a penalty function and probabilistic heuristic selection. The low-level heuristic that is used in the current iteration will be tabu for a certain duration (0–4 iterations) to allow other low-level heuristics to perform. The experiments show that a tabu duration of two is the best duration. Kendall and Mohd Hussin [21] show the result of the approach on a university timetabling problem.

Cowling et al. [13] show another example of combining a hyper-heuristic with a meta-heuristic. The approach uses a genetic algorithm as the high level selector to evolve the order of the low-level heuristics for a personnel scheduling problem with each gene in the chromosome representing a low-level heuristic call. The results show that the hyper-GA is able to evolve a suitable sequence for a number of different problem instances. An adaptive length chromosome with guided operation is used in their later work [18]. The evolution process evolves a suitable sequence without explicit concern on chromosome length; poor-performing heuristics can be removed and/or efficient heuristics can be added. Their strategies of when to inject or remove are based on the length of the chromosome, if it is too long then poor blocks of genes will be removed, if it is too short then fit blocks of genes will be injected. Han and Kendall [19] extend their previous work by adding tabu search to increase the efficiency of each low-level heuristic call. The gene is penalised when it does not make any change to the objective function. The penalty is in the form of forbidding a poor-performing low-level heuristic to be executed for a certain number of iterations. The results are comparable to their guided operator hyper-GA, but using less processing time.

Burke et al. [5], [10] discuss an alternative to simple low-level heuristics, the graph heuristics, for tabu-based hyper-heuristics. The graph heuristics provide ways to order events that are not yet scheduled based on the difficulties of scheduling them into a feasible timeslot, for example, they can be ordered based on the number of enrolment, conflict or feasible slots. Burke et al. [10] use a tabu-based hyper-heuristic to guide the sequence of these graph heuristics. Burke et al. [5] compare a tabu-based hyper-heuristic to select graph heuristics with the hybrid approach of also using case-based reasoning. The results show better performance when using a tabu-based approach. In graph heuristics, events are ordered based on some measurement of their difficulties recorded in matrices, Burke and Newall [11] propose an adaptive approach in estimating these difficulties by experience, basing its consideration on constraint-based strategies: hard constraint only, or both hard and soft constraint.

Ayob and Kendall [3] apply Monte Carlo acceptance criteria for new solutions. At any iteration, a low-level heuristic is selected randomly to modify a given solution. The Monte Carlo acceptance criteria always accept an improved solution, though a worse solution will be accepted probabilistically. The three Monte Carlo strategies discussed are linear, exponential, and exponential with a counter of consecutive non-improvement iteration. The experiment shows that the performance of the linear Monte Carlo approach is the most parameter sensitive. The exponential with counter approach was also found to give equally good performance and it is parameter-free, thus it is a more robust approach.

Many sequential scheduling algorithms have been extended to new parallel versions. The algorithmic parallelisation techniques are aimed either at reducing the time consuming processes of the algorithm or at increasing the search space coverage. Cost function calculation and neighbourhood transition have been distributed onto multiple processors for algorithms including simulated annealing, genetic algorithms, ant algorithms and tabu search, to speed up the execution of the algorithm [1], [2], [4], [23]. The simplest form of parallelism that increases the search coverage would be to execute the algorithm on different processors using different random seeds or starting solutions. Interactions of these processors, e.g. transferring good solutions to other processors, are found to increase the overall fitness, with the added cost of a communication [23]. Cantu-Paz [12] lists several topologies for the interaction of processors in a genetic algorithm. De Falco et al. [17] describe the technique where partial solutions are explored on processors and combined at the master processors. Although this technique reduces the communication overhead, this search dividing process becomes too problem specific. Hyper-heuristic methods, comprising a very large number of simple steps, favour parallelisation by increasing the search coverage, e.g. carrying out multiple low-level heuristics in parallel.

The two key aims of the research are the developments of a general hyper-heuristic framework and of distributed algorithms. The general framework simplifies the development of effective low-level heuristics, and permits their reuse in different problem applications. The distributed algorithm expands the search possibilities, increasing both the search efficiency and its coverage, which can be limited in sequential algorithms. This paper reports on on-going research in the design of choice function hyper-heuristics, the modelling of general-purpose low-level heuristics, and the exploitation of parallel computing platforms for hyper-heuristics.

The University Timetabling Problem is concerned with the production of good quality class timetables for universities. An abstract version of the problem was used as the basis for an international timetabling competition, organised by the Metaheuristics Network [30]. Each problem instance comprises a set of events, a set of students, and a set of locations and times at which events may take place. Students each attend a subset of the available events. Events may require specific facilities (e.g. computers, OHPs) to be available in the rooms in which they take place, and these may not be available in all locations. Additionally, the assigned rooms must be large enough to seat all the attending students. No student can be required to attend two or more events simultaneously, and only one event can take place at a given location and time.

A feasible solution to a given problem instance consists of an assignment of each event to a room and time, such that all the above conditions are met.

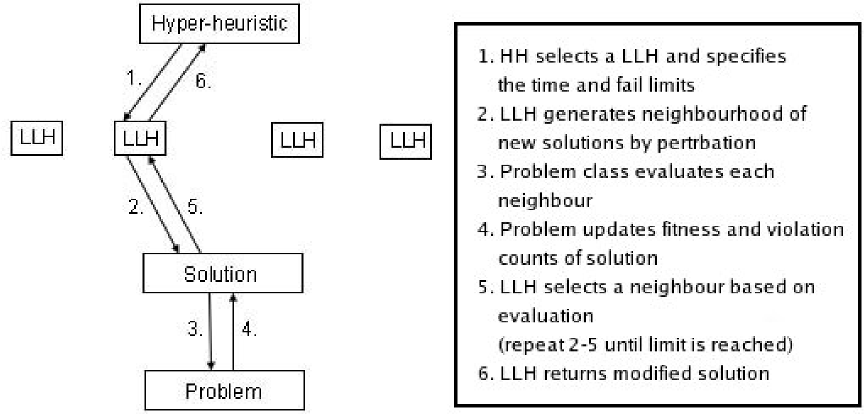
In addition to these hard constraints, there are a number of soft constraints, representing undesirable characteristics of a timetable, which should be minimised. These are: students attending events during the last hour of the day; students attending three events in successive hours of a day; and students attending a single event on any given day. Occurrences of each of these should be minimised, so the number of instances found in a feasible timetable gives a measure of solution quality.

2 Hyper-heuristic Framework

Figure 1 shows an overview of the general framework for scheduling and relationships of its components. The hyper-heuristic applies a low-level heuristic to an existing solution. The quality of the modified solution is determined by the problem class and stored with the solution. An initial solution is required, and this may be generated in a number of ways, including randomly, or using a problem-specific method.

2.1 Solution

A solution represents a set of assignments of resources to demands. An abstract representation of a working solution is used in our framework, which includes meta-data on the state and history of the solution, and the progress of the search. The class provides a number of generic methods for making changes to



1. HH selects a LLH and specifies the time and fail limits
2. LLH generates neighbourhood of new solutions by pertrbation
3. Problem class evaluates each neighbour
4. Problem updates fitness and violation counts of solution
5. LLH selects a neighbour based on evaluation
(repeat 2-5 until limit is reached)
6. LLH returns modified solution

Fig. 1. Overview of hyper-heuristic framework

the solution it represents (to add or drop assignments, or swap resources between two assignments, for example). These are used by low-level heuristics to generate new solutions.

2.2 Low-Level Heuristic

Each low-level heuristic contains a solution perturbing mechanism. It is used to generate a set of new solutions (known as a neighbourhood) from an initial solution by making small changes. Each neighbour is evaluated, and based on these evaluations one will be selected. This becomes the initial solution for the next iteration. The evaluation criteria vary, but best neighbour, or best neighbour not worse than the current solution, are common. At this stage of development, problem-specific low-level heuristics were used.

H1 : Change room of the assignment with the highest number of violations
H2 : Same as H1 but for a random assignment
H3 : Change time of the assignment with the highest number of violations
H4 : Same as H3 but for a random assignment
H5 : Change day of the assignment with the highest number of violations
H6 : Same as H5 but for a random assignment
H7 : Find best slot for a chain of assignments

These low-level heuristics are grouped into two groups: for *Intensification* and for *Diversification*. Intensifying low-level heuristics (H1–H6) only accept a new neighbour that does not have lower quality than the initial one, whereas, the diversifying low-level heuristic (H7) will accept the neighbour as long as no better one can be found, allowing it to move the search away from a local optimum and into new areas of the search space.

2.3 Choice Function

A choice-function-based hyper-heuristic based on that of Soubeiga [27] was developed to provide a ranking of low-level heuristics based on their performance statistics. The performance statistic is calculated using the improvement the low-level heuristic makes to the timetable (the reduction of violations) and the time taken to obtain that improvement. The performance statistics are time-weighted averages, used in order to steer the selection to the best performing low-level heuristics, with an emphasis on recent performance. The choice function uses the information about the individual performance of each low-level heuristic ($f1$), joint performance of pairs of heuristics ($f2$), and the amount of time elapsed since the low-level heuristic was last called ($f3$). At each decision point, the choice function is evaluated for all heuristics, and the heuristic with the highest score is selected for the next iteration:

$$f = \sum (\alpha(f_1) + \beta(f_2) + \gamma(f_3)) \ . \tag{1}$$

In this paper we determined values of α, β, and γ experimentally. It is desirable for the tuning processes of these parameters to be automated, to preserve the problem-independence of the approach. Future work will investigate adaptively changing heuristic parameters during the solving process itself.

3 Distributed Hyper-heuristic

Parallel processing power has been used to speed up search processes for many scheduling algorithms. As discussed, speed is a smaller issue when all low-level heuristic are small and simple moves. Our distributed approach investigates multiple neighbourhood searches for better solutions.

Search sequence is the term used to represent a sequence of solutions obtained from repeated application of low-level heuristics to a starting solution, until the time limit is reached. Within the sequential framework, this single search sequence changes between the intensification and diversification strategies, depending on which low-level heuristic is used. The decision on this transition is crucial and can increase efficiency of the search. Diversifying at the last iteration would be a trivial example of search inefficiency.

Figure 2 demonstrates the importance of the transition between intensification and diversification. In this figure, the rule for intensification is simply to accept only non-worsening solutions (CPU 1, and CPU 3). The diversification strategy accepts worsening solutions when no better solution can be found (CPU 2, and CPU 4). The best solution obtained by CPU 4 is obviously unobtainable if only the intensification strategy is used. Although this combination of strategies gives better solution quality than using any single strategy alone, other combinations might give even better solutions. By selecting the right solution (at the right time) and the right combination of strategies, search coverage and efficiency of processor usage will be increased, thus the chances of finding solutions closer to the global optimum will also be increased.

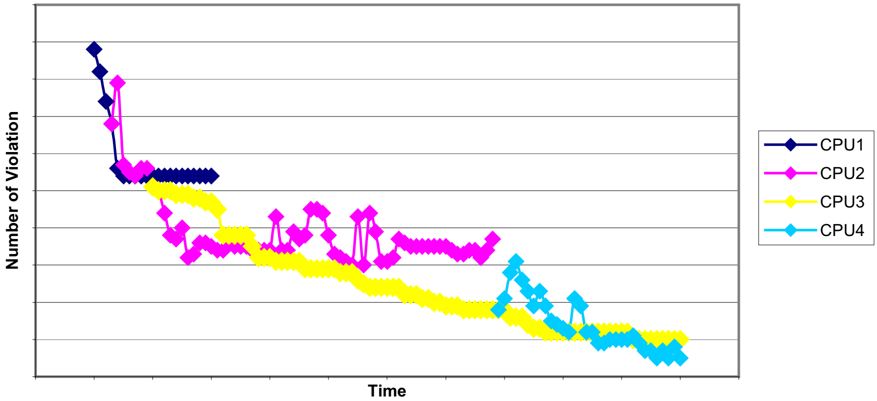


Fig. 2. Branching of the search sequence

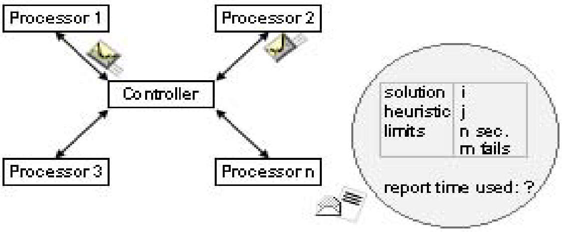


Fig. 3. Task representation and distribution on hierarchical architecture

3.1 Parallel Architectures

In order to investigate the benefit of parallelisation for hyper-heuristics, two parallel architectures were designed: *hierarchical* and *hybrid-agent*. The hierarchical approach aims to investigate the benefits of branching in the search sequence, while the hybrid-agent approach advances the previous investigation on the transfer of good solutions between agents.

In the hierarchical approach, the hyper-heuristic is located on one processor, called the *controller*. The controller distributes tasks to every other processor under its control. The term *task* represents the application of a selected low-level heuristic to a selected solution within the controller's collection, with specified time and failure limits. The failure limit acts as a stopping mechanism when the low-level heuristic cannot improve the solution any further after a number of iterations. Once the task is completed, the *processor* returns the modified solution to the controller. The controller then determines whether to keep or discard the incoming solution.

A number of hierarchical groups are formed as *agents* in the hybrid-agent approach. Each agent is initialised with different starting solutions. Only the

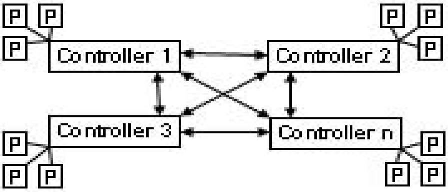


Fig. 4. Hybrid-agent architecture

Controller of each agent is allowed to communicate. The successful agent can request a service of the others via this communication by transferring their good solutions. Theoretically, different agents starting from the same initial solution will produce different results, as their accumulated performance data from previous tasks will differ, resulting in different strategies. The communication rate between controllers is maintained at a regular but infrequent rate to prevent premature convergence of the search.

3.2 Asynchrony and Load Balancing

Each task is given time and failure limits as stopping mechanisms. The Processor applies the low-level heuristic specified within the task description to the given solution. During task execution, failures (iterations not improving the solution) are counted. Tasks terminate on reaching the time or failure limits, and the processor then initiates communication to the controller and returns the completed task.

Processors thus work independently until reaching their specified limits, and initiating the communication with the controller this way makes the process asynchronous. This reduces the amount of idle waiting time when the processing power is varied between processors or if the duration of the search is different.

Load balancing also benefits from the asynchronous design; as long as the task queue at the controller is not empty, all processors will be kept busy. When task results are returned, the sending processor is known to be available, and the controller assigns it a new task from the queue.

3.3 Multiple Branches

Branching of the search sequence (Figure 2), to introduce different strategies (intensification and diversification) can be carried out at any stage of the search, and is used in both architectures. A maximum number of branching points is specified, with the value chosen to maintain the balance of search space coverage and available processing power. Too few branches may not broaden the search sufficiently, whilst too many would reduce the opportunities for finding high-quality solutions.

Experimentally, it was determined that at least one-third of the branches should be allocated to diversification, to maintain a variety of searches and act as an escape mechanism from local optima. We suspect that it would be more beneficial to adjust adaptively the number of diversifying branches. If performance trends in existing branches can be monitored, we could use the information to adjust the allocation appropriately.

The branching points are generated from the diversification branches. Whenever an improving solution is found, an intensification branch is created. While a diversification branch might carry on finding new interesting neighbourhoods, the new branch will be trying to find the optimum of the neighbourhood. When reaching the maximum number of branches, work on less promising branches will be suspended, but maybe resumed later, depending on whether better branches are subsequently found.

4 Results

4.1 Sequential Hyper-heuristic

To give a good understanding of the nature of the scheduling and a better insight into what can be further generalised before the performance suffers, we compare the performance of our generic approach (Table 1) with those in [26] which use max–min ant system (MMAS) and random restart local search (RRLS).

The experiments were run on a processor on the White Rose Grid [29]. parameters α, β, and γ for the choice function are chosen as 0.7, 0.2, and 0.1 respectively. The experiments compare the solving performance of five small problem instances (100 events, five rooms, and five features) and five medium instances (400 events, 10 rooms, and five features).

In term of feasibility, we see that the hyper-heuristic was able to produce a feasible solution for each instance whereas RRLS could not. In particular, RRLS could not produce any feasible solutions in 40 runs on M5. In terms of cost (soft constraint violations) we found that the best solution produced by hyper-heuristic in five runs gave better cost than the average cost of RRLS in most cases.

This approach was compared to a more problem specific algorithm (MMAS). As the problem size increases, MMAS begins to have a performance advantage, as seen from the results of M3– M5. It is suspected that by adaptively tuning the parameters (α, β, and γ), the hyper-heuristic should be able to overcome these small differences in the violation numbers.

4.2 Distributed Hyper-heuristic

The distributed asynchronous hyper-heuristic was tested using the White Rose Grid. The experiments evaluate this design and examine the impact of some important design parameters: number of processors, solution collection size, etc. For these experiments, we used all 20 instances of the PATAT timetabling competition problems [30]. The research indicates several areas in which performance can be improved.

Table 1. Performance comparison between MMAS, RRLS and our hyper-heuristics. In the HH column, the format is: average soft constraint violation for feasible solution / best soft constraint violation in all runs. In HH, feasible solutions were obtained in all five runs. In the RRLS column, 77.5% and 100% correspond to the proportion of infeasible solutions in 40 runs. Otherwise, for both MMAS and RRLS, feasible solutions were found at the end of each run. The best results are shown in bold.

	HH	MMAS	RRLS
S1	2.4 / **1**	1	8
S2	4.9 / **3**	3	11
S3	4.7 / **1**	1	8
S4	4.6 / **1**	1	7
S5	2.6 / **0**	0	5
M1	190.6 / **182**	195	199
M2	185.2 / **164**	184	202.5
M3	262.4 / 250	**248**	(77.5%)
M4	182.6 / 168	**164.5**	177.5
M5	237.8 / 222	**219.5**	(100%)

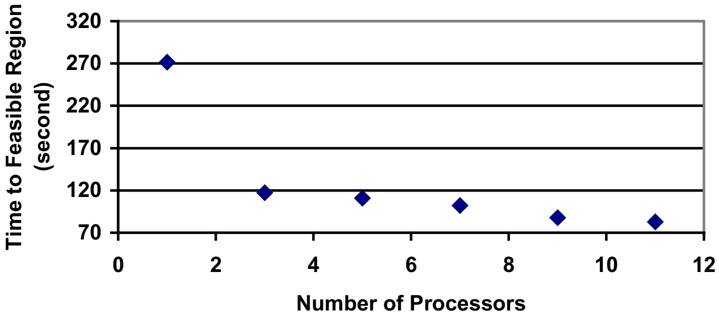


Fig. 5. System performance using different number of processors

The results in Figures 5–10 are the average of five runs, of 15 minutes each, solving the first problem instance provided (400 events, 10 rooms, 10 features and 200 students) where initial solutions are randomly generated.

The first experiment uses the hierarchical architecture with a maximum of three search branches to examine how well the system performs when the number of Processors increases. Figure 5 shows the average time needed to reach a feasible solution. It shows that the performance of the algorithm increases when more Processors are added. The time to feasible solution decreases dramatically when the number of processors increases from one to three but although the time decreases slightly afterward, the improvements are noticeably less dramatic. This could be because of the limited number of search branches and the limited number of low-level heuristics used, increasing the chances of several processors working on the same region of solution space.

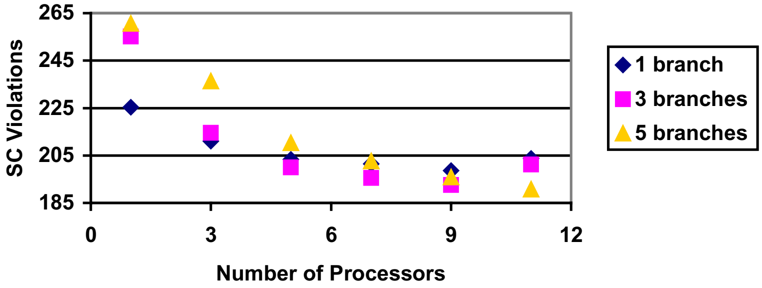


Fig. 6. Soft constraint violations with different number of processors

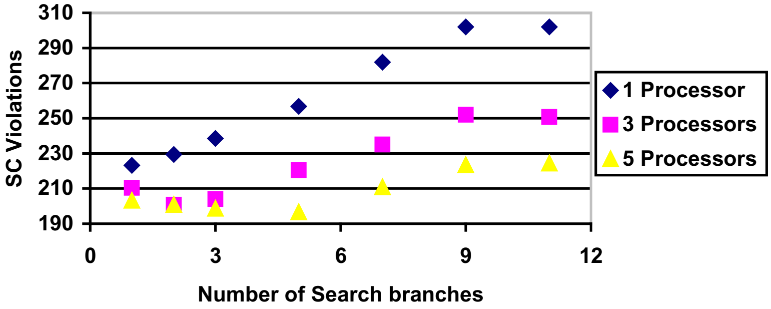


Fig. 7. Soft constraint violations with different number of solution candidates

While the benefit of adding extra Processors is obvious when measuring the hard constraint violations, Figure 6 shows more subtle but equally important results when the final fitness (number of soft constraint violations) is examined. It seems that the performance suffers if too many processing units are used with too few search branches, probably due to a degree of repetition by the small number of low-level heuristics. This graph shows three curves, the first uses a hierarchical approach that keeps one search branch in memory, the second keeps three search branches in memory and the last keeps five. It is clear that the optimal number of processors is related to the number of search branches that the hyper-heuristic works on concurrently, the optimal number of processors increasing with the number of search branches kept.

We investigated further the effect of variation in the number of solutions kept at one time. Figure 7 shows how the number of the search branches retained affects the algorithm performance. When only one processor is used it appears to be most effective to keep one search branch, as with the serial approach, but as more processors are used the optimal size of the solution population increases, suggesting a direct relationship between processors used and optimal population size. It appears that a small population causes an increased likelihood of becoming trapped in local optima. An excessively large population of solutions may lead to an overall increase in processing time in order to reach feasible solutions, as processing power is wasted trying to optimise less promising solutions.

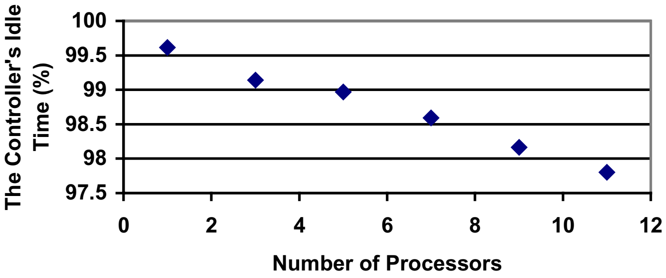


Fig. 8. Controller idle time

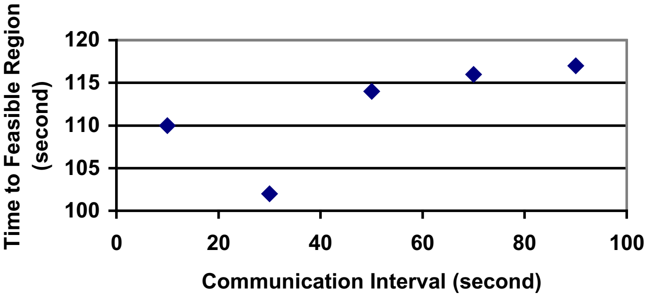


Fig. 9. Effect of communications interval on time taken to reach feasibility

The task of the controller is very different to that of the processors running low-level heuristics. By looking at the idle time of the controller we can estimate its capacity in terms of how many processors it can control. Figure 8 suggests that it would need to be controlling about 40 processors before reaching 100% usage, when the inevitable delays became an issue.

We examined our hybrid-agent architecture to determine optimal communication intervals between controllers. Figures 9 and 10 are the results of experiments using two controllers and three processors each. Figure 9 shows that in order to maintain the variety of solutions, and not having all the nodes converging prematurely, we should specify the length in between each communication to be within the range of 25–35 seconds. We can compare the time to reach a feasible solution for the hybrid architecture with results obtained for the hierarchical architecture in Figure 5. Using the same number of CPUs (one controller and seven processors), the former model increases robustness without a decrease in performance.

Figure 10 shows the optimal communication interval at 30 seconds for the more challenging task of lowering constraint violations and shows clearly that if the interval is too long or too short sub-optimal results are achieved.

The 20 problem instances are used vary in the number of events and resources. We used the hierarchical approach with three processors, allowing a maximum of two search branches for the parallel version. Table 2 compares the result from our sequential and parallel approaches. The first number represents the time in

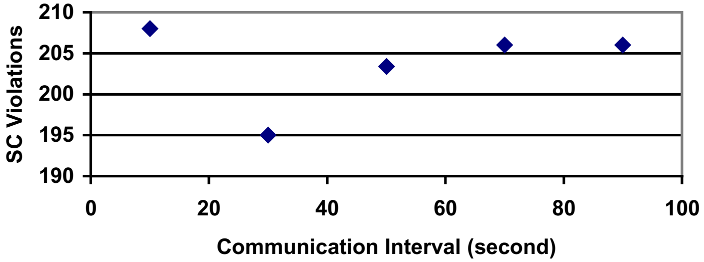


Fig. 10. Effect of communication interval on solution quality

Table 2. Performance comparison between the serial and parallel version

S	146/235	72/208	127/233	408/467	–	213/398	422/510	92/240	131/206	306/247
P	118/205	78/165	152/218	299/427	571/524	134/357	517/509	87/197	120/182	343/223
S	129/243	–	260/310	–	256/369	717/192	511/510	50/181	144/390	615/281
P	101/220	241/258	183/257	441/531	253/368	85/172	689/166	637/179	196/336	134/265

seconds to reach a feasible region and the second number represents the soft constraint violations after the time limit. This shows that the parallel hyper-heuristic is promising, solving all the problems within the time limit and that problems 5, 12, 14 cannot be solved within the available time using a sequential approach. While the parallel approach usually reaches a feasible solution faster than a serial approach, its real benefits are in finally reaching a better solution (fewer soft constraint violations), which it does in every case.

5 Conclusion

5.1 Discussion

In this paper we have shown how hyper-heuristics can be easily and effectively used for timetabling. A hyper-heuristic framework was designed and developed in order to obtain an understanding of the nature of the problem before taking further steps in building a framework for a general scheduler. We have shown that, though the choice-function-based hyper-heuristics are not problem-specific, they manage to obtain as good or better timetables in small-to-medium sized problems as those purposely designed for the problem. Automatic parameter tuning for the choice function will be added to increase the selection performance, which may improve performance, especially on larger problems. We feel that the low-level heuristic can be further generalised using non-specific performance statistics, such as the number of violations of individual constraints, whilst preserving the reusability of the whole framework.

Carrying out intensification and diversification simultaneously has been investigated on two parallel architectures: hierarchical and hybrid-agent. The in-

Table 3. Best result in the competition (min/max) [30]

45/257	25/128	61/266	112/441	77/412	3/246	5/281	4/214	16/184	54/308
38/273	79/290	71/364	25/345	14/235	11/300	69/409	24/153	40/414	0/185

vestigations have shown that it is useful for the controller to keep several branching points, and that the number should be adaptively set during the search. We have discovered close links between the number of processors used and the optimal configuration of the algorithm.

The issue of scalability is an important one and while logically and experimentally we have shown that our approach scales sensibly to a point, we also feel that if we were to aim to run the algorithm on massively parallel or peer to peer resources, a less heterogeneous approach will be needed. An approach in which controller and processor functions are combined into a single agent will be investigated. Under such a scheme, search branches could run on separate agents, without the group controller, and heuristic selection capability would be derived from individual agent experience rather than the average of all experiences.

5.2 Further Work

Several issues arose during this research and we are investigating several options to improve the performance and generality of our approach.

Firstly, a set of low-level heuristic precursors could be added to the scheduler. These would be used for building low-level heuristics automatically, by providing information on which parts of the schedule need to be tackled, and good ways to tackle them. The scheduler can use these precursors or the low-level heuristics that are provided by the problem provider. Two low-level heuristic classes have been discussed, *intensification* and *diversification*. A third group, *conditioning* heuristics, will be investigated. These aim not to improve the solution directly, but rather to create a state from which others can then generate an overall improvement.

Secondly, it seems fair to say that the best source of information in problem solving, other than that provided within the algorithm, is information collected while solving it. For our general scheduler, we intend to make available information such as the contribution to overall fitness of individual assignments in the solution and individual terms in the fitness function, and data on the times at which changes were made. This will make the search process more effective, by allowing heuristics to be selected based on their ability to improve specific features of a solution, and to target specific solution characteristics. Just as importantly, it ensures that heuristics are generally selected only when the current solution has characteristics that they can expect to resolve.

We have found a problem at the crossover between feasible and infeasible solutions when using a single choice function. Because infeasible components generally have high contributions to the fitness function, a heuristic that reaches a feasible solution receives a high performance score, causing repeated calls to the same heuristic for a long consecutive period afterward. When the heuristic is ineffective in the feasible region, this can be inefficient. We propose a multi-level hyper-heuristic in solving this, with a top-level controller selecting from a set of hyper-heuristics based on solution feasibility and strategy. These hyper-heuristics then choose suitable low-level heuristics for the specific situation. This makes the decision-making process in the search more explicit, and allows heuristics to be designed in the knowledge that they would be used at appropriate times. This would also make the selection of low-level heuristic against the constraint group more effective. The long-term influence of previously applied low-level heuristics on current ones will also be examined. This extends the idea of performance metrics for pairs of heuristics to sequences of arbitrary length, potentially improving the efficiency of the selection process, especially for conditioning heuristics.

Finally, we noticed that in the parallel implementation, preserving one branch for sub-optimal solutions generated by diversification heuristics proved essential in avoiding stagnation at local minima. We will further this study to adjust adaptively the number of diversifying branches: for example, to see if machine learning can be used to predict search trends to help the scheduler to balance the intensification and diversification phases. An autonomous agent approach will be investigated to see whether having an independent decision-making mechanism on every processor helps increase overall performance. Apart from getting closer to the global optimum, the scalability of the algorithm would also increase. We will also investigate how parallel processing can improve the search process, by avoiding premature convergence and strategically applying methods for escaping local optima.

References

1. Abramson, D.: Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms. Manage. Sci. **37** (1991) 98–113
2. Abramson, D., Abela, J.: A Parallel Genetic Algorithm for Solving the School Timetabling Problem. In: Proc. 15th Australian Computer Science Conference (ACSC-15), Vol. 14 (1992) 1–11
3. Ayob, M., Kendall, G.: A Monte Carlo Hyper-heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine. In: Proc. Int. Conf. on Intelligent Technologies (InTech'03, Chiang Mai, Thailand, Dec. 17–19, 2003) 132–141
4. Bullnheimer, B., Kotsis, G., Strauss, C.: Parallelization Strategies for the Ant System. In: High Performance Algorithms and Software in Nonlinear Optimization. Applied Optimization Series, Vol. 24. Kluwer, Dordrecht (1998) 87–100
5. Burke, E. K., Dror, M., Petrovic, S., Qu, R.: Hybrid Graph Heuristics within a Hyper-heuristic Approach to Exam Timetabling Problems. In: Golden, B. L., Raghavan, S. and Wasil, E. A. (eds.): The Next Wave in Computing, Optimization, and Decision Technologies. Conference Volume of the 9th INFORMS Computing Society Conference Springer, Berlin (2005) 79–91
6. Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-Heuristics: An Emerging Direction in Modern Search Technology. Chapter 16 in: Glover, F., Kochenberger, G. (eds.): Handbook of Meta-Heuristics. Kluwer, Dordrecht (2003) 457–474

7. Burke, E. K., Kendall, G., Soubeiga, E.: A Tabu Search Hyper-heuristic for Timetabling and Rostering. J. Heuristics **9** (2003) 451–470
8. Burke, E. K., Landa Silva, J. D., Soubeiga, E.: Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling. To appear in: Ibaraki T., Nonobe K., Yagiura M. (eds.): Meta-heuristics: Progress as Real Problem Solvers, Springer, Berlin (2005)
9. Burke, E. K., MacCarthy, B.L., Petrovic, S., Qu, R.: Knowledge Discovery in Hyper-heuristic Using Case-based Reasoning on Course Timetabling. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 276–287
10. Burke, E. K., Meisels, A., Petrovic, S., Qu, R.: A Graph-Based Hyper Heuristic for Timetabling Problems. Eur. J. Oper. Res. (2005) accepted for publication
11. Burke, E. K., Newall, J. P.: Solving Examination Timetabling Problems through Adaption of Heuristic Orderings. Ann. Oper. Res. **129** (2004) 107–134
12. Cantu-Paz E.: A Survey of Parallel Genetic Algorithms. Calculateurs Paralleles, Reseaux Syst. Repartis **10** (1998) 141–171,
13. Cowling, P., Kendall, G., Han, L.: An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In: Proc. Congress on Evolutionary Computation (CEC2002, Honolulu, Hawaii, May 12–17, 2002) 1185–1190
14. Cowling, P., Kendall, G., Soubeiga E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E. K., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 176–190
15. Cowling, P., Kendall, G., Soubeiga, E.: A Parameter-Free Hyperheuristic for Scheduling a Sales Summit. In: Proc. 4th Metaheuristics Int. Conf. (MIC 2001, Porto Portugal) 127–131
16. Cowling, P., Kendall, G., Soubeiga, E.: Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. In: Cagnoni, C. et al. (eds.): EvoWorkShops. Lecture Notes in Computer Science, Vol. 2279. Springer, Berlin (2002) 1–10
17. De Falco, I., Del Balio, R., Tarantino, E.: Solving the Mapping Problem by Parallel Tabu Search. Technical Report. Instituto per la Ricerca sui Sistemi Informatici Paralli, Italy (1996)
18. Han, L., Kendall, G.: Guided Operators for a Hyper-Heuristic Genetic Algorithm. In: Gedeon, T. D., Lance Chun Che Fung (eds.): Proc. AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence. Lecture Notes in Artificial Intelligence, Vol. 2903 (2003) 807–820
19. Han, L., Kendall, G.: Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm. In: Proc. Congress on Evolutionary Computation (CEC 2003, Canberra, Australia). **3** 2230–2237
20. Kendall, G., Mohd Hussin, N.: An Investigation of a Tabu Search Based Hyperheuristic for Examination Timetabling. In: Kendall, G., Burke, E., Petrovic, S., Gendreau, M. (eds.): Multi-disciplinary Scheduling: Theory and Applications I (MISTA'03, Selected Papers). Springer, Berlin (2005) 309–328
21. Kendall, G., Mohd Hussin, N.: Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at the MARA University of Technology. In: Burke, E. K., Trick, M. (eds.): Practice and Theory of Automated Timetabling V (PATAT'04, Selected Papers). Lecture Notes in Computer Science, Vol. 3616. Springer, Berlin (2005) this volume

22. Petrovic, S., Qu, R.: Case-Based Reasoning as a Heuristic Selector in a Hyper-Heuristic for Course Timetabling Problems. In: Proc. Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies, Vol. 82 (2002) 336–340
23. Randall, M., Abramson, D.: A General Parallel Tabu Search Algorithm for Combinatorial Optimisation Problems. In: Proc. 1999 Parallel and Real Time Conference (Melbourne, Australia) 68–79
24. Ross, P., Marín-Blázquez, J. G., Schulenburg, S., Hart, E.: Learning a Procedure That Can Solve Hard Bin-Packing Problems: A New GA-Based Approach to Hyper-heuristics,. In: Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2003, Chicago, IL) Lecture Notes in Computer Science, Vol. 2724. Springer, Berlin (2003) 1295–1306
25. Ross, P., Schulenburg, S., Marín-Blázquez, J. G., Hart, E.: Hyper-heuristics: Learning to Combine Simple Heuristics in Bin-Packing Problems. In: Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2002, New York) 942–948
26. Socha, K., Knowles, J., Sampels, M.: A Max–Min Ant System for the University Course Timetabling Problem (2002). In: Dorigo, M., Di Caro, G., Sampels, M. (eds.): Ant Algorithms 3rd Int. Workshop (ANTS 2002, Brussels, Belgium). Lecture Notes in Computer Science, Vol. 2463. Springer, Berlin (2002) 1–13 (Also Technical Report TR/IRIDIA/2002-18)
27. Soubeiga, E.: Development and Application of Hyperheuristics to Personnel Scheduling, Ph.D Thesis. University of Nottingham (2003).
28. Terashima-Marin, H., Ross, P. M., Valenzuela-Rendon, M.: Evolution of Constraint Satisfaction Strategies in Examination Timetabling. In: Banzhaf, W. et al. (eds.): Proc. Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann, San Mateo, CA (1999) 635–642
29. http://www.leeds.ac.uk/iss/wrgrid/
30. http://www.idsia.ch/Files/ttcomp2002/

Transport Timetabling

A Hybridised Integer Programming and Local Search Method for Robust Train Driver Schedules Planning

Ignacio Laplagne, Raymond S.K. Kwan, and Ann S.K. Kwan

School of Computing, University of Leeds,
Leeds LS2 9JT, UK
{ignacio, rsk, ann}@comp.leeds.ac.uk

Abstract. When a train arrives at a station, it often stops for some time before continuing, giving rise to a *window of relief opportunities* (WRO), during which the train may be handed over between drivers. Incorporating these windows into the scheduling model may help improve the robustness and efficiency of driver schedules. However, if it is formulated as a set covering problem, the incorporation of WROs would cause the resulting model to be too big to be solved in realistic times with current technology.

In this paper, we propose a combined integer programming and local search approach. In the first step, WROs are approximated, and the problem is solved using integer programming. Using the solution thus obtained as a starting point, WROs are restored and a multi-neighbourhood local search algorithm takes over. We also investigate the possibility of deriving a new set of approximations from the local search solution, and loop back to the integer programming phase.

The algorithm is tested using real-life data from a large rail network in Scotland, producing improved, operational schedules for this network.

1 The Driver Scheduling Problem

Driver scheduling (also known as *run cutting*) is the problem of determining the composition of a set of driver *shifts* (a *schedule*) for a day's transport operation requiring coverage by drivers. Shifts are in turn formed by *spells*, which are sections of continuous work on a vehicle. The main objective is to minimise the operational cost of the schedule (usually measured either in number of shifts, total payable hours, or a combination of both). Driver scheduling can be seen as the third process in transport operations planning, coming after route/timetable planning and vehicle scheduling, and before driver rostering. We will concentrate in this paper on train driver scheduling. Instance input information includes vehicle *diagrams* or *blocks*, which specify the work to be covered, some description of the network (nodes, walking and/or taxi passenger travel times, etc.), labour agreement rules (which govern the formation of shifts, and are usually quite complex), and other time allowances and schedule constraints, e.g. capping the number of drivers at a small depot [7].

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 71–85, 2005.

Although the main objective is usually centred on efficiency, in the last few years there has also been increasing interest in enhancing the capability of schedules to *recover* from irregularities in operations; more specifically, schedules should assist in avoiding, as much as possible, the propagation of delays. Measures of enhancing schedule robustness could be incorporated into the problem either through constraints or as penalties in the cost function. Most of the work on robust schedules has been done in the area of air transportation. Ehrgott and Ryan [2] introduce a bicriteria framework (considering cost and robustness as separate objective functions), and an algorithm to obtain Pareto optimal solutions from this model. A discussion of the trade-off between robustness and efficiency can be found in [1].

1.1 Generate and Select (GaS)

Generate and Select is possibly the most widely adopted approach to solve the driver scheduling problem. GaS consists of two main steps. In the *generation* phase, GaS builds a (very large) set $\mathcal{C}$ of candidate shifts. These shifts satisfy all the constraints specified in labour agreement rules. Usually, some constraints such as *minimum spell length* are added to limit the size of $\mathcal{C}$; note that these constraints are artificial, in the sense that they are not needed to properly describe the problem. In the *selection* phase, a minimal cost subset $\mathcal{S} \subset \mathcal{C}$ such that all vehicle work is covered is selected to form a schedule. The problem is thus effectively formulated as an (ILP) set partitioning problem, where shifts are sets of pieces of vehicle work and a legal schedule is a set of shifts that covers all vehicle work. In practice, however, a set covering formulation is used instead; this allows for a piece of work to be covered by more than one shift (i.e. *overcover*), which usually makes finding feasible solutions easier.

The ILP model is presented in Equation (1), where n is the number of candidate shifts, m is the number of work pieces, x_j indicates whether shift j is selected ($x_j = 1$) or not ($x_j = 0$), c_j is the cost of shift j, and a_{ij} is 1 if and only if shift j covers work piece i. W_1 and W_2 are used to balance the different objectives. Some domain-specific constraints will appear as extra constraints to the model; for example, the number of drivers at a specific depot k can be capped with a constraint $\sum_{j \in S_k} x_j \leq b_k$. Many constraints governing the validity of shifts (including the artificial constraints described earlier) are not explicit in the model, but rather implicit in the generation of the set of candidate shifts:

$$\begin{array}{lll} \min & W_1 \sum_{j=1}^{n} c_j x_j + W_2 \sum_{j=1}^{n} x_j & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq 1\,, & 1 \leq i \leq m\,, \\ & x_j \in \{0,1\}\,, & 1 \leq j \leq n\,. \end{array} \tag{1}$$

1.2 Windows of Relief Opportunities

Vehicle runs range from a few hours to eighteen or even more hours. This means that more than one driver may be needed to cover a single run of vehicle work.

Drivers can only be relieved at particular <`location, time`> pairs; these are called *relief opportunities* (ROs). In the context of train driver scheduling, relief opportunities happen almost exclusively at train stations or at depots. We will say that a relief opportunity is *active* in a schedule if a driver relief takes place at that RO in the schedule; otherwise we will say that the RO is *inactive*.

When a train arrives at a station, it often stops for some time before continuing. We call these *windows of relief opportunities* (WROs). The attention of a driver may be required throughout the train's stay at the station platform. Sometimes a train may be allowed to be unattended by immobilising it on arrival and mobilising it again before departure, in which case the driver getting off the train does not have to wait for the driver getting on to arrive. However, these activities take time and it might not be worth the trouble unless the stoppage at the station is significantly long. Although WROs exist, they are usually simplified in driver scheduling as a single <`location, time`> pair, using the train's arrival time at the station.

1.3 Modelling WROs

The three main motivations of incorporating WROs into the scheduling model are as follows.

- *It enriches the scheduling model.* Information such as window length or attendance constraints, for example, allows users to better describe the real-world situation.
- *More efficient schedules can be obtained.* Given an instance I of the scheduling problem, the solution space of a model for I which contemplates WROs is a superset of those of its "approximated" counterparts, and thus the optimal solutions in the former model are always better than or equal to those obtainable in the latter.
- *Robustness can be enhanced.* Although this could be thought of as a special form of cost reduction, it is especially important *per se*, because windows provide for a very specific way of creating buffers, thus helping avoid the cascading of delays. Figure 1 shows an example where a time window is used to create a 3-minute buffer for driver S_2. Suppose the WRO is such that the train must remain attended, and S_1 is to take work on another vehicle after being relieved from vehicle A by S_2. Because there is a three-minute period in which both S_1 and S_2 are covering the window, S_2 can begin its spell on vehicle A up to 3 minutes late without causing any disruption to the execution of S_1. Had this WRO been modelled as a single <`location, time`> pair, there would have been no opportunity to consider the creation of this buffer.

2 Expanding the Scheduling Model

We want to expand the scheduling model, incorporating WROs as comprehensibly as possible. WROs could be modelled by a discrete set of contiguous

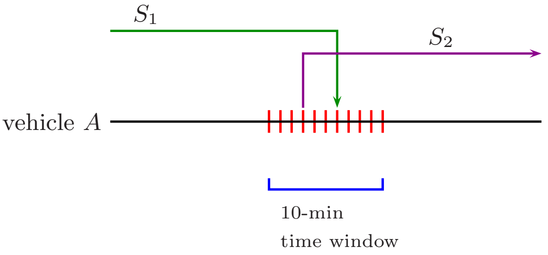


Fig. 1. An example on how WROs can be used to enhance robustness

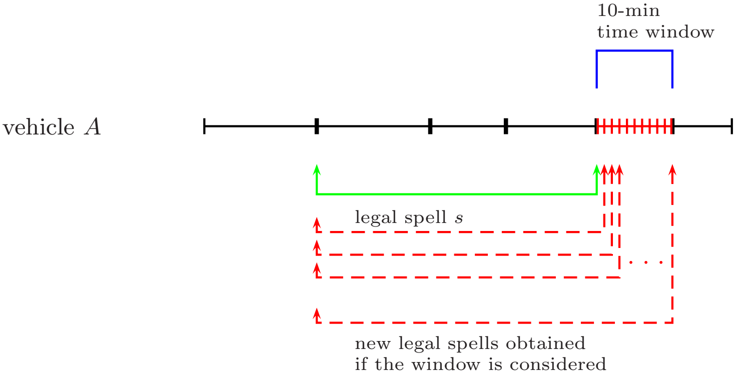
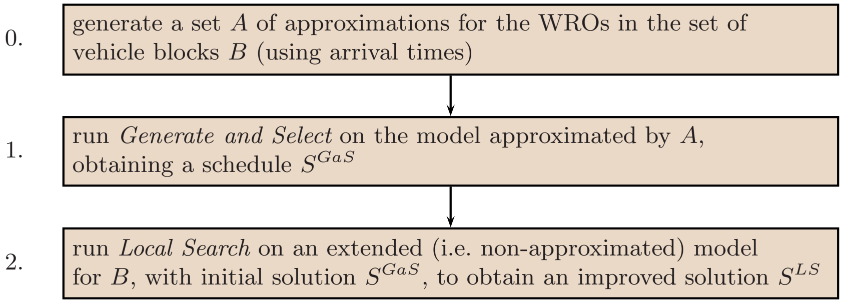


Fig. 2. Combinatorial explosion in the number of shifts when incorporating time windows into the scheduling model. Spells which are legal in the non-windowed model usually give rise to many new legal spells if a WRO on one of its ends is expanded.

<`location, time`> pairs, e.g. at each minute within the time window. In GaS, however, this would result in a combinatorial explosion in the number of legal shifts, which in turn would make the selection phase impracticably long. To illustrate this problem, consider Figure 2. Suppose s is a legal spell on vehicle A. The right end of the spell is marked by the presence of a 10-minute WRO, which in a non-windowed formulation is approximated by its arrival time. Now, if we expand the time window as explained before, 10 new ROs will be added. Because these new ROs are so close to the original one, it is very likely that changing the end-RO for spell s to any of the new ROs will result in a new legal spell. This means that there exist 10 new legal spells now. The combinatorial nature in which spells are mixed to form shifts adds up to create a huge new set of valid shifts.

2.1 A Two-Phase Approach to the Expanded Scheduling Problem

To overcome the difficulties presented while extending the scheduling model, we propose a two-phase scheme. In the first phase, GaS is run on an approximated



0. generate a set A of approximations for the WROs in the set of vehicle blocks B (using arrival times)

1. run *Generate and Select* on the model approximated by A, obtaining a schedule S^{GaS}

2. run *Local Search* on an extended (i.e. non-approximated) model for B, with initial solution S^{GaS}, to obtain an improved solution S^{LS}

Fig. 3. The proposed two-phase approach

(i.e. non-windowed) model of the problem, yielding a solution S^{GaS}. After that, windows are restored to the model, and a local search phase is run, using S^{GaS} as the initial solution. The approach is illustrated in Figure 3. It is worth noting that:

1. to obtain the initial set of approximations for the time windows, arrival times are used; this is consistent with preferred practice, as already mentioned;
2. because the solution set of the windowed model is a superset of *any* approximated model, S^{GaS} is naturally a feasible solution for the expanded (windowed) scheduling model, and thus can be safely used as an initial solution for the local search phase.

2.2 The Local Search Phase

The choice of a local search method for the second phase of our proposal is quite natural, because we want to exploit the fact that we already have an algorithm to obtain near-optimal solutions for the approximated model (`TRACS`, see Section 2.3). But, in addition, local search happens to be complementary to GaS in other areas. Essentially, local search is not restricted to the set of shifts generated by GaS (as the selection phase in GaS is). We exploit this fact in two ways. First, moves can work at a *piece-level* (i.e. the atomic unit on which spells and shifts are built upon). Second, it is usually the case that GaS is provided with certain parameters to restrict the size of the resulting pool of shifts; a typical example is restricting the length of spells. These parameters need not be enforced during the local search, allowing it to use shifts which would be deemed illegal during the generation phase of GaS, even under an approximated model. This adds another way of achieving improvements in the schedule, which is actually independent of the introduction of windows to the model.

The Moves. Apart from any moves that may be applicable to the non-windowed scenario, special consideration must be put in the way in which windows can be

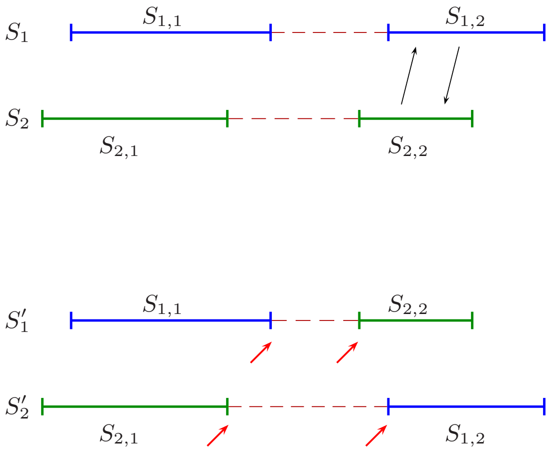


Fig. 4. Example of a move: spell swapping. S_1 is a shift with two spells $S_{1,1}$, $S_{1,2}$; S_2 is a shift with two spells $S_{2,1}$, $S_{2,2}$. Dotted lines represent non-driving periods. First, spells $S_{1,2}$ and $S_{2,2}$ are swapped (move-specific perturbation). Then, ROs that are relevant to the perturbation just performed are inspected for optimal relief times within WROs.

exploited. Our first idea was to design some window-related moves, along with some other moves for the non-windowed problem. However, we realised that windows could be exploited in practically every move that we could think of for the non-windowed version. Therefore, in our approach, a move is a composition of two steps. The first step is a specific perturbation to a subset of the shifts of the current schedule. The second one is a generic procedure that, given a set R of (active) relief opportunities, analyses the possibility of altering the times at which reliefs take place inside any windows that may belong to R. Each move selects this set R according to the characteristics of the perturbations it has performed on the first step. For example, if a move splits a shift into two shifts, the second phase will probably receive a set that consists of the relief opportunities happening at the ends of those two new shifts. An example of this two-step way of implementing a move can be seen in Figure 4. After the specific perturbation (in this case, a spell swap) is performed, the relief opportunities delimiting the gaps between the new spells are chosen as the set R to be passed to the window-evaluating routine (these are shown with arrows in the figure). The routine will test retiming the relief inside any window that may exist at those ROs, generating different sets of candidate shifts from S'_1, S'_2, which only differ in the relief times inside windows in R. Eventually, the best of these options will be chosen as the new solution.

The validity of schedules is governed by shift-level and schedule-level constraints. An example of the former is the maximum shift length; an example of the latter, the number of part-time shifts in relation to the total number of shifts in the schedule. Since a move is basically a perturbation of a subset of shifts of the current solution, shift-level constraints need only be checked for shifts that have been perturbed.[1] However, dealing with real-life train driver scheduling implies that the resulting shifts must adhere to a large and complex set of labour agreement rules. This makes the task of assessing the feasibility of a potential shift hard. In our implementation, the local search interfaces in a black-box fashion with existing checking routines that are used in the generation phase of the GaS solver, guaranteeing that shift-level constraints are treated consistently throughout the whole procedure. It is worth noting that the relief retiming trials in the second stage of a move may not only decrease the cost of a solution, but also turn an infeasible solution into a feasible one. Therefore, candidate solutions are not checked for validity during the first step of the move, but rather for each possible combination of relief timings inside the set R of ROs derived by the move.

Moves Implemented. We designed a two-level hierarchy for moves. L_0 moves are atomic moves, i.e., they do not resort to other moves to perform the perturbation.

- *Retiming of a relief inside a time window.* In the first phase, an (active) WRO is selected. In the second phase, all possible retimings of the relief inside that window are evaluated.
- *1-point crossover.* Two shifts are selected. One relief opportunity (windowed or not) is selected on each shift. The portions of the shifts to the right of those ROs are exchanged (if certain basic checks are satisfied). Because selected ROs are not necessarily placed at spell boundaries, this move can in effect break existing spells. Also, if the shifts share an RO, it is possible to select this RO as the crossover point. In this case, the so-far active RO becomes inactive after the exchange. In the second phase, ROs selected as crossover points are tested for relief retiming.
- *Transfer of spells and pieces of work.* Individual spells or pieces of work can be transferred from one shift to another. Begin and end ROs of the spell or piece transferred are tested for relief retiming in the second phase of the move.

Some of the L_0 moves we use were originally developed by Shen [8] and Shen and Kwan [9] for a simplified version of the problem.

L_0 moves are used or combined into L_1 moves. Some of these moves are high-level, requiring perhaps a sequence of many L_0 moves to be called; for example, "shift dismembering" is an L_1 move in which a procedure systematically tries to transfer every piece of work in a shift to other shifts in the schedule. Other L_1 moves are aimed at avoiding the cost of calling L_0 moves which we can predict will generate infeasible solutions, and instead only calling L_0 moves for which certain checks are satisfied. Therefore, although the feasibility of a shift is tested with external, black-box-like checking routines, some domain-specific knowledge *is* incorporated into the local search for efficiency reasons.

[1] This may include shifts which were not initially selected by the move, but which took part in a relief retiming in the second step of the move.

Finally, the overall driving scheme (what we call an L_2 move) is quite simplistic at the moment. Several loops of sequential calls to L_1 moves are tried, sometimes combined with conditioning phases (see next section). At the moment, we are using the disposition that experimentally showed the best results. We are investigating more intelligent and adaptive ways of doing this.

Conditioning. All the moves described above are called with the intention of decreasing the cost of the current solution; therefore, new solutions are accepted only if the cost is decreased (or, under certain circumstances, not increased). This may lead the local search to quickly stagnate in a local optimum. Metaheuristic approaches like *Tabu Search* [4] or *Simulated Annealing* [5] attempt to alleviate this problem by accepting certain cost-increasing moves; however, those approaches leave the cost function and (to a certain extent) the neighbourhoods unchanged.

We want to be more proactive in guiding the local search to good areas, and away from local optima; more specifically, we intend to favour certain structural properties in the solutions, which intuitively could lead the local search to better final solutions. As an example, short-spreadover shifts might be easier to "dismember" into other shifts; if the cost function is based on the number of shifts in the schedule, having a schedule with many short-spreadover shifts could make the task of reducing the number of shifts in the solution easier. The way we favour the appearance of these properties is by temporarily changing the cost function with which the local search is conducted. We call these the *conditioning phases*.

We investigated favouring two different properties:

1. Generate short-spreadover shifts. For that purpose, the cost of a move (which is a perturbation to a subset $S_m \subseteq S$ of shifts) is the decrease in the minimum spreadover of S_m ($\min_{s \in S_m} \mathit{spreadover}(s)$), relative to the original minimum spreadover in S_m before executing the move.
2. Generate long inter-spell gaps of non-work time, which may increase the chances of adding new work to the shifts in those gaps, and also of performing some spell-swapping moves that were previously infeasible. For that purpose, the cost of a move is defined to be the increase in the maximum inter-spell gap of S_m, relative to the original maximum inter-spell gap in S_m before executing the move.

The first criterion produced better results than the second. Overall, this way of alternating between conditioning and non-conditioning phases has the inconvenience that moves executed in one phase may be later undone by moves made on the following phase, because different, often conflicting, cost functions are used in each phase. In our experience, it is quite complex to tailor the specific moves and cost functions to stop this from happening. Instead, it might be better to adopt a more general solution, borrowing from the concepts of the *tabu list* and *aspiration criteria* in Tabu Search.

Table 1. Results of the GaS + LS approach for a real-life instance. Two variants of the hybridised method were analysed: in the first one, the artificial constraints introduced by the GaS model are enforced during the local search phase; in the second one, they are removed.

Artificial constraints	GaS	GaS + LS	Improvement
Not relaxed	459.48 h	458.13 h	−0.34%
Relaxed	N/A	455.18 h	−0.98%

2.3 Experiments

The algorithm was tested using real-life instances from the Scottish train operator *Scotrail*. A typical instance contains around 1,000 relief opportunities, which under the usual set of constraints (including the artificial ones) results in a pot of around 170,000 candidate shifts generated during the execution of GaS.

We used `TRACS` as the GaS solver. This is a commercial package developed over the last 30 years by the University of Leeds, which is currently being used by 30 transport companies in the UK [6], [10], [3]. Through extensive testing by experienced schedulers in the transport industry, `TRACS` solutions have consistently produced solutions at least as good, most often better, than the best known solutions. Hence, `TRACS`' solutions are regarded as near-optimal in this paper. The solutions obtained after the local search phase adhere to all operating rules as defined by the operator for its scheduling exercises.

Schedule Cost. We first analysed the potential of the new approach for reducing the cost of a schedule. Results for a sample run are shown in Table 1. For an average train operator, a 1% cost reduction in the drivers schedules would mean a saving of several hundred thousand pounds per year. The improvements obtained are thus considerable for train operators, especially given the near-optimality of the initial solution (on the approximated model).

Schedule Robustness. WROs could play a very important role in increasing schedules' robustness. Interaction with schedulers has allowed us to identify some aspects of a schedule that are key to enhancing its capability of absorbing delays that may occur during operation. We have defined two such indicators.

1. *Slack* is the unproductive time during signing on, mealbreak or joinup. Slack occurs naturally in a schedule; it might be added to a shift so that it conforms to some labour union rules. Moreover, slack in a shift could be used to absorb some delay when it occurs. The scheduler has to strike a balance between producing a cost-efficient schedule and adding enough slack so that the schedule meets robustness criteria.
2. When a train is so delayed that the slack in the driver's shift is unable to absorb it, a pragmatic way which a train crew manager would employ is to switch an available driver, say d, at the same location to take over the next train that the delayed driver was due to take on. This way the train driver

Table 2. Results of the GaS + LS approach for a real-life instance, using measures of robustness in the LS phase. Both total slack and number of swap opportunities could be increased, at no extra cost in the schedule.

Measure of robustness	S^{GaS}	After LS	Improvement
slack	1,604	1,648	2.74%
swapOpp	6	8	33.33%

can be relieved and have his/her legal break, and then take on the train that was originally assigned to d. We refer to this as a *swap opportunity*.

To test the capability of our approach in tackling robustness, we translated these two measures of robustness into separate cost functions: given a schedule S, *slack*(S) computes the total minutes of slack in S, and *swapOpp*(S) computes the number of swap opportunities in S. Let $c(S)$ be the original cost function related to the number of shifts and total payable hours in S. Then, the GaS phase is still executed using c as the cost function, generating a solution S^{GaS}. The local search phase, however, is now driven by one of the new, robustness-related cost functions. It is likely that train operators will want to balance cost and robustness in a schedule. For these initial tests we decided to force the local search to guarantee that every solution obtained during its execution is as cost-efficient as the starting solution, S^{GaS}; this was achieved by adding a constraint $c(S') \leq c(S^{GaS})$, where S' is any candidate solution being considered during the local search phase.

Results for experiments on the same data used in the previous tests are presented in Table 2. These suggest that both measures of robustness can be increased by using our approach, with no additional cost to the schedule.

3 Looping Back to GaS

3.1 Motivation

The final step in the approach proposed in Section 2.1 is a call to a local search phase, which produces a schedule S^{LS}. Now, even if the schedule was generated on the extended, windowed model, all driver reliefs in S^{LS} happen at precise time points. Therefore, S^{LS} is also a solution for a particular approximated model A^{LS}, namely that obtained by approximating all windows of relief opportunities by the times at which the reliefs are taking place in S^{LS} (inactive WROs are approximated by their arrival times). If *cost*(S^{LS}) < *cost*(S^{GaS}), it can be argued that A^{LS} is a *better approximated model* than the one derived from A, which was used in the GaS phase.

The GaS approach has been very good in solving approximated models; in particular, TRACS is being successfully used by transport operators for their scheduling tasks. Therefore, if restricted to an approximated model, it may happen that GaS/TRACS can perform better than the local search we have implemented. It

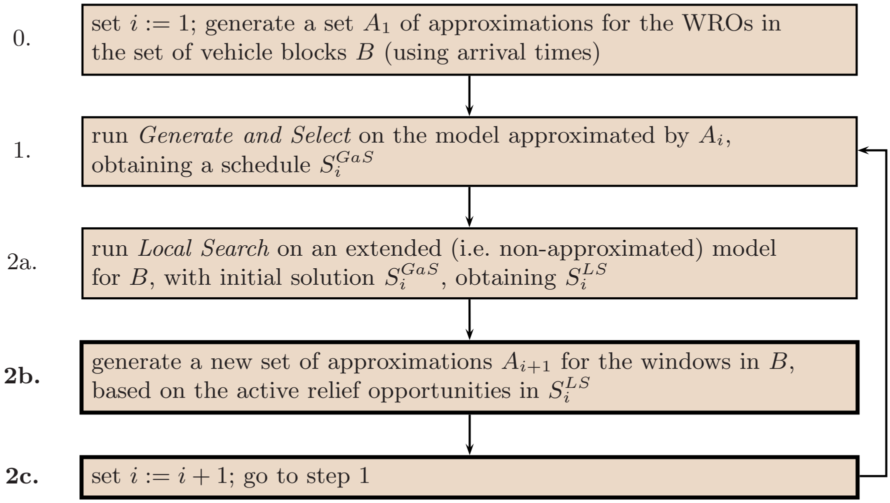

0.
set $i := 1$; generate a set A_1 of approximations for the WROs in the set of vehicle blocks B (using arrival times)
1.
run *Generate and Select* on the model approximated by A_i, obtaining a schedule S_i^{GaS}
2a.
run *Local Search* on an extended (i.e. non-approximated) model for B, with initial solution S_i^{GaS}, obtaining S_i^{LS}
2b.
generate a new set of approximations A_{i+1} for the windows in B, based on the active relief opportunities in S_i^{LS}
2c.
set $i := i + 1$; go to step 1

Fig. 5. The loop-back version of the algorithm: a new set of approximations A_{i+1} is generated from the solution S_i^{LS} obtained in the last call to local search, and fed back to the GaS solver

may thus be wise to attempt solving the problem for the approximated model A^{LS} using GaS. Once this first loop-back to GaS is established, it is natural to consider extending the idea to running GaS and local search in a loop, iterating until no further improvement is achievable. The algorithm is presented in the following section.

3.2 A Loop-Back Version of the Algorithm

The loop-back version of the algorithm is presented in Figure 5. Again, an initial set A_1 of approximations is generated using the arrival times. GaS and local search phases are then carried out in sequence. In step 5, a new set of approximations A_{i+1} is generated according to the times at which reliefs are taking place inside windows in S_i^{LS}, and the GaS+LS loop starts again with this new set of approximations.

3.3 Experiments

We present two sets of experiments on the loop-back mechanisms to GaS. These provide initial evidence that the approach is viable, pointing out at the same time the main issues arising when implementing such a scheme.

One: Unconstrained GaS Phases. On the first set of experiments, the only information derived from the local search phase for the next call to GaS is that of the next set of approximations A_{i+1}. We call these *unconstrained experiments*,

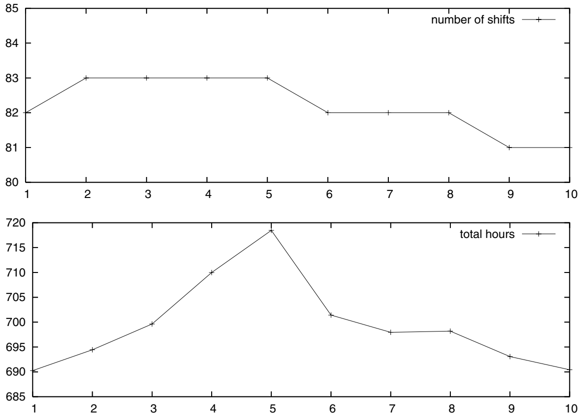


Fig. 6. Unconstrained experiments for the GaS+LS loop. The x-axis shows the number of iteration of the loop; the y-axes show two measures of cost, number of shifts and payable hours. The objective is to first minimise the number of shifts, and then the total payable hours.

because even if we know that solutions with specific cost values exist (we obtained them on the local search phase), we don't force GaS to equal or better those costs. The results of a typical run are shown in Figure 6.

As shown in the figure, the behaviour of the loop is erratic. Since the local search is set to accept only cost-decreasing solutions for these experiments, it is intuitive to expect that the overall behaviour of the cost function during the loop is decreasing. The reason why this is not reflected in the actual results is that the GaS solver is not an exact algorithm, and some heuristics are built into it to speed up the CPU times; for example, with the default settings the branch and bound phase will stop as soon as it reaches a "good enough" solution, which might not be the best that can be achieved if all branches are explored. It is still interesting to see that, even with these settings, the loop has been able to generate better solutions (iterations 9 and 10) than the one obtained on the first iteration. This supports the intuition that considering time windows would lead to more efficient solutions.

Two: Constrained Schedule Size on GaS Phases. For the second set of experiments, we added a hard constraint on the calls to GaS, specifying that the number of shifts in the final solution must not exceed the one obtained in the previous call to the local search. The results are shown in Figure 7.

The results show that GaS now enforces the max-schedule-size condition. However, since there was no constraint issued on the total payable hours, the behaviour on that component of the cost is still erratic. It is easy to think of different ways of further constraining the GaS phase to control its behaviour. As an example, we tried adding a constraint on the total payable hours; however, this seemed to render the solver unable to find a feasible solution. At the time

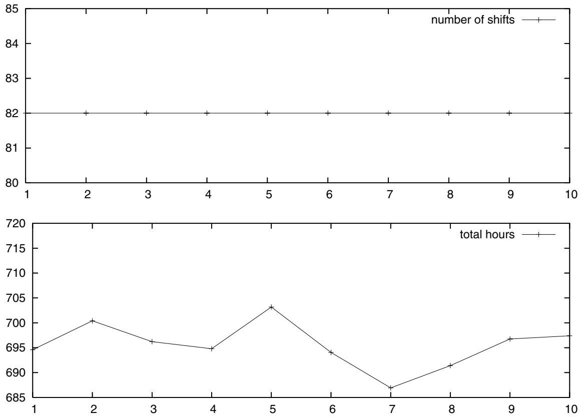


Fig. 7. Cost-constrained experiments for the GaS + LS loop: schedule size and total payable hours vs iteration number. While the constraint on schedule size succeeds in controlling the upper limit on size, the end result is worse than the one obtained when the constraint was not enforced. The algorithm is erratic when dealing with payable hours.

of writing this paper, we do not have a final explanation for this. However, we can think of several possible reasons for this, including:

1. The local search solution is present at some node of the branch-and-bound tree, but a limit on the number of nodes to expand prevents the algorithm from finding it; this limit is currently set to 5,000 nodes.
2. Because the GaS solver we are using includes a preprocessing phase, in which some relief opportunities deemed not useful are taken out of the problem, it may happen that the solution found in the local search (and every other solution with equal or better cost) is actually left out of the solution space considered by the branch-and-bound phase (no matter how many nodes are visited). Forcing `TRACS` to consider all ROs is not viable, because of the resulting increase in problem size.
3. Because the generation phase is artificially constrained to restrict the number of shifts generated, it might happen that some of the shifts in the local search solution are not available to the next phase of GaS. We can circumvent this problem by explicitly adding those shifts to the pot available for the selection phase; in fact, the choice of shifts to add is not restricted to those in the final schedule S_i^{LS}, but could also include shifts generated during the search process.

The reasons just described are quite independent from each other, and therefore it is possible that we have to tackle all of them before being able to get GaS to always find a solution that is better than or equal to the one obtained in the previous call to local search. However, tackling any of them would involve relaxing some heuristic rule which was originally added to make the execution time of the GaS phase feasible. This means that we must be intelligent in how to relax them, the way we solved the third problem being an example of that.

4 Conclusions

The experiments conducted in this paper suggest that a local search approach is capable of overcoming the limitations of the Generate and Select approach on an extended scheduling model, which fully incorporates windows of relief opportunities. We have developed different algorithms which show that cost improvements can be achieved; we also show that there is room for enhancing the robustness of the schedules. Better models need to be built to analyse the how cost and robustness can be tackled at the same time.

There are many open areas, which are subject to further research. The local search phase should be less conservative; on the other hand, constraints on shift structure mean that generating feasible perturbations of the current solution (schedule) may be a hard task. These two observations suggest that it could be useful to accept infeasible intermediate solutions. We are currently working on the application of *repair heuristics*, with two specific purposes: to measure how "far" an infeasible solution is from the feasible region, and to provide a way to cost feasible and infeasible solutions alike. The problem of cycling when including conditioning phases must also be properly studied. While the test instances used did not contain schedule-level constraints, more of them should be handled by the local search phase in order to make this a general-purpose tool.

The loop-back version of the algorithm is still quite basic at this point, and our intuition is that much better results might be obtained with the right algorithms. These should be centred on better exploiting the information gathered during a local search phase for the next call to the GaS phase, which may include new shifts generated during the search, information about relief opportunities that were instrumental in producing cost improvements, etc. Feasible solutions obtained during the LS phase could even help in pruning the branch-and-bound tree on the following call to GaS.

Acknowledgements

This research is supported by the Engineering and Physical Sciences Research Council (EPSRC) in the UK, with collaboration from AEA Technology Rail, grant no GR/520949/01. The authors would like to thank Jerry Farquharson and Mark Quinn from Scotrail for their valuable advice and the provision of test data.

References

1. Ageeva, Y.: Approaches to Incorporating Robustness into Airline Scheduling. Master's Thesis. Massachusetts Institute of Technology (2000)
2. Ehrgott, M., Ryan, D. M.: Constructing Robust Crew Schedules with Bicriteria Optimization. J. Multi-Criteria Decision Anal. **11** (2002) 139–150

3. Fores, S., Proll, L., Wren, A.: Experiences with a Flexible Driver Scheduler. In: Voss, S., Daduna, J. R. (eds.): Computer-Aided Scheduling of Public Transport. Lecture Notes in Economics and Mathematical Systems, Vol. 505. Springer, Berlin (2001) 137–152
4. Glover, F.: Tabu Search—Part I. ORSA J. Comput. **1** (1989) 190–206
5. Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P.: Optimization by Simulated Annealing. Science **220** (1983) 671–680
6. Kwan, A. S. K., Parker, M. E., Kwan, R. S. K., Fores, S., Proll, L., Wren, A.: Recent Advances in TRACS. In: Proc. 9th Int. Conf. on Computer-Aided Scheduling of Public Transport (2004)
7. Kwan, R. S. K.: Bus and Train Driver Scheduling. In: Leung, J. Y-T (ed.): Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapter 51. CRC Press, Boca Raton, FL (2004)
8. Shen, Y.: *Tabu Search for Bus and Train Driver Scheduling with Time Windows.* Ph.D. Thesis. School of Computing, University of Leeds (2001)
9. Shen, Y., Kwan, R. S. K.: Tabu Search for Driver Scheduling. In: Voss, S., Daduna, J. R. (eds.),: Computer-Aided Scheduling of Public Transport. Lecture Notes in Economics and Mathematical Systems, Vol. 505. Springer, Berlin (2001) 121–135
10. Wren, A., Fores, S., Kwan, A., Kwan, R., Parker, M., Proll, L.: A Flexible System for Scheduling Drivers. J. Scheduling **6** (2003) 437–455

Logistics Service Network Design for Time-Critical Delivery

Cynthia Barnhart[1] and Su Shen[2]

[1] Massachusetts Institute of Technology,
Cambridge MA 02139, USA
[2] Fedex Corporation,
Memphis, TN 38125, USA
cbarnhart@mit.edu

Abstract. Service network design is critical to the profitability of express shipment carriers. In this paper, we consider the problem of designing the integrated service network for premium and deferred express shipment delivery. Related existing models adapted to this problem are intractable for realistic problem instances, requiring excessive computer memory and solution times. We extend existing models and introduce a new approach to solve the resulting integrated service network design model. Our approach results in order or magnitude reductions in the numbers of variables to be considered in the integer program, allowing us to solve previously unsolvable problem instances. Applying our approach to the service network design problems of a large express package service provider, we demonstrate the potential for tens of millions of dollars in annual operating cost savings, and reductions in the numbers of aircraft needed to perform the service.

1 Introduction

In 1998, UPS generated revenues of $7.1 billion in domestic, air-express shipment service [12]. In 2002, the revenue of UPS' air-express shipment service grew by more than 15%, to about $8.2 billion [12]. Many Wall Street analysts attributed UPS' revenue growth and gain in market share to its emphasis on operating efficiency [10]. Efficient operations give a carrier a decisive competitive advantage, allowing the carrier to price its service more aggressively and gain market share, or use the cash flow generated to make further advantageous investments.

Given the high-revenue and low-operating margins of air express shipment service, even a single-digit percentage reduction in operating costs translates to a significant increase in profitability. Because service network design is at the core of express shipment delivery, it is a critical element in achieving operating efficiency. In this paper, we develop optimization models and algorithms to facilitate the design of cost-minimizing express shipment service.

1.1 Problem Description

Express shipment carriers operate transportation equipment, including both aircraft and ground vehicles, and fixed facilities, such as hubs, to serve customer

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 86–105, 2005.

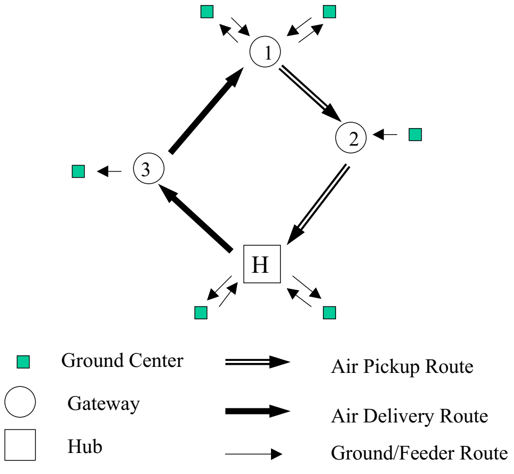


Fig. 1. Express package service operations

pick-up and delivery requests within tight time windows. Figure 1 depicts a partial express shipment delivery service network. Typically, packages are transported by ground vehicles to *ground centers,* or more specifically, *origin ground centers.* A ground center can serve as both an origin and a destination ground center, depending on whether the operation is a pickup or delivery. A ground center is usually associated with a city, although there might be several ground centers for a large city. After packages arrive at the origin ground center, an *origin sort* is conducted to determine the routing for each package based on its destination and a pre-specified package service plan. Although there are exceptions, the shipment is transported to a *gateway* (that is, an airport) either by a ground vehicle or a small aircraft. Packages at gateways are then loaded onto jet aircraft and transported over a *pickup route* to a *hub.* Upon arrival at a hub, packages are sorted, consolidated by destination, and loaded for delivery along *delivery routes* to final destination gateways. At the destination gateway, packages are offloaded onto ground vehicles or small aircraft and transported to their respective destination ground centers. At the destination ground center, a *destination sort* is conducted, and packages are loaded onto ground vehicles for final delivery.

Carriers usually offer different levels of service and charge higher premiums for higher levels of service. The level of service is characterized by the time from pick-up to delivery. For example, UPS offers both next-day and second-day services. For shipments picked up on a given day, next-day service has guaranteed delivery by the early morning of the next day, typically before 10AM, and second-day service has guaranteed delivery by the end of the second day. For both services, the full premium is refunded to customers if delivery is not made on time [13].

Operations for different services are similar, with the same equipment and facilities used, although at different times. More specifically, the same aircraft is used to deliver next-day shipments during the night and second-day shipments during the day. Although next-day and second-day operations are performed sequentially, the two services are linked. Fleet position as a result of the pickup and delivery operation of the next-day operation affects fleet position and costs associated with the second-day operation, and vice versa. In the carrier's current practice, because of problem size and complexity, tactical planning for next-day and second-day services is done sequentially, solving two independent problems, one for next-day and another for second-day service, with fleet position fixed in the second problem based on the results of the first. Planning next-day and second-day services simultaneously, that is, considering the *integrated next-day and second-day problem,* is the focus of the research we describe in this paper.

1.2 Contributions and Paper Outline

The contributions of our research include:

- Designing a solution methodology to solve the integrated next-day and second-day express shipment service problem. Because existing approaches are intractable for large-scale problems, we introduce a new approach that allows us to solve previously unsolvable problem instances. In addition to its relevance to express package delivery, our approach can also be applied to other problem types, including multi-commodity flow problems and crew-scheduling problems, to reduce model size and improve solution speed; and
- Demonstrating the efficacy of our approach on problem instances provided by a large U.S. carrier. Our results indicate that tens of millions of dollars in annual operating costs can be saved, with even greater potential savings in aircraft ownership costs and hub set-up and maintenance costs.

In Section 2 of this paper, we present our modeling approach for the integrated next-day and second-day express shipment service design problem. Then, in Section 3, we detail our solution approach involving decomposition and column generation. In Section 4, by applying our approach to data representing the integrated next-day and second-day express shipment operation of UPS, we demonstrate the scalability and practical significance of our work.

2 Modeling the Integrated Next-Day and Second-Day Express Air Service Problem

The integrated next-day and second-day express shipment service problem of a large U.S. carrier is to determine a *cost-minimizing* service network design for next-day and second-day operations simultaneously. Costs are incurred for aircraft operation (including *ferry flights*), ground vehicle operation and package handling. Ferry flights represent the repositioning of empty aircraft, usually to increase aircraft productivity. Aircraft operating cost includes two components:

1. *block time cost* including crew and fuel costs resulting from operating a flight leg;
2. *fixed cycle cost* incurred on each flight leg, typically including the landing fees and other one-time charges.

Ground vehicle operating costs, largely based on the distance traveled, are much smaller than aircraft operating costs, and hence, we consider them to be zero. Package handling cost also includes two components: a cost based on block time and a fixed handling cost. Block time cost is a proxy for the marginal fuel cost, and handling cost largely includes the package handling cost at ground centers and hubs. Package handling costs are insignificant compared to aircraft operating costs, and hence, we consider them also to be zero.

The shipments for each origin–destination pair must follow a pre-defined service plan specifying the origin and destination gateways, and the hub at which the packages will be sorted. Given this, we aggregate shipments by origin-gateway–destination-gateway pairs, referred to as *origin–destination* (O–D) commodities or origin–destination volumes hereafter. We consolidate O–D commodities originating from the same gateway and assigned to the same hub into a single gateway–hub demand, defined as the *pickup demand* for the gateway–hub pair. Similarly, we consolidate O–D commodities destined to the same gateway and assigned to the same hub into a single gateway–hub demand, defined as the *delivery demand* for the gateway–hub pair. We assume all demands are deterministic.

In addition to serving all demands within specified time windows, express shipment service network design is subject to a number of restrictions, including:

1. Conservation of aircraft at gateways and at hubs: the number of arriving aircraft of a specified type must equal the number departing, for each location;
2. Airport capacity: the number of aircraft arrivals at a hub cannot exceed the number of aircraft parking spots at the hub;
3. Aircraft count: the number of aircraft of each fleet type used must not exceed the available number;
4. Aircraft capacity: the packages assigned to each aircraft cannot exceed the aircraft capacity; and
5. Hub sort capacity: the packages routed through a hub must not exceed its sort capacity.

Various forms of the express shipment service network design problem have been studied. Grünert and Sebastian [5] identify planning tasks faced by postal and express shipment companies and define corresponding optimization models. Leung and Cheung [9] propose models for the ground distribution network design problem. Kuby and Gray [8] consider the limited capacity, single-hub problem and apply the formulation to a case study involving Federal Express' west-coast hub. Barnhart and Schneur [2] present a formulation for the uncapacitated single-hub problem and Kim et al. [6], Krishnan et al. [7] and Armacost et al. [1] consider a capacity-restricted, multi-hub problem with flexible hub assignment, and conclude that service network design models, containing both

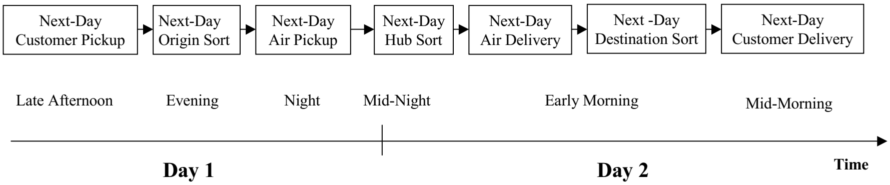


Fig. 2. Next-day air operations

integer aircraft route variables, referred to as *design variables*, and continuous package flow variables, have associated tractability issues. Their corresponding linear programming (LP) relaxations have solutions that are often fractional and difficult to transform into good-quality feasible solutions. Armacost et al. [1] report success in overcoming these tractability issues by applying extended formulation techniques that embed package flow decisions within the design variables. Given this, we address the integrated next-day and second-day express shipment service design problem by adapting the modeling approach of Armacost et al. and developing a new decomposition algorithm.

2.1 A Daily Model for the Integrated Problem

Figure 2 depicts the service timeline for the next-day operation, assuming packages are collected on Day 1. Carriers schedule pickup of packages from customers as late as possible to allow customers sufficient time to prepare their packages. Hence, packages arrive at origin ground centers in the late afternoon or early evening. After the origin sort in the evening, packages moving by air service are transported to origin gateways at night and loaded onto aircraft. From origin gateways, aircraft are transported along next-day air (NDA) pickup routes and arrive at hubs in the late night or early morning of the next day. The hub sort for NDA packages starts around midnight and lasts for 2–3 hours. After the hub sort, packages are delivered to their destination gateways, and then their destination ground centers, arriving in the early morning of the next day. At that point, the destination sort occurs at the ground center and packages are loaded onto ground vehicles and delivered to customers to meet delivery requirements. The same next-day air operation, starting with air pickup and ending with air delivery, is repeated each day except Sunday.

The second-day operation is similar to the next-day operation except for an expanded service time. Figure 2 depicts the service timeline for the second-day operation, assuming packages are collected on Day 1. At the origin ground center, the origin sort for second-day packages begins at night after the origin sort for next-day packages is completed. Then, second-day packages to be transported via air service stay at the origin ground center overnight, while others are transported to destination ground centers or hubs via ground service. On the morning of the next day, second-day packages at origin ground centers are transported to gateways and loaded onto aircraft that have just completed their

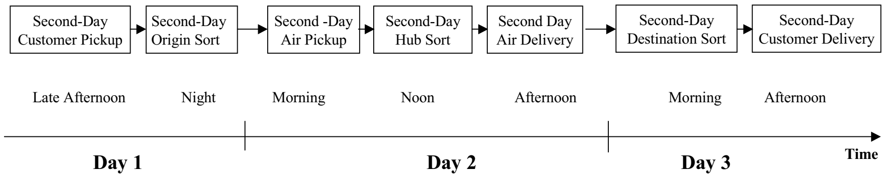


Fig. 3. Second-day air operation

NDA delivery routes. Aircraft then follow second-day air (SDA) pickup routes, arriving at hubs before noon. After the hub sort, packages are delivered either to destination ground centers via ground service or to destination gateways via air service. In the case of air delivery, aircraft carrying SDA packages arrive at destination gateways in the evening of Day 2. After SDA packages are unloaded, aircraft are available to begin their NDA pickup routes. The unloaded SDA packages are transported to destination ground centers, where they wait overnight for other second-day packages transported via ground. On the morning of Day 3, the destination sort for second-day packages begins after the completion of the destination sort for NDA packages collected on Day 2. SDA packages are then delivered to customers in the afternoon. Note that compared with the next-day service, the extended service time allows more extensive use of ground transport.

Although the complete second-day operation spans three days, as depicted in Figure 3, we can model the SDA operation as a *daily problem,* that is, the same operation is repeated daily, because a new second-day operation initiates each day. We illustrate this concept as follows. In Figure 4, we depict second-day operations over three days. The number in parentheses at the upper left corner of each box indicates the starting day of the corresponding SDA operation. We refer to a second-day operation starting on Day n as *second-day operation* n. On any given day n, there are three sets of second-day activities underway, one set for packages entering the system on Day $(n-2)$, one set for those entering on Day $(n-1)$, and finally, one for those on Day n. In the morning of Day n, the destination sort for second-day operation $(n-2)$ is conducted and packages of second-day operation $(n-1)$ are transported by air to hubs. Around noon, packages of second-day operation $(n-1)$ are sorted at hubs, and then, in the afternoon, packages of second-day operation $(n-2)$ are delivered to customers. Next, in the late afternoon, packages of second-day operation $(n-1)$ are delivered by air to destination gateways, and packages of second-day operation n are collected from customers. Finally, in the night, the origin sort for the second-day operation n is conducted. As is evident in Figure 4, the same air operation is repeated daily in second-day operations.

By recognizing that the second-day express shipment service operation can be captured by a single, representative day, we are able to model the integrated next-day and second-day operation as a daily problem. This allows us to minimize the number of variables and constraints in our models, and helps to reduce the challenges associated with solving these very large-scale formulations.

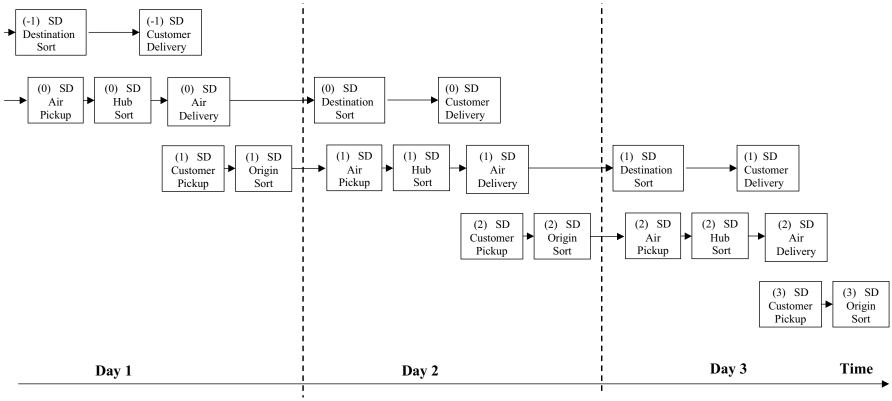


Fig. 4. Daily second-day air operation

2.2 Integrated Problem Formulation

Armacost et al. [1] present a new model for express shipment service network design using *composite variables* to reduce fractionality of the LP relaxation and enhance tractability. They define a *demand composite* to be a set of aircraft routes providing sufficient capacity to transport *all* the demand between the nodes contained in the selected aircraft routes. We illustrate the concept through a simple example in which we have three units of demand to be transported from gateway i to hub h. There is a single fleet type with capacity of two units. The operating cost of each aircraft on the route i to H is 10 units. One possible composite variable, denoted c, is two aircraft from i to h with cost 20, providing four units of capacity to transport all three units of demand. Note that one aircraft from i to h is not a valid composite variable because two units of capacity is insufficient to serve all the demand from i to h. In conventional network design models, to ensure that the three units of demand are served, we specify a constraint

$$2\,y \geq 3\,,$$

with variable y representing the number of aircraft selected. The optimal solution to the LP relaxation is then 1.5 aircraft, with 15 units of operating cost. In contrast, with composite variables, the condition that all demand must be served can be specified as

$$c \geq 1\,.$$

In the optimal solution to the LP relaxation using composite variables, c equals one, implying that two aircraft are selected to serve the demand, with a total operating cost of 20 units. This small example illustrates the improved LP bound achievable with composite variables.

We apply the demand composite modeling concept to the integrated problem and introduce the following notation. Let T indicate the type of service: next-day (denoted N) or second-day (denoted S); and let O indicate the operation:

pickup (denoted P) or delivery (denoted D). We define the following additional sets and variables.

Sets

F set of fleet types.
H set of hubs.
$\mathcal{N}$ set of gateways.
$\mathcal{C}^T$ set of demand composites for NDA ($T = N$) or SDA ($T = S$) network.
$\mathcal{C}^T_O$ $\left\{\begin{array}{l}\text{set of pickup } (O = P) \text{ or delivery } (O = D) \text{ demand composites for} \\ \text{NDA } (T = N) \text{ or SDA } (T = S) \text{ network.}\end{array}\right.$

Data

a^T_h $\left\{\begin{array}{l}\text{number of aircraft parking spots at hub } h \text{ for NDA } (T = N) \\ \text{or SDA } (T = S) \text{ network.}\end{array}\right.$
$b^{ih}_{T,O}$ $\left\{\begin{array}{l}\text{pickup } (O = P) \text{ or delivery } (O = D) \text{ demand between gateway } i \\ \text{and hub } h \text{ for NDA } (T = N) \text{ or SDA } (T = S) \text{ network.}\end{array}\right.$
γ^r_c number of aircraft routes r in demand composite c.
d_c cost of demand composite c, $d_c = \sum_{r \in c} \gamma^r_c d_r$.
d^f_{ij} ferrying cost for an aircraft of type f ferried from gateway i to j.
n^T_f $\left\{\begin{array}{l}\text{number of aircraft of typef available for NDA } (T = N) \\ \text{or SDA } (T = S) \text{ network.}\end{array}\right.$
γ^f_c number of aircraft of type f in demand composite c.
$\gamma^f_c(\bar{i})$ $\left\{\begin{array}{l}\text{number of aircraft of fleet type } f \text{ originating at gateway } i \text{ (or hub } h) \\ \text{in demand composite } c.\end{array}\right.$
$\gamma^f_c(\underline{i})$ $\left\{\begin{array}{l}\text{number of aircraft of fleet type } f \text{ destined to gateway } i \text{ (or hub } h) \\ \text{in demand composite } c.\end{array}\right.$
$\delta^{ih}_{T,O,c} = \left\{\begin{array}{l}1 \text{ if demand composite } c \text{ covers NDA } (T = N) \text{ or SDA } (T = S) \\ \text{pickup } (O = P) \text{ or delivery } (O = D) \text{ demand between gateway } i \\ \text{and hub } h, \text{ and} \\ 0 \text{ otherwise.}\end{array}\right.$

Decision Variables

v_c equals 1 if demand composite c is selected, and 0 otherwise.
$\varpi^{T,O}_{f,i}$ $\left\{\begin{array}{l}\text{number of aircraft of type } f \text{ on the ground at gateway (hub) } i \\ \text{during NDA } (T = N) \text{ or SDA } (T = S) \text{ pickup } (O = P) \text{ or} \\ \text{delivery } (O = D) \text{ operation. } \varpi^{T,P}_{f,i} = \varpi^{T,D}_{f,i}, \text{ if } i \notin H.\end{array}\right.$
$\phi^{T,f}_{ij}$ $\left\{\begin{array}{l}\text{number of aircraft of type } f \text{ ferried from gateway (hub) } i \text{ to } j \text{ after} \\ \text{the NDA } (T = N) \text{ or SDA } (T = S) \text{ operation.}\end{array}\right.$

We present the following formulation (INS) for the integrated NDA-SDA problem:

$$\min \sum_{T=\{N,S\}} \sum_{c \in \mathcal{C}^T} d_c v_c + \sum_{T=\{N,S\}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} d_{ij}^f \phi_{ij}^{T,f} \tag{1}$$

subject to

$$\begin{aligned} &\sum_{c \in \mathcal{C}_D^S} \gamma_c^f(\underline{i}) v_c - \sum_{c \in \mathcal{C}_P^N} \gamma_c^f(\bar{i}) v_c - \varpi_{f,i}^{N,P} + \varpi_{f,i}^{S,D} \\ &+ \sum_{j \in \mathcal{N}, j \neq i} \phi_{ji}^{S,f} - \sum_{j \in \mathcal{N}, j \neq i} \phi_{ij}^{S,f} = 0, \ i \in \mathcal{N}, \ f \in F, \end{aligned} \tag{2}$$

$$\begin{aligned} &\sum_{c \in \mathcal{C}_D^N} \gamma_c^f(\underline{i}) v_c - \sum_{c \in \mathcal{C}_P^S} \gamma_c^f(\bar{i}) v_c + \varpi_{f,i}^{N,D} - \varpi_{f,i}^{S,P} \\ &+ \sum_{j \in \mathcal{N}, j \neq i} \phi_{ji}^{N,f} - \sum_{j \in \mathcal{N}, j \neq i} \phi_{ij}^{N,f} = 0, \ i \in \mathcal{N}, \ f \in F, \end{aligned} \tag{3}$$

$$\sum_{c \in \mathcal{C}_P^T} \gamma_c^f(\underline{h}) v_c + \varpi_{f,h}^{T,P} - \sum_{c \in \mathcal{C}_D^T} \gamma_c^f(\overline{h}) v_c - \varpi_{f,h}^{T,D} = 0, \ h \in H, \ f \in F, \ T = \{N,S\}, \tag{4}$$

$$\sum_{c \in \mathcal{C}_P^T} \gamma_c^f v_c \leq n_f, \qquad f \in F, \ T = \{N,S\}, \tag{5}$$

$$\sum_{f \in F} \sum_{c \in \mathcal{C}_P^T} \gamma_c^f(\underline{h}) v_c \leq a_h, \quad h \in H, \ T = \{N,S\}, \tag{6}$$

$$\sum_{c \in \mathcal{C}_O^T} \delta_{T,O,c}^{ih} v_c \geq 1, \ (i,h) : b_{T,O}^{ih} > 0, \ T = \{N,S\}, \ O = \{P,D\}, \ i \in \mathcal{N}, \ h \in H, \tag{7}$$

$$\begin{gathered} v_c \in \{0,1\} \text{ for all } c \in \mathcal{C}^N \cup \mathcal{C}^S, \\ \varpi_{f,i}^{T,O} \in \mathbb{Z}_+ \text{ for } T = \{N,S\}, \ O = \{P,D\}, \ i \in \mathcal{N}, \\ \phi_{ij}^{N,f}, \phi_{ij}^{S,f} \in \mathbb{Z}_+ \text{ for } i,j \in \mathcal{N}, i \neq j, \ f \in F. \end{gathered}$$

The objective is to minimize the sum of the total NDA and SDA operating costs and the ferry costs between the operations. Constraints (2) and (3), the *boundary balance constraints*, ensure that aircraft at gateways are balanced between NDA and SDA operations. Constraints (2) require that the number of aircraft of type f at a gateway (hub) i at the start of the NDA pickup operation equals the number of aircraft of type f at gateway (hub) i at the end of the SDA delivery operation, adjusted by the number of aircraft of type f ferried into and out of gateway (hub) i at that time. Constraints (3) similarly require that the number of aircraft of type f at a gateway (hub) i at the end of the NDA delivery operation equals the number of aircraft of type f at gateway (hub) i at the beginning of the SDA pickup operation, adjusted by the number of aircraft of type f ferried into and out of gateway (hub) i at the end of the NDA operations.

Constraints (4) are *hub balance* constraints that ensure conservation of flow of aircraft by type at each hub, for both the NDA and SDA operation. The *count constraints* (5) limit the number of aircraft of each fleet type selected in the NDA and in the SDA operation to be no more than the number available. We need only to specify these constraints for pickup routes because conservation of flow constraints ensure that aircraft count will also be satisfied for delivery. The *landing constraints* (6) ensure that the number of aircraft arriving at a hub during NDA and during SDA operations does not exceed the parking spots available. We similarly only specify the landing constraints for pickup routes because aircraft conservation of flow ensures satisfaction for delivery. The *cover constraints* (7) ensure that at least one composite is selected to cover each nonzero gateway–hub demand. Because each demand composite is guaranteed to serve the associated gateway–hub demands fully, the cover constraints also ensure satisfaction of the aircraft capacity constraints. Finally, the last set of constraints ensure that the solution is comprised of a non-negative, integer number of composite variables, representing a set of aircraft routes, some of which will be flown by more than one aircraft.

3 Solving the Integrated NDA-SDA Formulation

Populating the INS formulation with all possible variables results in an intractable model: computer memory requirements and solution times are excessive. To address this issue, we use column generation to reduce the number of columns considered in solving the IP.

In column generation, we maintain a restricted version of the original model, called the restricted master problem (*RMP*), which includes only a limited set of columns. At each so-called master iteration, we solve the *RMP* to obtain a set of dual prices. Using this set of dual prices, we can either compute the reduced cost of each column explicitly, or solve a pricing sub-problem, as in Dantzig–Wolfe decomposition [4], to identify columns that potentially can improve the objective value of the *RMP*. If a problem has a diagonal block structure, pricing sub-problems can be specified for each block, resulting in simpler sub-problems. We repeat the process until no column is generated in one master iteration.

In this section, we explore different solution approaches for the INS formulation. We refer to the first approach as *naive column generation*; a standard column-generation approach in which demand composite variables with negative-reduced cost are generated as needed, with restrictions on the number of variables generated per iteration. In the second approach, referred to as *aggregate information-enhanced column generation*, smaller hub pickup or delivery sub-problems are solved to generate the necessary variables, and a master column represents the network design for the pickup or delivery operation of a hub. In the third approach, referred to as *disaggregate information-enhanced column generation*, we similarly solve hub pickup or delivery sub-problems, but each master column represents a demand composite variable, and we partition the hub sub-problem solution, that is, the solution to the pricing problem, into

Table 1. UPS next-day air network design problem statistics

Columns	195,009
Rows	3,302
Nonzeros	2,062,466

Table 2. Settings for CPLEX 6.5 MIP solver

Parameter	Setting
Backtrack	0.85
Branching direction	Up direction selected first
Node selection	Best estimate search
Variable selection	Based on strong branching
Relative best IP–best bound gap tolerance	0.0001

individual demand composites when adding columns to the *RMP*. In each of these solution approaches, we limit column generation to the root node LP relaxation, and consider only columns generated in solving the root node LP in branch-and-bound.

To evaluate these solution approaches, we first apply them to UPS' NDA problem only, not the integrated NDA-SDA problem, to gain insights into their respective effectiveness. The UPS NDA network includes 101 gateways, 7 hubs, 9 fleet types, 198 pickup and 195 delivery gateway–hub demands. Formulation statistics are reported in Table 1.

All computations were performed on an HP C3000 workstation with 400MHz CPU and 2GB RAM, running HPUX 10.20. The models and column generation processes were compiled using HP's aCC compiler with calls to the ILOG CPLEX 6.5 Callable Library [3]. CPLEX MIP Solver settings are reported in Table 2. For parameters not indicated, the CPLEX default values were used.

3.1 Naive Column Generation

In *naive column generation*, we evaluate the cost of demand composite variables explicitly using the dual prices obtained from solving the *RMP*. Denote the objective coefficient vector for demand composite variables as $\mathbf{d}$, and the constraint matrix for demand composite variables in constraints (2)–(7) as $\mathbf{B}_1, \mathbf{B}_2, \mathbf{H}, \mathbf{N}, \mathbf{A}$ and $\mathbf{C}$, respectively, and let the dual vector of the corresponding constraints be denoted $\pi^{\mathbf{B}_1}$, $\pi^{\mathbf{B}_2}$, $\pi^{\mathbf{H}}$, $\pi^{\mathbf{N}}$, $\pi^{\mathbf{A}}$ and $\pi^{\mathbf{C}}$. The reduced cost vector of demand composite variables is given by

$$\mathbf{d}' - (\pi^{\mathbf{B}_1})'\mathbf{B}_1 - (\pi^{\mathbf{B}_2})'\mathbf{B}_2 - (\pi^{\mathbf{H}})'\mathbf{H} - (\pi^{\mathbf{N}})'\mathbf{N} - (\pi^{\mathbf{A}})'\mathbf{A} - (\pi^{\mathbf{C}})'\mathbf{C} .$$

Demand composite variables with negative reduced cost are generated when solving the LP relaxation. In order to limit the size of the integer programming

Table 3. All-column and naive column generation results for the UPS NDA problem

	Solution approach	
	AC	NCG
Columns generated	–	16259
IP objective value	–	+0.01%
Run time (s):		
Root node LP	28	23
IP	8692	1550

model, we evaluate the effect of limiting the number of columns generated in one iteration to at most 100, 500, 1000, 2000, and 4000, respectively, and determine that generating at most 1000 columns in an iteration results in the fewest number of columns generated.

Our results for the naive column generation approach, limiting the number of columns generated in one iteration to at most 1000, are reported in Table 3. For comparison, we also solve the problem with *all* demand composite variables present, referred to as the *all-column approach.* "AC" represents the all-column approach, and "NCG" represents the naive column generation approach. In both approaches, the optimal LP value is the same. The objective value of the best IP solution using the naive column generation approach is 0.01% higher than that obtained with the all-column approach. This difference is explained by the fact that we generate columns only at the root node of the branch-and-bound tree, and hence, we do not consider certain demand composite variables whose reduced cost becomes negative as we branch in the branch-and-bound solution algorithm. This small degradation of the objective value is compensated for by the reduction in algorithmic complexity resulting from limiting column generation to the root node. In comparing running times, the naive column generation approach takes less than one fifth of the time required by the all-column approach.

3.2 Aggregate Information-Enhanced Column Generation

In our information-enhanced column generation approach, instead of generating *individual* demand composite variables with negative reduced cost, we generate a *set* of demand composite variables that is both feasible and has, summing over the demand composites in the set, a negative reduced cost.

We define a *set* of pickup (or delivery) demand composites to be a *hub pickup* (or *delivery*) *composite* if it

1. includes integer numbers of aircraft routes; and
2. satisfies the count constraints, and the landing and cover constraints specified for the pickup (or delivery) gateway–hub demands, at a set of hubs.

We introduce the following additional notation.

Sets and Data

$\mathcal{H}^T$ set of hub composites for the NDA ($T = N$) or SDA ($T = S$) network.

$\mathcal{H}_O^T$ $\begin{cases}\text{set of pickup } (O = P) \text{ or delivery } (O = D) \text{ hub composites for the} \\ \text{NDA } (T = N) \text{ or SDA } (T = S) \text{ network.}\end{cases}$

d_Θ cost of hub composite Θ, $d_\Theta = \sum_{c\in\Theta} d_c$.

γ_Θ^f number of aircraft of type f in hub composite Θ.

$\gamma_\Theta^f(\bar{i})$ $\begin{cases}\text{number of aircraft of type } f \text{ originating at gateway (hub) } i \text{ in hub} \\ \text{composite } \Theta.\end{cases}$

$\gamma_\Theta^f(\underline{i})$ $\begin{cases}\text{number of aircraft of type } f \text{ destined to gateway (hub) } i \text{ in hub} \\ \text{composite } \Theta.\end{cases}$

$\delta_{T,O,\Theta}^{ih} = \begin{cases}1 \text{ if hub composite } \Theta \text{ covers NDA } (T = N) \text{ or SDA } (T = S) \\ \text{pickup } (O = P) \text{ or delivery } (O = D) \text{ demand between gateway } i \\ \text{and hub } h, \text{ and} \\ 0 \text{ otherwise.}\end{cases}$

Decision Variables

v_Θ equals 1 if hub composite Θ is selected, and 0 otherwise.

We rewrite the INS formulation with hub composite variables (INS-H) as follows:

$$\min \sum_{T=\{N,S\}} \sum_{\Theta\in\mathcal{H}^T} d_\Theta v_\Theta + \sum_{T=\{N,S\}} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} d_{ij}^f \phi_{ij}^{T,f} \tag{8}$$

subject to

$$\begin{aligned}&\sum_{\Theta\in\mathcal{H}_D^S} \gamma_\Theta^f(\underline{i}) v_\Theta - \sum_{\Theta\in\mathcal{H}_P^N} \gamma_\Theta^f(\bar{i}) v_\Theta - \varpi_{f,i}^{N,P} + \varpi_{f,i}^{S,D} \\ &+ \sum_{j\in\mathcal{N},j\neq i} \phi_{ji}^{S,f} - \sum_{j\in\mathcal{N},j\neq i} \phi_{ij}^{S,f} = 0, \ i\in\mathcal{N}, \ f\in F\end{aligned} \tag{9}$$

$$\begin{aligned}&\sum_{\Theta\in\mathcal{H}_D^N} \gamma_\Theta^f(\underline{i}) v_\Theta - \sum_{\Theta\in\mathcal{H}_P^S} \gamma_\Theta^f(\bar{i}) v_\Theta + \varpi_{f,i}^{N,D} - \varpi_{f,i}^{S,P} \\ &+ \sum_{j\in\mathcal{N},j\neq i} \phi_{ji}^{N,f} - \sum_{j\in\mathcal{N},j\neq i} \phi_{ij}^{N,f} = 0, \ i\in\mathcal{N}, \ f\in F\end{aligned} \tag{10}$$

$$\sum_{\Theta\in\mathcal{H}_P^T} \gamma_\Theta^f(\underline{h}) v_\Theta + \varpi_{f,h}^{T,P} - \sum_{\Theta\in\mathcal{H}_D^T} \gamma_\Theta^f(\overline{h}) v_\Theta - \varpi_{f,h}^{T,D} = 0, \ h\in H, \ f\in F, \ T=\{N,S\} \tag{11}$$

$$\sum_{\Theta\in\mathcal{H}_P^T} \gamma_\Theta^f v_\Theta \leq n_f, \qquad f\in F, \ T=\{N,S\} \tag{12}$$

$$\sum_{f\in F} \sum_{c\in\mathcal{H}_P^T} \gamma_\Theta^f(\underline{h}) v_c \leq a_h, \quad h\in H, \ T=\{N,S\} \tag{13}$$

$$\sum_{\Theta \in \mathcal{H}_O^T} \delta^{ih}_{T,O,\Theta} v_\Theta \geq 1, \;\; (i,h) : b^{ih}_{T,O} > 0, \; T = \{N, S\}, \; O = \{P, D\}, \; i \in \mathcal{N}, \; h \in H \tag{14}$$

$$v_\Theta \in \{0,1\} \text{ for all } \Theta \in \mathcal{H}^N \cup \mathcal{H}^S$$

$$\varpi^{T,O}_{f,i} \in \mathbb{Z}_+ \text{ for } T = \{N, S\}, \; O = \{P, D\}, \; i \in \mathcal{N}$$

$$\phi^{N,f}_{ij}, \phi^{S,f}_{ij} \in \mathbb{Z}_+ \text{ for } i, j \in \mathcal{N}, \; i \neq j, \; f \in F$$

The formulation is the same as the INS formulation except that demand composite variables are replaced with hub composite variables. It is straightforward to show [11] that the INS-H formulation is at least as strong as the INS formulation. In the following example we describe a case in which the INS-H formulation is strictly stronger than the INS formulation.

Consider the example in Figure 5. There is a single fleet type with two units of capacity. We want to cover all gateway–hub demands in the example. We only consider the pickup operation for simplicity, but we can easily expand the examples to include delivery operations and aircraft balance without affecting formulation strength.

We consider only the demand composite variables and hub composite variables in the figure. (Other demand and hub composite variables do not affect the optimal integer or LP relaxation solution to the INS and INS-H formulation.) Excluding the balance, count, and landing constraints, the INS formulation is

$$\begin{gathered} \text{Min } 6dc_1 + 5dc_2 + 12dc_3 + 12dc_4 + 8dc_5 \\ dc_3 + dc_4 = 1 \\ dc_1 + dc_3 + dc_5 = 1 \\ dc_2 + dc_4 + dc_5 = 1 \\ dc_i \in \{0,1\}, \quad i = 1, 2, \ldots, 5\,. \end{gathered}$$

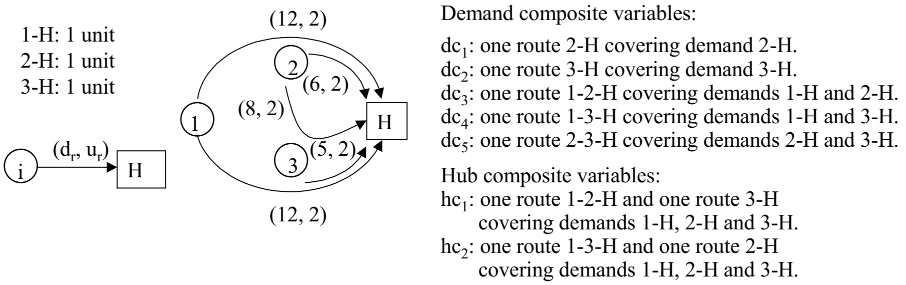
The resulting optimal solution to the LP relaxation is $\{dc_1 = 0,\ dc_2 = 0,\ dc_3 = 0.5,\ dc_4 = 0.5,\ dc_5 = 0.5\}$, with objective value 16.

Excluding the balance, count, and landing constraints, the INS-H formulation is

$$\begin{gathered} \text{Min } \; 17hc_1 + 18hc_2 \\ hc_1 + hc_2 = 1\,. \end{gathered}$$

The optimal solution to the LP relaxation is $\{hc_1 = 1,\ hc_2 = 0\}$, with objective value 17.

We denote the dual vector for constraints (9)–(14) as $\pi^{\mathbf{B_1}}$, $\pi^{\mathbf{B_2}}$, $\pi^{\mathbf{H}}$, $\pi^{\mathbf{N}}$, $\pi^{\mathbf{A}}$ and $\pi^{\mathbf{C}}$, respectively. Let $\mathcal{C}^{T,O,h}$ be the subset of NDA ($T = N$) or SDA ($T = S$) demand composite variables covering subsets of gateway–hub demands to hub h in the case of pickup ($O = P$) or subsets of gateway–hub demands from hub h in the case of delivery ($O = D$), and $\mathbf{v}^{T,O,h}$ the vector indicating the selection of



Demand composite variables:

dc$_1$: one route 2-H covering demand 2-H.
dc$_2$: one route 3-H covering demand 3-H.
dc$_3$: one route 1-2-H covering demands 1-H and 2-H.
dc$_4$: one route 1-3-H covering demands 1-H and 3-H.
dc$_5$: one route 2-3-H covering demands 2-H and 3-H.

Hub composite variables:
hc$_1$: one route 1-2-H and one route 3-H covering demands 1-H, 2-H and 3-H.
hc$_2$: one route 1-3-H and one route 2-H covering demands 1-H, 2-H and 3-H.

Fig. 5. Example of hub composite variable

those demand composite variables. Following the matrix notation introduced in describing naive column generation, we denote the constraint matrix for demand composite variables in $\mathcal{C}^{T,O,h}$ in constraints (2)–(7) as $\mathbf{B}_1^{T,O,h}$, $\mathbf{B}_2^{T,O,h}$, $\mathbf{H}^{T,O,h}$, $\mathbf{N}^{T,O,h}$, $\mathbf{A}^{T,O,h}$, $\mathbf{C}^{T,O,h}$, respectively. Denote the right-hand-side vector of constraints (5) and (6) for $T = \{N, S\}$ as $\mathbf{n}_T$ and $\mathbf{a}_T$. Denote the right-hand-side vector of constraints (7) for gateway–hub demands for $T = \{N, S\}$, $O = \{P, D\}$ and $h \in H$ as $\mathbf{I}^{T,O,h}$. We define the following sub-problem for $T = \{N, S\}$, $O = \{P, D\}$, and $h \in H$:

$$\begin{aligned}&\min\ [\mathbf{d}' - (\pi^{\mathbf{B}_1})'\mathbf{B}_1^{T,O,h} - (\pi^{\mathbf{B}_2})'\mathbf{B}_2^{T,O,h} - (\pi^{\mathbf{H}})'\mathbf{H}^{T,O,h}\\ &-(\pi^{\mathbf{N}})'\mathbf{N}^{T,O,h} - (\pi^{\mathbf{A}})'\mathbf{A}^{T,O,h} - (\pi^{\mathbf{C}})'\mathbf{C}^{T,O,h}]\mathbf{v}^{T,O,h} \qquad (15)\end{aligned}$$

subject to

$$\mathbf{A}^{T,O,h}\ \mathbf{v}^{T,O,h} \le \mathbf{a}_T \qquad (16)$$

$$\mathbf{N}^{T,O,h}\ \mathbf{v}^{T,O,h} \le \mathbf{n}_T \qquad (17)$$

$$\mathbf{C}^{T,O,h}\ \mathbf{v}^{T,O,h} \ge \mathbf{I}^{T,O,h} \qquad (18)$$

$$v_c \in \{0, 1\},\ c \in \mathcal{C}^{T,O,h}\,. \qquad (19)$$

Constraints (16) ensure that the selected demand composite variables at hub h satisfy the landing constraints at *all* hubs. We consider all hubs because the demand composite variables in $\mathcal{C}^{T,O,h}$ might include routes entering or departing hubs other than h. Constraints (17) are the count constraints, specified for each fleet type, and constraints (18) are the cover constraints specified for each gateway–hub demand to or from hub h.

The solution to a sub-problem is a hub composite, and the objective value is its reduced cost. If the objective value of a solution is negative, we add the corresponding hub composite variable to the *RMP*. The process terminates if after solving all sub-problems, one for the pickup operation and one for the delivery operation at each hub, and for NDA and SDA, not one sub-problem solution has a negative objective value. Because we ensure the set of columns generated

Table 4. Aggregate information-enhanced column generation results for the UPS NDA problem

Columns generated	7101
Master iterations	270
Objective value:	
root node LP	+0.001%
IP	n/a
Run time (s):	
root node LP	2842
IP	n/a

are feasible and the sum of their reduced cost is negative, we call this approach *information-enhanced column generation*. We refer to the information-enhanced column generation approach in which a sub-problem solution is introduced into the *RMP* in its *aggregate* form, that is, as a hub composite variable, *aggregate information-enhanced column generation*.

We apply aggregate information-enhanced column generation to the same UPS NDA problem instance that we solved with the naive column generation and the all-column approaches. Our results are reported in Table 4. Compared with the INS formulation, the optimal LP objective value increases by 0.001%. The MIP solver, however, runs out of memory and fails to find a feasible integer solution after 20 hours with the set of columns generated. The best bound achieved at that point is 2.4% higher than the true IP optimal objective value.

3.3 Disaggregate Information-Enhanced Column Generation

The columns generated by the aggregate information-enhanced column generation at the root node of the branch-and-bound tree fail to provide a feasible solution. The issue is that too many decisions are embedded in a column of the *RMP*. To overcome this issue, we introduce *disaggregate information-enhanced column generation*.

We replace the INS-H formulation with the INS formulation as the *RMP*. At each master iteration, we similarly solve the pricing problem (15)–(19) for the pickup and delivery operation of each hub and for NDA and SDA. If the objective value of a sub-problem is negative, instead of adding to the *RMP* a single column representing all the demand composite variables in the sub-problem solution, we partition the solution into individual demand composite variables and add to the *RMP* those that are not currently included. (Some demand composite variables might have been included in the *RMP* in earlier iterations.)

We apply disaggregate information-enhanced column generation to the same UPS NDA problem instance and report our results in Table 5.

Using disaggregate information-enhanced column generation, we generate less than 1% of all possible columns, and less than 10% of the number of columns generated using naive column generation. This indicates that the hub-based

Table 5. Disaggregate information-enhanced column generation results for the UPS NDA problem

Columns generated	1535
Master iterations	34
IP objective value	+0.11%
Run time (s):	
root node LP	307
IP	185

sub-problems are more effective than naive column generation in identifying columns that can be used in an optimal solution. The root node LP converges to the true objective value, but the IP objective value is somewhat worse than that obtained with naive column generation, because columns are again generated only at the root node of the branch-and-bound tree. Compared with the naive column generation and the all-column approaches, the root node LP relaxation takes longer to solve, but the IP solution time is significantly reduced using disaggregate information-enhanced column generation. Overall, disaggregate information-enhanced column generation achieves a 70% reduction in total solution time compared with naive column generation, and a 95% overall reduction compared with the all-column approach.

Compared with aggregate information-enhanced column generation, disaggregate information-enhanced column generation not only produces fewer columns, but also converges in fewer master iterations. Most importantly, solutions with objective values close to the optimal value can be identified with the set of columns generated.

4 Case Study

We apply the disaggregate information-enhanced column generation approach to the integrated UPS NDA-SDA problem, with the objective to minimize daily operating costs. Problem statistics are reported in Table 6.

To compare the integrated solution to sequential solutions, we use disaggregate information-enhanced column generation to solve in sequence the NDA and SDA problems. To solve the SDA problem, which is relatively small compared to the NDA problem, we generate only 3113 columns, or 1.4% of all variables, using the disaggregate information-enhanced column generation approach.

In Table 7, we compare the results of the sequential and integrated approaches with the solution generated by planners at UPS. Costs are reported as the percentage difference from those of the UPS solution. In the UPS solution, the SDA network is designed manually, while the design of the NDA network is accomplished using the composite variable approach of Armacost et al. [1].

In the first scenario, the *unconstrained NDA and SDA problem*, boundary balance conditions are not enforced between the NDA and SDA operations, and the two problems are solved independently, without aircraft balance constraints.

Table 6. UPS Integrated NDA-SDA problem statistics

Composite variables	NDA	SDA
	168372	59969
Ferry and ground variables	76215	
Rows	4623	
Master iterations	33	
Generated demand composite variables	3113	

Their combined solution value provides an upper bound on the potential savings achievable through integration of the NDA and SDA problems. In the second scenario, the NDA problem is first solved without aircraft balance constraints. Then the SDA problem is solved with balance constraints ensuring that the NDA operations can be executed as planned. The resulting total cost is slightly better than that of the UPS solution. Notably in this case, ferry costs increase significantly because many ferry flights are required to re-position aircraft before or after the NDA operation to perform the SDA operations. These ferry costs more than offset the savings achieved in the NDA solution. In the third scenario, a reverse sequence is followed, the SDA problem, without aircraft balance conditions, is first solved, and the NDA problem, with balance constraints ensuring the execution of the SDA operations, is then solved. The resulting operating costs of the NDA solution are greater than those of the UPS solution, but the daily total cost is much lower. This sequential approach produces less expensive solutions than the previous one for the following reasons:

- Because the SDA operation uses only about one third of the fleet used in the NDA operation, there is sufficient flexibility to position the unused aircraft in the SDA operation to match the needs of the NDA operation; and
- Most aircraft re-positioning for the SDA operation can be accomplished with revenue flight movements in the NDA operation, given the large number of NDA gateway–hub demands to be served.

In the last scenario, we solve the integrated NDA-SDA problem with disaggregate information-enhanced column generation. Although ferrying costs are more than double those in the UPS solution, the NDA and SDA operating costs are both reduced, reflecting the better coordinated aircraft movements. The daily operating cost savings of the integrated approach translates into tens of millions of dollars annually. Compared with the best sequential approach, the savings from the integrated approach come from: (1) reduced ferry costs; and (2) better coordinated NDA and SDA fleet movements. Beyond the tens of millions of dollars in operating cost savings, two fewer aircraft are needed in the integrated solution than in the sequential solution. This is significant because annual ownership costs for aircraft measure in the millions of dollars.

In all scenarios, savings attributable to the NDA operation are small or nonexistent, whereas savings attributable to the SDA operation are large,

Table 7. Sequential and integrated approach results for the UPS NDA-SDA problem

	Scenario	Daily revenue flight cost	Daily ferry flight cost	Total daily cost	Fleet usage
1	Unconstrained SDA	−23.4%	−100%	−15.9%	
	Unconstrained NDA	−7.3%			
2	Unconstrained NDA	−7.3%	+903.6%	−0.3%	−4
	Constrained SDA	−17.5%			
3	Unconstrained SDA	−23.4%	+218.5%	−5.9%	−3
	Constrained NDA	+1.9%			
4	Integrated SDA	−19.5%	+140.7	−8.1%	−5
	Integrated NDA	−1.2%			

reflecting the carrier's use of the Armacost et al. [1] optimization approach to design their NDA network, but not the SDA network.

We acknowledge that some operating requirements are not considered explicitly in our models. The staging of package arrivals at hubs is one example. Hence, the savings reported here might not be fully realized.

5 Summary

In this paper, we adapt the Armacost et al. [1] model to solve the integrated next-day and second-day express shipment service design problem, and present a new solution approach designed for large-scale problems. Our disaggregate information-enhanced column generation approach is shown to generate many fewer columns and help reduce IP solution time significantly. By solving the *integrated* NDA-SDA problem, we demonstrate potential savings of tens of millions of dollars.

We make the following observations about column generation approaches. First, high quality columns, that is, columns that are likely to be present in the optimal solution, and fewer generated columns can be achieved if interactions among columns are considered. This point is seen by comparing the performance of the naive and disaggregate information-enhanced column generation approaches. Second, better convergence and fewer generated columns can be achieved if a column in the restricted master problem includes fewer decisions. Specifically, in disaggregate information-enhanced column generation, each column in the *RMP* represents a single demand composite variable, indicating the selection of a small number of aircraft and routes. In contrast, each column in the *RMP* in aggregate information-enhanced column generation represents decisions for all aircraft routes at a hub.

In the service network design problem, using the disaggregate information-enhanced column generation approach, we generate a *set of columns* representing a solution to a *sub-network* of the overall network design problem. This approach greatly reduces the total number of columns generated, and is efficient in identifying columns that are likely to be in an optimal solution. We can extend this idea to other classes of problems. For example, in the multi-commodity network flow problem, we can establish at each iteration, a feasible flow in part of the network, instead of a single commodity flow.

Acknowledgements

The authors would like to thank Keith Ware and Alysia Wilson of UPS for providing test data and sharing their expertise in service network design. This research is supported by UPS.

References

1. Armacost, A., Barnhart, C., Ware, K.: Composite Variable Formulations for Express Shipment Service Network Design. Transportation Sci. **36** (2002) 1–20
2. Barnhart, C., Schneur, R. R.: Air Network Design for Express Shipment Service. Oper. Res. **44** (1996) 852–863
3. ILOG CPLEX 6.5 User's Manual, (1999)
4. Dantzig, G. B., Wolfe, P.: Decomposition Principle for Linear Programs. Oper. Res. **8** (1960) 101–111
5. Grünert, T., Sebastian, H. J.: Planning Models for Long-Haul Operations of Postal and Express Shipment Companies. Eur. J. Oper. Res. **122** (2000) 289–309
6. Kim, D., Barnhart, C., Ware, K., Reinhardt, G.: Multimodal Express Package Delivery: A Service Network Design Application. Transportation Sci. **33** (1999) 391–407
7. Krishnan, N., Barnhart, C., Kim, D., Ware, K.: Network Design for Express Shipment Delivery. Comput. Optim. Appl. **21** (2002) 239–262
8. Kuby M., Gray, R. G.: The Hub Network Design Problem with Stopovers and Feeders: The Case of Federal Express. Transportation Res. A **27A** (1993) 1–12
9. Leung L., Cheung W.: An Integrated Decision Methodology for Designing and Operating an Air Express Courier's Distribution Network. Decision Sci. **31** (2000) 105–126
10. Salomon S. B.: Equity Research: United Parcel Service Inc. (2000)
11. Shen, S.: Logistics Service Network Design: Models, Algorithms and Applications. Ph.D. Dissertation. Massachusetts Institute of Technology (2004)
12. UPS Annual Report (2002)
13. UPS Terms and Conditions of Service (2003)

University Course Timetabling

The University Course Timetabling Problem with a Three-Phase Approach

Philipp Kostuch

Oxford University, Department of Statistics,
1 South Parks Road, Oxford, OX1 3TG, UK
kostuch@stats.ox.ac.uk
http://www.stats.ox.ac.uk

Abstract. This paper describes the University Course Timetabling Problem (UCTP) used in the International Timetabling Competition 2003 organized by the Metaheuristics Network and presents a state-of-the-art heuristic approach towards the solution of the competition instances. It is a greatly improved version of the winning competition entry. The heuristic is divided into three phases: at first, a feasible timetable is constructed, then Simulated Annealing (SA) is used to order the thus created time-slots optimally, and finally SA is used to swap individual events between time-slots to improve the solution quality.

1 Timetabling Problems

Timetabling describes a problem that usually arises in an educational context (schools, universities, conferences). At its core is the task of placing entities generically called *events* (courses, classes, exams, lessons, talks) into a timetable made up of a limited number of atomic units called *time-slots*. Literature describing both timetabling models and solution algorithms abounds, for a recent overview see [18].

Here, we are concerned with a special class of timetabling problems called *course timetabling* [4]. The task is to place courses (e.g. university courses) in a weekly timetable, such that all students can attend all their courses, i.e. without requiring a student to attend two events at the same time (called a *student clash*). In this basic formulation, course timetabling is equivalent to graph colouring [9]. In addition to the universally present hard constraint of student clashes, there are more complex timetabling models in the literature adding further hard constraints–such as room availability–and soft constraints, that lay the foundation for an objective function to assess solution quality beyond mere feasibility [5].

With regards to algorithmic solution strategies, the easiest and least successful heuristics are constructive approaches, usually inspired by sequential graph colouring methods. For examples, see [6] and [3]. While often efficient in pure graph colouring environments these methods fail, on their own, to deal well with the complex nature of the objective function. Yet they play a major role in combination with many meta-heuristics, where they represent one of the main ways

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 109–125, 2005.

to incorporate domain-specific knowledge. Almost all meta-heuristic concepts have been applied to timetabling problems.

A simulated annealing approach on a larger, partially benchmarked, set of problems can be found in [22] and a tabu search approach with extensive analysis of the implemented algorithm is given in [10]. Hopfield neural networks are used in [19] with some original improvements to overcome the common problem of pre-mature convergence to local optima often seen in neural network optimization approaches. Various approaches based on genetic algorithms have been implemented. For some examples see [15] and [13]. A very detailed implementation of a CSP approach is presented in [8].

A number of implementations have originated in connection with the International Timetabling competition. A very recent implementation of ant colony optimization for timetabling problems is described in [21]. A highly successful hybrid approach, mainly based on simulated annealing, is given in [7]. Other entries included a great deluge algorithm [2] and a multi-neighbourhood local search [11].

2 The International Timetabling Competition

In 2002/03 the Metaheuristics Network organized an International Competition on a problem called University Course Timetabling Problem (UCTP) [17]. The competition sought to promote research in the field of timetabling and to establish a widely accepted benchmark problem for the timetabling community. The UCTP model is a reduction of a real-world timetabling problem encountered at Napier University, Edinburgh, UK. The (artificial) instances used for the competition were produced with an instance generator written by Paechter[1].

2.1 UCTP

The UCTP consists of a set of n events $E = \{e_1, \ldots, e_n\}$ to be scheduled in a set of 45 timeslots $T = \{t_1, \ldots, t_{45}\}$ (five days in a week, nine time-slots a day), and a set of j rooms $R = \{r_1, \ldots, r_j\}$ in which events can take place. Additionally, there is a set of students S who attend the events, and a set of features F provided by rooms and required by events. Each student attends a subset of events. A feasible timetable is one in which all events have been assigned a time-slot and a room, so that the following hard constraints are satisfied:

- no student attends more than one event at the same time;
- the assigned room is big enough for all the attending students and provides all the features required by the event;
- only one event is taking place in each room at a given time.

In addition, a feasible candidate timetable is penalized equally for each occurrence of the following soft constraint violations:

[1] `http://www.dcs.napier.ac.uk/~benp`

– a student has a class in the last slot of the day;
– a student has more than two classes in a row on a given day (one penalty for each class above the first two);
– a student has exactly one class during a day.

Infeasible timetables are worthless and are considered equally bad regardless of the actual level of in-feasibility. The objective is to minimize the number of soft constraint violations in a feasible timetable. For all instances under consideration, it is known that a zero penalty solution exists.

2.2 The Competition

The International Timetabling Competition began on October 1st 2002 with the publication of 10 instances of the UCTP. On March 17th, 2003 another 10 instances were released and solutions to all 20 instances had to be submitted by the end of March. Only entries with feasible timetables for all problems were considered. It is known that all instances have a feasible solution without soft constraint violations.

For an entry to be accepted, the rules stipulated the following:

> The algorithm should not take account of additional knowledge about the instance (e.g. results from previous runs or other processing). The same version (and parameters) of the algorithm must be used for all instances (i.e. the algorithm should not “know” which instance it is solving).

Nonetheless, submission of different random seeds for different instances was allowed, thus enabling the selection of the best run of the algorithm over a range of different seeds. The results had to be obtained within a given time limit (to be determined for every machine by running a provided benchmark program).

The results of the competition, descriptions of various entries and the exact rules for determining the winner can be found on the competition web page [1]. The official winner of the competition is an early version of the algorithm described in this paper. At that time, its results ranked second behind a cooperative effort of members of the Metaheuristic Network, which was not allowed to enter the competition.

The remainder of the paper is organized as follows. In Section 3, the algorithm used to solve the UCTP is described in detail. Section 4 presents the algorithm’s performance on the benchmark instances and an analysis of some aspects of the algorithm. Finally, the findings are summarized in Section 5.

3 The Algorithm

In this section, the algorithm used to solve the UCTP will be described in detail. At first, some general comments about the internal data structures will be made. Then, the three phases will be discussed in detail. Phase I of the algorithm tries to construct a feasible initial timetable within 40 time-slots, leaving the five

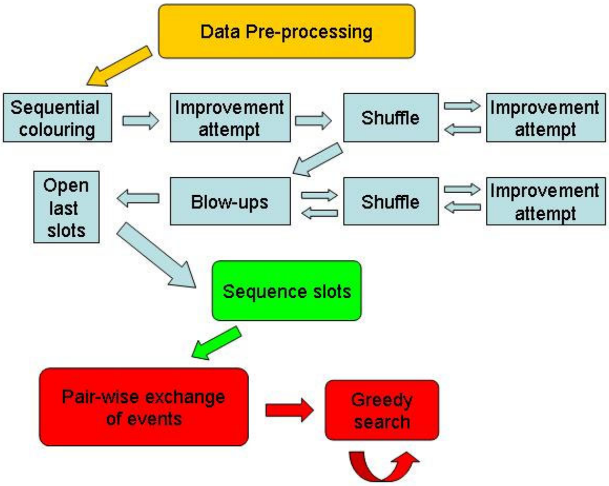


Fig. 1. Flow chart of the three-phase algorithm for UCTP

penalized end-of-day slots empty. In Phase II, the time-slots created before are sequenced using Simulated Annealing (SA) in order to minimize the objective function value. Once this is done, Phase III swaps individual events between time-slots, always maintaining feasibility. This is first done by a standard SA approach. Then–to obtain the final solution–the same neighbourhood structure is used at lower temperatures with periodical restarts at the best solution found so far. A flowchart representation of the algorithm is given in Figure 1.

3.1 Pre-processing and Data Structures

As there is a time limit for finding the solutions, besides a powerful algorithmic concept, efficient data structures and internal representations of the problem are needed to obtain competitive results. While some design choices presented in the following will be helpful for many algorithms, others are tailored more closely to the needs of this particular approach and will therefore not necessarily provide benefits in other settings.

The first step is to reduce the whole information contained in the instance file to a compact format. Besides the generic information provided (number of students, events and rooms) everything else can be represented in two matrices, the *incidence* matrix and the *event–room* matrix. The incidence matrix describes the hard constraints stemming from student clashes: we imagine a graph where events are nodes and edges indicate that two events share at least one student and can therefore not be scheduled together. The matrix of dimension $(events \times events)$ representing this graph is the incidence matrix. The event–room matrix has been compiled from the information about which features are needed by events, which are provided by rooms and whether the room size can accommodate a given event. This leads to a matrix of dimension (*events* × *rooms*), indicating which rooms are feasible for a given event.

Before starting to solve the problem, one useful manipulation of the incidence matrix is implemented. As we will accept only feasible moves in Phase III, every move has to be checked for feasibility before evaluating the objective function. There are two possible sources of infeasibility: either a student clash occurs or it is not possible to find suitable rooms for all events scheduled in a given time-slot. The first is computationally easy to detect by looking at the incidence matrix, while the second one needs a maximum matching algorithm for bipartite graphs to be run on a subset of rows from the event–room matrix. So, it is computationally attractive to shift information to the incidence matrix, if possible.

We observe that there are a substantial number of events with only one room option. Two such events having the same room as their only option can obviously not be scheduled together. Therefore, we search for the subsets of events with the same single room option and introduce edges into the incidence matrix so that these subsets form cliques in the graph, i.e. all possible edges between these events are present. The beauty of this step is that—while not altering the trajectory of the algorithm itself—it speeds up the detection of infeasibilities considerably.

A second manipulation, suggested in [20], goes a step further: based on the observation that some of these subsets of events sharing the same room as their single option have exactly 40 member events and the knowledge that a perfect timetable exists, further alterations are implemented. Given such a subset of size 40, we do not only know that these events form a clique in the incidence matrix, but can also conclude that in the perfect solution no other event is assigned to this particular room. If it were, we would need more than 40 time-slots to place all events from this subset conflict-free, thus incurring end-of-day penalties. This allows us to delete this room as an option for all other events from the event–room matrix. As this may create new single room option events, this step has to be implemented iteratively until there are no further changes. Unfortunately, this does not just speed up the algorithm, but changes its results, as flexibility in the possible assignments is lost. Experiments show that the algorithm presented here does suffer from this step, and it was therefore not used. Nonetheless, for other algorithms it may be beneficial.

Besides these two static matrices, we keep a number of dynamic data structures during the execution. First, we have an attendance matrix of dimension ($\mathit{students} \times 45$) that tells us for the current timetable in which time-slots a given student has events. Also, we have a matrix of dimension ($\mathit{students} \times 5$) that keeps track of the number of events per day for a given student. These two will allow us to implement a cheap evaluation of new timetables. To represent the current timetable, we have arrays identifying the time-slot and room of every event, as well as a representation that gives us the events currently placed in a particular time-slot. This last information reduces the computational effort when checking the feasibility of neighbourhood moves.

3.2 Phase I: Feasibility

With these preparations completed, we can enter Phase I, where we try to construct an initial feasible timetable using graph colouring and maximum matching algorithms. In this phase, we will try to assign events to time-slots and within the time-slots to rooms in such a way that no hard constraints are violated. At first, we will attempt to achieve this goal using only time-slots without penalty, thus implicitly catering for the soft constraint involving end-of-day events. If this fails, we open up the yet unused but available slots.

Initial Attempt. We start by using a sequential colouring algorithm for the graph given by the incidence matrix. The order in which events are considered for colouring is the degeneracy order [23]. The degeneracy order is found in the following way: at first, the event with the minimum degree in the original graph is identified and placed last in the degeneracy order. Then, this event is removed from the graph. In the remaining subgraph we, again, identify the minimum degree event, place it in the second-last position of the degeneracy order and remove it from the graph. This procedure is repeated until all events are placed in the degeneracy order.

The assignment of a time-slot (which we identify with the colours) to the event under consideration works in the following way: we determine all those slots among the initial 40 slots that are feasible for the event, i.e. there is no colour conflict with an already placed event. Among these options we assign a slot with the minimal number of events in it, provided the number of events in this time-slot does not exceed the number of rooms. Events that cannot be assigned to a time-slot are placed in a pool of unassigned events for further consideration.

After this initial assignment of time-slots, we run a maximum matching algorithm for bipartite graphs [14] for each time-slot. The bipartite graph is the subgraph of the event–room matrix where we only use those rows that correspond to the events assigned to the time-slot under consideration. Events that do not get a room in a maximum matching are removed from the list of events in this time-slot and go into the pool of events that need further consideration.

Improvement Attempt. We call the next phase *improvement attempt* and describe it here in full detail. It will be used in unchanged form later again. The idea is to take every unplaced event and scan the 40 existing time-slots: first we check whether there is a colour-conflict with a given time-slot and if there is none whether we can add the event to the time-slot such that a maximum matching algorithm can assign rooms to this event and to all events that were previously in the time-slot. If this is possible, the event is assigned to this time-slot and room and leaves the pool of unassigned events.

There are two mechanisms by which this may work at this point:

- events that could not find a time-slot and therefore entered the pool during the colouring phase of the initial attempt may now fit in some time-slots where events were removed during the room assignment phase.

– events removed during the room assignment phase may well fit into other time-slots.

Shuffling. For the events still left, there is no way of placing them conflict-free, unless we change existing assignments. We call the first step in this direction *shuffling*. For a number of iterations we look at every event from the pool of unplaced events and choose a time-slot at random. First we check whether there is a colour-conflict between the unplaced event and the chosen time-slot. If this is not the case, we provisionally assign the event to this time-slot and run a maximum matching algorithm. If, as a result, all events in this time-slot—including the added event—are assigned to a feasible room, we have managed to place an unassigned event. If not, there must be one event in this time-slot without room. We take this one event and put it into the pool of unplaced events (note that the number of events with feasible room assignments cannot drop below the initial number, hence there is at most one such event). The hope is that in such a case the newly unassigned event is different from the initially unassigned event. If this is true, we run the procedure called improvement attempt (see Section 3.2) for this event before placing it in the pool of unassigned events, hoping that it fits into a different time-slot. To increase our chance, that the initially unassigned event is part of the new maximum matching, we randomize the order in which nodes appear in the maximum matching algorithm.

Blow-Ups. The events that are still left must be considered as problematic and we have to go one step further in using invasive measures, that change the assignments made so far. The idea here is to force the placement of an unassigned event in an existing time-slot by first removing all events from this time-slot (*blow-up*). Then, we place the unassigned event in the now empty time-slot. Next, we fill up the time-slot with those removed events that do not have a colour-conflict with the initially unassigned event. Then we assign rooms to the events and place those events that had a colour-conflict or that were not given a room in the pool of unassigned events.

In this way, we accept changes that may lead to an increased number of unplaced events. After this change, we apply improvement attempt to all unassigned events and afterwards we run a number of repetitions of the shuffling step. After a certain number of blow-up steps (i.e. blow-ups combined with improvement attempt and shuffling), we restart the whole procedure at the assignment that was the best so far in terms of number of unassigned events. This is necessary, as often the number of unplaced events would otherwise just keep growing.

Opening the Last Slots. Events that are not placed despite the above efforts are distributed over the last five time-slots with the methods described above. If this does not place these events conflict-free in the five new time-slots, we could use the shuffling and blow-up steps on all 45 time-slots to overcome problems. In reality, however, this proves unnecessary, as the number of events left after the blow-ups are for all instances and all random seeds below 5, which guarantees finding a feasible initial assignment.

3.3 Phase II: Slot Sequence

Having found a feasible assignment which does in the majority of cases satisfy the soft constraint of not using end-of-day time-slots, we turn our attention towards the other two soft constraints, namely students with single events on a day and students with more than two events in a row. In Phase II of the algorithm, we take the time-slots formed during Phase I and try to sequence them optimally. The individual time-slots, i.e. the grouping of the events into 45 subsets, are not changed. For this optimization we will use Simulated Annealing (SA) [12].

The solution space are all possible permutations of the time-slots and the neighbourhood move is swapping two entries of the permutation. This formulation automatically preserves feasibility. Improving moves and side steps are always accepted. Deteriorating moves are accepted with a probability

$$P(Acceptance|\Delta, T) = \mathrm{e}^{-\frac{\Delta}{T}}$$

where Δ is the size of the deterioration in the objective function and T the current temperature. There are 35 different temperature levels T, starting at $T_{start} = 6.5$ and going down to $T_{end} = 1.5$. The schedule used is a standard geometric cooling, this means

$$T_{n+1} = T_n \times (1 - \delta_t)$$

where $\delta_t \ll 1$ is determined in such a way, that T_{end} is reached after 35 steps. At a given temperature, all possible moves are tried in a deterministic order, independently of the acceptance or rejection of earlier moves. We note that this simulated annealing problem has a connected search space and that the neighbourhoods are equi-sized.

Our selection process differs in two respects from the standard. Firstly, we choose to scan the neighbourhood deterministically instead of randomly. Secondly, we do not react in any way if a move is accepted, i.e. we do not restart the neighbourhood scanning in this new solution but continue with the predefined deterministic neighbourhood search from this new solution. We choose this approach to reduce computation time. We can be sure that every possible pair exchange is considered at least once per temperature without running a large number of random steps. This is acceptable as this phase is only a supportive measure for the main part of the algorithm, which is described in the next section. It is not desirable to spend much computation time in order to improve the result of this phase as it is very likely that by fixing the time-slots in this feasible but arbitrary way, the optimal sequence will have an objective function value far away from the overall achievable optimum of 0.

3.4 Phase III: Exchange of Events

This phase forms the main part of the algorithm, both in terms of time consumption as well as impact. The basic idea is again a local search guided by Simulated Annealing. The starting position in the solution space will be the best timetable found by sequencing. The neighbourhood is defined by all timetables where exactly one pair of events has swapped time-slots. In comparison with other neighbourhoods suggested, see for example [16], this is a very simple structure. We only accept moves where this swap does not lead to infeasibilities, be it student clashes or events that cannot be assigned a room within their time-slot.

As we have in most instances managed to put all events into 40 time-slots, this insistence on feasibility while only considering pair swaps, seriously curtails the algorithm's flexibility. After all, there are no events that could be swapped in the five end-of-day slots. Instead of complicating the algorithm by introducing another neighbourhood move that changes only one event, we introduce 10 so-called *dummy* events. These are additional events without students attending them that can go in all rooms. We place two of them in every end-of-day time-slots, so that they can take part in the swapping of events. Apart from keeping the simple neighbourhood structure, this allows us also to control the maximum number of events in end-of-day time-slots. In the final timetable, these dummy events are just removed. This has the same flavour as allowing temporary infeasibilities to occur (if we think of end-of-day penalties as hard constraints), in the hope that they will be correctable later but allowing the local search in the meantime to discover more attractive regions of the search space.

A second manipulation of the problem deals with *empty* events, i.e. original events that have no students. One could think of this as a flaw in the instance generator, but there is actually a reasonable interpretation for them: they are events whose audience cannot be determined in advance. So, they should not in any way affect the objective function, yet they have to be placed in the timetable. In the original formulation, these events may not be able to go into all rooms. This is changed for the reminder of the search. Of course, this has to be corrected in the final timetable. Those empty events that end up in a room that is infeasible have to be placed somewhere else in the timetable. This is relatively easy as they clash with no other events. They can also go into end-of-day slots without causing penalties. It turns out that this repair of the final timetable was always possible and no timetable had to be discarded because of this. The reason why this helps is that valuable room resources in the not-penalized time-slots are not blocked by events that could be placed in end-of-day slots without incurring penalties.

Main Schedule. As mentioned above, we use a SA approach to optimize the current timetable with a neighbourhood that is defined by all feasible swaps of two events. Improving moves and side steps are always accepted. Deteriorating moves are accepted with a probability

$$P(Acceptance|\varDelta, T) = \mathrm{e}^{-\frac{\varDelta}{T}}$$

where $\varDelta$ is the size of the deterioration in the objective function and T the current temperature. We start at $T_{start} = 4$ and go down to $T_{end} = 0.36$. The cooling schedule we use between these two temperatures is not the standard geometric cooling. Instead, in our schedule the inverse temperature T^{-1} grows linearly. This leads to

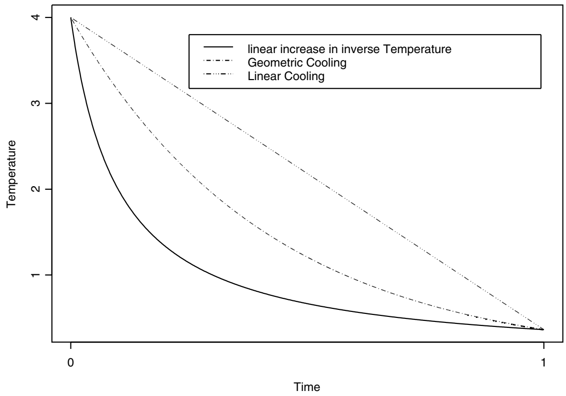


Fig. 2. Three different cooling regimes for simulated annealing

$$T_{n+1}^{-1} = T_n^{-1} + \delta_t \equiv T_{n+1} = \frac{1}{T_n^{-1} + \delta_t} \approx T_n \times (1 - \delta_t \times T_n)\,.$$

So, in contrast to geometric cooling, the cooling factor between two steps depends on the current temperature. The two different schedules plus a linear schedule are shown in Figure 2. This schedule was chosen as preliminary tests showed it to perform better than geometric or linear cooling, mainly because a high starting temperature was found to be important for good results and the geometric schedule spends too little time in low temperature regions when used with a high T_{start}.

For a given temperature T, all possible moves are tried in a deterministic order, independently of the acceptance or rejection of earlier moves, i.e. if a move is accepted we do not restart the neighbourhood search in this new solution but continue the deterministic scanning order from the new solution. This makes for $\frac{events^2 - events}{2}$ moves per temperature level. We will call this quantity a *cycle.* The majority of moves in a cycle will be discarded as infeasible. Again, we do not follow the standard SA procedure of selecting neighbours at random.

In contrast to Phase I, it is not practical to preset the number of temperature steps between T_{start} and T_{end}. There are huge differences in time demand from instance to instance for the completion of one cycle. Therefore, we first try to get an on-line prediction of the time demand for a given instance. For a fixed time (roughly 5% of the available time) we run the SA routine at T=0.15 recording how many cycles can be completed in this time. We then determine the number of temperature levels or cycles to be used in this phase, so that at its end 80% of the available time has been used, i.e. 20% remains for the last part of phase III.

Earlier, we described how we manipulated the incidence matrix to speed up the detection of infeasible moves by translating room conflicts into student clashes. But swaps that do not cause student clashes have still to be checked for whether rooms can be assigned. As mentioned above, a standard maximum matching algorithm for bipartite graphs between events and rooms in a time-slot is used. The simple structure of our neighbourhood allows us to save time on this. In order to determine whether we can assign rooms to the old events in a time-slot plus the new one minus the one that has left, we do not start the maximum matching algorithm from scratch. Instead, we start the matching algorithm with the partial assignment of the old events to their rooms. This way, there is at most one augmenting path to be found, which greatly reduces the time demand.

For the actual evaluation of feasible moves, we use a fast Δ-evaluation. Instead of re-evaluating the whole timetable, we make maximum use of the old value plus our knowledge of the neighbourhood structure. First, we determine all the students that are in one of the two events to be swapped, but not in both. Only these students are relevant for the change in the objective function value. We then use their rows in the old attendance matrix, the matrix describing for every student in which time-slot he has events, to determine the change caused by the swap. We only have to look two time-slots up and two time-slots down in this matrix to decide on the change in penalty from consecutive events. The penalty in single events per day can easily be determined from the matrix giving for every student the number of appearances per day. Changes in end-of-day penalties are straightforward to calculate. If a move is accepted, the data are updated before the next move.

Greedy Search. As a final step to improve the timetable, we run the SA search as described above with a different schedule until the time limit is reached. The starting temperature is $T_{start} = 0.5$ and the end temperature is $T_{end} = 0.1$ with 100 temperature steps between them. This phase is meant to greedily optimize the best solution found so far, placing emphasis on exploitation instead of exploration. At the end of the schedule, the greedy search is restarted, but not in the current solution but in the best found so far. This is repeated until the time limit is reached.

4 Results

In this section, the performance of the algorithm on the 20 competition instances will be presented. For every instance, 50 runs with different random seeds are recorded. If not indicated otherwise, quantitative results will be given as the mean over these 50 runs.

Table 1 presents the distribution of the computation time over the different phases in percent. Note that, due to rounding errors, the results may not add up to 100%. Those runs that were prematurely aborted, because an optimal solution was discovered within the time limit, have not been considered for in this table. We clearly notice the dominating role that the main schedule of Phase III plays in the algorithm. We also see that our online time prediction is reasonably accurate. We wanted to save 20% of the time for the greedy part of Phase III and have managed to design the main schedule accordingly. Phase

Table 1. Run time distribution over the different phases in percent of total time, for the 20 competition instances

Instance	1	2	3	4	5	6	7	8	9	10
Phase I	3.4	1.5	8.4	6.5	8.6	0.5	0.4	1	0.7	7.9
Phase II	2.5	2.5	2.5	3.6	3.6	3.6	4.2	2.9	2.7	2.5
Phase III Main	74.9	70.7	70.1	72.6	72.7	80.1	80.1	71.8	72.5	72
Phase III Greedy	19.1	25.3	19	17.3	15.1	15.8	15.3	24.3	24.1	17.7
Instance	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Phase I	5.3	8.4	6.7	2.7	1.6	0.3	12.3	1.5	3.8	0.2
Phase II	2.7	2.5	3.1	4.2	3.6	2.7	3.6	3	3.6	3.3
Phase III Main	72.1	71.2	71	78	79.8	68.5	69.3	80.6	74.8	74.1
Phase III Greedy	20	17.9	19.1	15.1	14.9	28.5	14.8	14.9	17.7	22.4

II, the sequencing of the initial time-slots, takes up only a small portion of the overall time. Interestingly, the variation in the time demand for Phase II between the instances is almost perfectly correlated with the number of students in the instance (correlation value of 0.97). The time demand of Phase I varies greatly among instances, indicating that finding a feasible timetable is not equally hard for the competition instances.

This claim is also supported by the first row of Table 2. Here, we see the average number of events that could not be placed into the 40 time slots, i.e. that had to go into end-of-day slots. For a number of instances, all runs succeeded in placing all events, but for others there were regularly events left. We note that even for the events with higher average values, such as instances 5 and 17, there were never more than five events left. So, finding a feasible timetable was always possible. The time Phase I takes and the average number of end-of-day events is with 0.93 highly correlated, supporting the notion that variation in time demand for Phase I is due to varying difficulty of successful initialization.

At this point we note that the competition instances are well-behaved in terms of constrainedness. It is perfectly possible and maybe more realistic to have instances where the generation of feasible initial timetables is harder, especially when using only 40 instead of 45 time-slots. Evidently, the algorithm presented here would have to be modified in order to deal with such a situation, as it depends on the successful completion of Phase I. There is currently work in progress to use Grouping Genetic Algorithms to generate initial timetables for UCTP instances where the presented method fails.

The further rows of Table 2 show the objective function value of the best solution found so far at the end of the respective phase. Phase I values are not yet optimized and represent therefore some kind of random sample from the feasible solution space. Unsurprisingly, the values are again almost perfectly correlated (values of 0.98) with the number of students. This could be expected from a random sample, as all penalties are tied to students, and therefore the more students there are the more penalty we can incur. Phase II manages to cut

Table 2. Results at the end of the individual phases, for the 20 competition instances

Instance	1	2	3	4	5	6	7	8	9	10
Phase I: end-of-day events	0.3	0.0	2.4	0.6	3.1	0.0	0.0	0.0	0.0	1.8
Phase I	537	493	545	760	846	800	859	634	547	539
Phase II	245	219	262	350	392	347	365	271	249	261
Phase III Main	32	15.9	32.6	61.6	73.8	6.6	11.1	6.5	12.3	36.4
Phase III Greedy	30.2	11.4	31	60.8	72.1	2.4	8.9	2	5.8	35
Instance	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Phase I: end-of-day events	0.6	2.7	1.2	0.2	0.0	0.0	3.3	0.0	0.3	0.0
Phase I	557	535	662	889	761	582	820	516	786	761
Phase II	259	246	301	368	325	274	387	225	367	330
Phase III Main	16.9	78.1	48.6	27	12	9.7	56.4	14.2	19.6	3.5
Phase III Greedy	12.9	76.3	47.1	22.3	8.4	3.4	54	9.4	16.4	0.5

the objective function value drastically by ordering the time-slots. An interesting observation is that the improvement achieved in this phase is very similar for all instances. The resulting values are highly concentrated around 45% of the starting value.

Evidently, the main schedule of Phase III has the most significant impact on the objective function. The values of the previous phases show no correlation with these values. The greedy part of Phase III can still improve these results, but it seems that its impact is higher on those instances that have already a low penalty value.

Before we discuss the final results in more detail, including the best values found, we take a quick look at the effect of the dummy events introduced in Section 3.4. In Figure 4 we see a panel of all 20 instances. In every panel box, we can see the average performance over 20 runs for a varying number of dummy events, indicated by the circle. The crosses mark the best solution found over 20 runs. We realize that the effect is not homogeneous over the instances. Nevertheless, we try to make some general comments. In general, too many dummy events lead to a loss in performance, clearly so for 20 dummy events, and also to a lesser extent for 15. Since the more dummy events, the greater the flexibility in the exchange phase, we must introduce other information to understand this result.

The time demand to complete a single cycle of the exchange phase, i.e. trying to exchange all possible pairs of events, increases steeply with the number of dummy events. In other words, as the number of dummy events goes up, the number of cycles that can be completed within the time limit goes down. This effect is very pronounced, as exchanges involving one dummy event are much more likely to be feasible. Therefore, their evaluation is much more likely to involve a fitness function evaluation. In a situation where this is a complexity driver, this leads to increasing time demand. So, we can say that the gains in flexibility have to be balanced with the losses in speed.

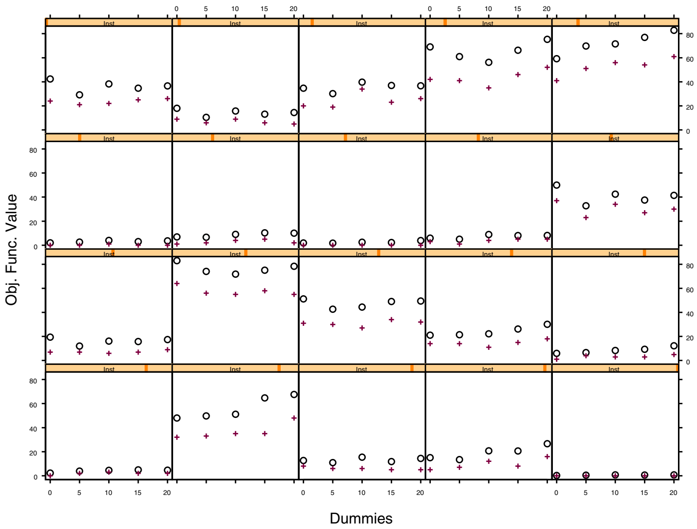


Fig. 3. Best and average performance for varying number of dummy events, for the 20 competition instances

Table 3. Best results from different algorithms, for the 20 competition instances

Instance	1	2	3	4	5	6	7	8	9	10
Official Competition Winner	45	25	65	115	102	13	44	29	17	61
Best Algorithm at the Competition	57	31	61	112	86	3	5	4	16	54
Chirandini et al.	45	14	45	71	59	1	3	1	8	52
3-phase approach	**16**	**2**	**17**	**34**	**42**	**0**	**2**	**0**	**1**	**21**
Instance	11	12	13	14	15	16	17	18	19	20
Official Competition Winner	44	107	78	52	24	22	86	31	44	7
Best Algorithm at the Competition	38	100	71	25	14	11	69	24	40	0
Chirandini et al.	30	75	55	18	8	5	46	24	33	0
3-phase approach	**5**	**55**	**31**	**11**	**2**	**0**	**37**	**4**	**7**	**0**

On the other hand, there is a good number of instances where the total absence of any dummy events will cause a sub-optimal performance. Most notably against this trend is the performance on instance 5. The graphs do indeed suggest that a low number of dummy events, i.e. 5 or 10, will be best for the overall performance, although maybe not optimally for certain instances. As the final choice was based on best and not mean performances, 10 dummy events were used in the final version of the algorithm, although 5 looks better on the mean values.

Figure 4 shows boxplot graphics for the 50 runs per competition instance. The boxes show the limits of the middle half of the data with the line inside

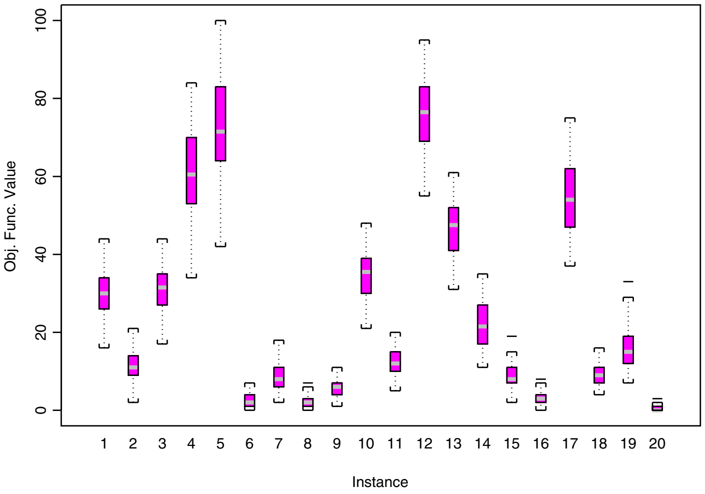


Fig. 4. Boxplots showing the 50 individual runs on the 20 competition instances

the box representing the median. The whiskers indicate extreme points. It gives us an idea about the variability of the performance for different random seeds. Clearly, the variability increases with the mean values. It also shows us that the best results for the instances are not lucky outliers, but fit very well to the overall distribution of results.

Table 3 presents the best values obtained by different algorithms. The official competition winner is an early version of the three-phase approach presented here, the best algorithm at the competition is an early version of the Chirandini et al. algorithm [7]. Both algorithms were in the aftermath of the competition substantially improved. This has led to significantly enhanced performances. The results for both up-to-date version are taken from 50 different seeds runs. The three-phase approach yields the best result achieved on all 20 instances. On four of them, it finds an optimal timetable.

5 Conclusion

This paper presents a state-of-the-art algorithm for the University Course Timetabling Problem (UCTP) based on Simulated Annealing. The results clearly outperform the next best algorithm known for UCTP. Yet, there remains plenty of scope for further research and improvement. Only for four out of 20 instances has a global optimum been found so far and, on some instances, the best objective function value is still far away from 0.

It is shown that, with a straightforward approach, it is possible to construct initial feasible timetables that use few or none of the penalized time-slots. Furthermore, a simple neighbourhood seems to suffice in order to achieve excellent results in the optimization phase.

The computational results for the so-called dummy events indicate that the temporary acceptance of infeasible moves (if we think of end-of-day penalties as hard constraints) or at least moves that are clearly hurting the objective function, has beneficial effects by increasing the algorithm's flexibility. Yet, the amount of such moves or the degree of infeasibility accepted has to be balanced carefully.

Acknowledgements

Philipp Kostuch acknowledges the support of a Marie Curie Training Site fellowship funded by the Improving Human Potential (IHP) programme of the Commission of the European Community (CEC), grant HPRN-CT-2000-00032, as well as the ongoing support of the Rhodes Trust. Special thanks go to Krzysztof Socha for fruitful discussions and valuable tips.

References

1. www.idsia.ch/Files/ttcomp2002.
2. Burke, E., Bykov, Y., Newall, J., Petrovic, S.: A Time-Predefined Approach to Course Timetabling. Yugoslav J. Oper. Res. **13** (2003) 139–151
3. Carter, M. W., Johnson, D. G.: Extended Clique Initialisation in Examination Timetabling. J. Oper. Res. Soc. **52** (2001) 538–544
4. Carter, M. W., Laporte, G.: Recent Developments in Practical Course Timetabling. In: Burke, E., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 3–19
5. Carter, M. W., Laporte, G., Chinneck, J. W.: A General Examination Scheduling System. Interfaces. **24** (1994) 109–120
6. Carter, M. W., Laporte, G., Lee, S.-Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
7. Chiarandini, M., Socha, K., Birattari, M., Rossi-Doria, O.: An Effective Hybrid Approach for the University Course Timetabling Problem. Technical Report AIDA-03-05. FG Intellektik, TU Darmstadt (March 2003)
8. Deris, S., Omatu, S., Ohta, H.: Timetable Planning Using the Constraint-Based Reasoning. Comput. Oper. Res. **27** (2000) 819–840
9. de Werra, D.: The Combinatorics of Timetabling. Eur. J. Oper. Res. **96** (1997) 504–513
10. Di Gaspero, L., Schaerf, A.: Tabu Search Techniques for Examination Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 104–117
11. Di Gaspero, L., Schaerf, A.: A Multineighbourhood Local Search Solver for the Timetabling competition TTComp 2002. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 475–478
12. Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P.: Optimization by Simulated Annealing. Science **220** (1983) 671–680

13. Paechter, B., Rankin, R. C., Cumming, A., Fogarty, T. C.: Timetabling the Classes of an Entire University with an Evolutionary Algorithm. In: Eiben, A. E., Schoenauer, M., Schwefel, H.-P. (eds.): Parallel Problem Solving From Nature—PPSN V (Amsterdam). Springer, Berlin (1998)
14. Papadimitriou, C. H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover, New York (1998)
15. Ross, P., Hart, E., Corne, D.: Some Observations About GA-based Exam Timetabling. In: Burke, E., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 115–129
16. Rossi-Doria, O., Blum, C., Knowles, J., Sampels, M., Socha, K., Paechter, B.: A Local Search for the Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): PATAT 2002—Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling 124–127
17. Rossi-Doria, O., Sampels, M., Chiarandini, M., Knowles, J., Manfrin, M., Mastrolilli, M., Paquete, L., Paechter, B.: A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 329–351
18. Schaerf, A.: A Survey of Automated Timetabling. Artif. Intell. Rev. (1999) 87–127
19. Smith, K. A., Abramson, D., Duke, D.: Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparative Results. Int. J. Comput. Indust. Eng. **44** (2003) 283–305
20. K. Socha, K.: $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for International Timetabling Competition. Technical Report TR/IRIDIA/2003-30. Université Libre de Bruxelles, Belgium (September, 2003)
21. Socha, K., Knowles, J., Sampels, M.: A Max–Min Ant System for the University Timetabling Problem. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.): Proc. ANTS 2002—3rd Int. Workshop on Ant Algorithms. Lecture Notes in Computer Science, Vol. 2463. Springer, Berlin (2002) 1–13
22. Thomson, J. M., Dowsland, K. A.: A Robust Simulated Annealing Based Examination Timetabling System. Comput. Oper. Res. **25** (1998) 637–648
23. West, D. B.: *Introduction to Graph Theory.* 2nd edn. Prentice-Hall, Englewood Cliffs, NJ (2001)

Minimal Perturbation Problem in Course Timetabling

Tomáš Müller[1], Hana Rudová[2], and Roman Barták[1]

[1] Faculty of Mathematics and Physics, Charles University,
Malostranské nám. 2/25, Prague, Czech Republic,
{muller, bartak}@ktiml.mff.cuni.cz

[2] Faculty of Informatics, Masaryk University,
Botanická 68a, Brno 602 00, Czech Republic,
hanka@fi.muni.cz

Abstract. Many real-life problems are dynamic, with changes in the problem definition occurring after a solution to the initial formulation has been reached. A minimal perturbation problem incorporates these changes, along with the initial solution, as a new problem whose solution must be as close as possible to the initial solution. A new iterative forward search algorithm is proposed to solve minimal perturbation problems. Significant improvements to the solution quality are achieved by including new conflict-based statistics in this algorithm. The proposed methods were applied to find a new solution to an existing large scale class timetabling problem at Purdue University, incorporating the initial solution and additional input changes.

1 Introduction

Most existing solvers are designed for static problems. These problems can be expressed, solved by appropriate means, and the solution applied without any change to the problem statement. Many real-life problems [18], [11], [9], [16], [12], however, are subject to change. Additional input requirements produce a new problem derived from the original problem. The dynamics of such a problem may require changes during the solution process, or even after a solution is generated. In many real-life situations, it is necessary to alter the solution process so that the dynamic aspects of the problem definition are taken into account.

Problem changes may result from changes to environmental variables, such as broken machines, delayed flights, or other unexpected events. Users may also specify new properties based on the solution found so far. The goal is to find an improved solution for the user. Naturally, the problem solving process should continue as smoothly as possible after any change in the problem formulation. In particular, the solution of the altered problem should not differ significantly from the solution found for the original problem. There are several reasons to keep a new solution as close as possible to the existing solution. If the solution has already been published, such as the assignment of gates to flights, frequent changes would confuse passengers. Moreover, changes to a published solution

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 126–146, 2005.

may necessitate other changes if initially satisfied wishes of users are violated by the proposed changes. This may create an avalanche reaction.

In this paper we focus on handling dynamic changes in course timetabling. In particular, our work is motivated by the class timetabling problem at Purdue University [15]. Here timetables for each semester are created nearly a semester in advance. Once timetables are published they require many changes based on additional input. These changes must be incorporated into the problem solution with minimal impact on any previously generated solution. Thus, the primary focus of our work is to provide support for making minimal changes to the generated timetable.

Our problem solver is based on constraint satisfaction techniques [4] which are frequently applied to solve timetabling problems [7], [3], [15], [12]. Moreover, dynamic constraint satisfaction [18], [11], [17] is able to cover dynamic aspects in the problem. The minimal perturbation problem as defined in [1], [16] allows us to express our desire to keep changes to the solution (perturbations) as small as possible.

We introduce a new iterative forward search algorithm to solve the minimal perturbation problem. It is based on our earlier work on solving methods for the static (initial) problem [12]. The method also allows us to solve the initial problem. The basic difference in application is that the optimization of the number of changes (perturbations) is not included while solving the initial problem. Our algorithm is close to local search methods [10]; however, it maintains partial feasible assignments as opposed to the complete conflicting assignments characteristic of local search. Similar to local search, we process local changes in the assignment. This allows us to generate a complete solution and to improve the quality of the assignment at the same time.

New conflict-based statistics are proposed to improve the quality of the final solution. Conflicts during the search are memorized and their potential repetition is minimized. Conflict-based heuristics have been successfully applied in earlier works [5], [8]. In our approach, the conflict-based statistics work as advice in the value selection criterion. They help to avoid repetitive, unsuitable assignments of the same value to a variable by memorizing conflicts caused by this assignment in the past. The proposed heuristics do not limit the number of conflicts and assignments that are memorized. We have extended our search algorithm using these conflict-based statistics. Note, however, that this is a general strategy that can be applied in other problem solvers.

The paper is organized as follows. Section 2 describes the timetabling problem at Purdue University that motivates our work. The subsequent section introduces the minimal perturbation problem and surveys related works on dynamic problems. Section 4 describes the iterative forward search algorithm. The subsequent section introduces conflict-based heuristics and defines how they have been included in the search algorithm. The solution of our class timetabling problem is discussed in Section 6. A short summary of the implemented system, along with experimental results for the initial and minimal perturbation problems, conclude the paper.

2 Motivation—Timetabling Problem

The primary purpose of our work is to solve a real timetabling problem at Purdue University (USA). Here the timetable for large lecture classes is constructed by a central scheduling office in order to balance the requirements of many departments offering large classes that serve students from across the university. Smaller classes, usually focused on students in a single discipline, are timetabled by "schedule deputies" in the individual departments. Such a complex timetabling process, including subsequent student registration, takes a rather long time. Initial timetables are generated about half a year before the semester starts. The importance of creating a solver for a dynamic problem increases with the length of this time period and the need to incorporate the various changes that arise.

Rescheduling of classes in the timetable for large lectures is the primary focus of this paper. This problem consists of about 830 classes (forming almost 1800 meetings) having a high density of interaction that must fit within 50 lecture rooms with capacities up to 474 students. Room availability is a major constraint for Purdue. Overall utilization of the time available in rooms exceeds 78%; moreover, it is around 94% for the four largest rooms. About 90,000 course requests by almost 30,000 students must also be considered. 8.4% of class pairs have at least one student enrolment in common.

The timetable maps classes (students, instructors) to meeting locations and times. A major objective in developing an automated system is to minimize the number of potential student course conflicts which occur during this process. This requirement substantially influences the automated timetable generation process since there are many specific course requirements in most programs of study offered by the University.

To minimize the potential for time conflicts, Purdue has historically subscribed to a set of standard meeting patterns. With few exceptions, 1 hour × 3 day per week classes meet on Monday, Wednesday, and Friday at the half hour (7:30, 8:30, 9:30, ...). 1.5 hour × 2 day per week classes meet on Tuesday and Thursday during set time blocks. 2 or 3 hours × 1 day per week classes must also fit within specific blocks, etc. Generally, all meetings of a class should be taught in the same location. Such meeting patterns are of interest to the problem solution as they allow easier changes between classes having the same or similar meeting patterns.

Currently, the timetable for Purdue University is constructed by a manual process. We have proposed an automated timetabling system to solve the initial problem [15]. This solution was based on constraint logic programming (CLP) with soft constraints. The CLP solver is currently undergoing comparison with the new solver described in this paper.

3 Minimal Perturbation Problem and Related Works

Dynamic problems appear frequently in real-life planning and scheduling applications where the task is to "minimally reconfigure schedules in response to a changing environment" [16]. Dynamic changes in the context of timetabling problems were first studied in [6]. Issues of interactive timetabling which needs to handle dynamic aspects of the problem were discussed in [3], [13], [12]. A survey of existing approaches to dynamic scheduling can be found in [9]. In an annotated bibliography included in the CP 2003 tutorial on dynamic constraint solving [17], it is notable that only four papers were devoted to the problem of minimal changes. An extended version of this tutorial will be published in [18]. The minimal perturbation problem was described formally in [16] and solved by a combination of linear and constraint programming. We have extended this definition in [1] and proposed a solution algorithm based on the Branch & Bound algorithm. An algorithm inspired by heuristic repair and limited discrepancy search has also been proposed in [14].

To define the minimal perturbation problem, we will consider an initial (original) problem, its solution, a new problem, and some distance function which allows us to compare solutions of the initial and the new problem. Subsequently we look for a solution of the new problem with minimal distance from the initial solution. Let us now look at particular components (for detailed information see [1]).

We can define both the initial and the new problem as a constraint satisfaction problem (CSP) [4]. It is a triple (V, D, C) where V is a finite set of variables, D is a set of possible values for variables (domain), and C is a finite set of constraints restricting the values of variables. A solution to a CSP is a complete assignment of the variables that satisfies all the constraints.

A distance function can be defined with the help of perturbations [1], [16], [14]. A perturbation is a variable that has a different value in the solutions of the initial and the new problem. Some perturbations must be present in each new solution. So called input perturbation means that a variable must have different values in the initial and changed problem because of some input changes (e.g., a course must be scheduled at a different time in the changed problem). The distance function can be defined as the number of additional perturbations. They are given by subtraction of the final number of perturbations and the number of input perturbations.

4 Iterative Forward Search Algorithm

In this section, an iterative forward search algorithm and its general setting are presented. It is based on local search methods [10], but in contrast to classical local search techniques, it operates over a feasible, though not necessarily complete, solution. In such a solution, some variables can be left unassigned; however, all hard constraints on assigned variables must be satisfied. Similar to backtracking-based algorithms, this means that there are no violations of hard constraints.

Working with feasible incomplete solutions has several advantages compared to the complete infeasible solutions that usually occur in local search techniques. For example, when the solver is not able to find a complete solution, a feasible

```
procedure SOLVE(initial)            // initial solution is the parameter
    iteration = 0;                  // iteration counter
    current = initial;              // current solution
    best = initial;                 // best solution
    while canContinue(current, iteration) do
        iteration = iteration + 1;
        variable = selectVariable(current);
        value = selectValue(current, variable);
        unassigned = CONFLICTING_VARIABLES(current, variable, value);
        UNASSIGN(current, unassigned);
        ASSIGN(current, variable, value);
        if better(current, best) then best = current
    end while
    return best
end procedure
```

Fig. 1. Pseudo-code of the search algorithm

one can be returned: i.e., a solution with the least number of unassigned variables found. Especially in interactive timetabling applications, such solutions are much easier to visualize, even during the search, since no hard constraints are violated. For instance, two lectures never use a single resource (e.g., a classroom) at the same time. Moreover, because of the iterative character of the search, the algorithm can easily start, stop, or continue from any feasible solution, either complete or incomplete.

The search is processed iteratively (see Figure 1 for the algorithm). During each step, either an unassigned or an assigned variable may be selected. Typically an unassigned variable is chosen. An assigned variable may be selected when all variables are assigned but the solution is not good enough, for example when there are still many violations of soft constraints. Once a variable is selected, a value from its domain is chosen for assignment. Even if the "best" value is selected, its assignment to the selected variable may cause some hard conflicts with already assigned variables. Such conflicting variables are removed from the solution and become unassigned. Finally, the selected value is assigned to the variable. The search is terminated when the desired solution is found or when there is a timeout, expressed for example as a maximal number of iterations or available time being reached. The best solution found is then returned.

Each current solution must be feasible at all times, but an assignment of a value to a variable may cause conflicts with other variables. For example, let the values of variables A, B and C must be different, and variable A is assigned the value 3. When variable B, together with the value 3, are selected during the following step, the value of A becomes unassigned during the assignment $B = 3$. In our algorithm, the function CONFLICTING_VARIABLES computes the set of conflicting variables that will be unassigned in the subsequent step.

The above algorithm schema is parametrized by several functions, namely

– the variable selection (function `selectVariable`),
– the value selection (function `selectValue`),

- the termination condition (function `canContinue`) and
- the solution comparator (function `better`).

These functions are discussed below.

Termination Condition. The termination condition determines when the algorithm should finish. For example, the solver should terminate when the maximum number of iterations or another given timeout value is reached. Moreover, it can stop the search process when the current solution is good enough (e.g., all variables are assigned and/or some other solution parameters are in the required ranges). As an example, the solver can stop when all variables are assigned and less than 10% of soft constraints are violated. The user may also terminate the process.

Solution Comparator. The solution comparator compares two solutions: the current solution and the best solution found. This comparison can be based on several criteria. For example, it can lexicographically order solutions according to the following criteria: the number of unassigned variables (a smaller number is better) or the number of violated soft constraints. Soft constraints can be weighted according to their importance and/or preferences. Then, a sum of weights of violated soft constraints can be used as the second criterion.

Variable Selection. As mentioned above, the presented algorithm requires a function that selects a variable to be (re)assigned during the current iteration step. This problem is equivalent to a variable selection criterion in constraint programming. There are several guidelines for selecting a variable [4]. In local search, the variable participating in the largest number of violations is usually selected first. In backtracking-based algorithms, the first-fail principle is often used: i.e., a variable whose instantiation is most complicated is selected first. This could be the variable involved in the largest set of constraints or the variable with the smallest domain, etc.

The variable selection criterion is split into two cases. If some variables remain unassigned, the first-fail principle can be applied as the basis for selection. Other choice could be the random selection of the unassigned variable. Because the algorithm does not need to stop when a complete feasible solution is found, the second case occurs when all variables are assigned but the solution does not meet the termination condition. Here we choose the variable for which a change of its value appears to offer the best opportunity for improvement of the solution. This may be, for example, a variable whose value violates the highest number of soft constraints.

Value Selection. After a variable is selected, we need to find a value to be assigned. This problem is usually called “value selection” in constraint programming [4]. Typically, the most useful advice is to select the best-fit value. So, we are looking for a value which is the most preferred for the variable and also which causes the least trouble during the future search. This means that we need to find a value with minimal potential for future conflicts with other variables. Note that we are not using constraint propagation explicitly in our algorithm. However, the power of constraint propagation can be substituted to some extent by sophisticated value selection. It can take into account possible future conflicts by analysing the past conflicts.

For example, a value which violates the smallest number of soft constraints among values with the smallest number of hard conflicts (i.e., the values whose assignment to the selected variable violate the smallest number of hard constraints) can be selected.

4.1 Adjustment for Solving a Minimal Perturbation Problem

Despite the local search nature of the IFS algorithm, there are some adjustments needed to effectively solve the MPP. The goal of these adjustments is to minimize the number of additional perturbations. The easiest way to do this is to adopt variable and value selection heuristics that prefer the previous assignments (but not always, to avoid cycling).

For example, value selection heuristics can be adopted to select the initial value (if it exists) randomly with a probability P_{init} (it can be rather high, e.g. 50–80%). In cases where the initial value is not selected, original value selection can be applied. Also, if there is an initial value in the set of best-fit values (e.g., among values with the minimal number of hard and soft conflicts), the initial value can be preferred here as well. Otherwise, a value can be selected randomly from the constructed set of best-fit values. A disadvantage of such heuristics is that the probability P_{init} has to be selected carefully: if it is too small, the search can easily move away and the number of additional perturbations will grow during the search. If it is too high, the search will stick too much with the initial solution and, if there is no solution with a small number of additional perturbations, it will be hard to find a feasible solution. We have achieved the best results using this probability P_{init} of around 60% (see Section 7.2 for details).

Variable selection heuristics can also assist in finding a solution with a small number of perturbations. For example, when all variables are assigned, a variable that is not assigned its initial value should be selected (e.g., randomly among all variables that are not assigned their initial values), and that participates in the highest number of violated soft constraints.

5 Conflict-Based Statistics

In this section, a very promising extension of the iterative forward search algorithm is presented. The idea behind it is to memorize conflicts and to avoid their potential repetition. When a value v_0 is assigned to a variable V_0, hard conflicts with previously assigned variables (e.g., $V_1 = v_1$, $V_2 = v_2, \ldots, V_m = v_m$) may occur. These variables $V_1, \ldots, V_m$ have to be unassigned before the value v_0 is assigned to the variable V_0. These unassignments, together with the reason for their unassignment (i.e., the assignment $V_0 = v_0$), and a counter tracking how many times such an event occurred in the past, is stored in memory.

Later, if a variable is selected for assignment again, the stored information about repetition of past hard conflicts can be taken into account: for example, in the value selection heuristics. Assume that the variable V_0 is selected for an assignment again (e.g., because it became unassigned as a result of a later assignment), we can weight the number of hard conflicts created in the past for each possible value of this variable. In the above example, the existing assignment $V_1 = v_1$ can prohibit the selection of value v_0 for variable V_0 if there is again a conflict with the assignment $V_1 = v_1$.

Conflict-based statistics are a data structure which memorizes the number of hard conflicts that have occurred during the search (e.g., that assignment $V_0 = v_0$ resulted c_1 times in an unassignment of $V_1 = v_1$, c_2 times of $V_2 = v_2$, ... and c_m times of $V_m = v_m$). More precisely, they form an array

$$CBS[V_a = v_a, V_b \neq v_b] = c_{\mathrm{ab}}$$

stating that the assignment $V_a = v_a$ caused the unassignment of $V_b = v_b$ a total of c_{ab} times in the past. Note that in case of n-ary constraints (where $n > 2$), this does not imply that the assignments $V_a = v_a$ and $V_b = v_b$ cannot be used together. The proposed conflict-based statistics do not actually work with any constraint, they only memorize unassignments and the assignment that caused them. Let us consider a variable V_a selected by the `selectVariable` function and a value v_a selected by `selectValue`. Once the assignment $V_b = v_b$ is selected by CONFLICTING_VARIABLES to be unassigned, the array cell $CBS[V_a = v_a, V_b \neq v_b]$ is incremented by one.

The data structure is implemented as a hash table, storing information for conflict-based statistics. A counter is maintained for the tuple $A = a$ and $B \neq b$. This counter is increased when the value a is assigned to the variable A and b is unassigned from B. The example of this structure

$$A = a \Rightarrow \begin{cases} 3 \times B \neq b \\ 4 \times B \neq c \\ 2 \times C \neq a \\ 120 \times D \neq a \end{cases}$$

expresses that variable B lost its assignment b three times and its assignment c four times, variable C lost its assignment a two times, and D lost its assignment a 120 times, all because of later assignments of value a to variable A. This structure is being used in the value selection heuristics to evaluate existing conflicts with the assigned variables. For example, if there is a variable A selected and if the value a is in conflict with the assignment $B = b$, we know that a similar problem has already occurred three times in the past, and hence the conflict $A = a$ is weighted with the number 3.

Then, a *min-conflict* value selection criterion, which selects a value with the minimal number of conflicts with the existing assignments, can be easily adapted to a *weighted min-conflict* criterion. The value with the smallest sum of the number of conflicts multiplied by their frequencies is selected.

Stated in another way, the weighted min-conflict approach helps the value selection heuristics to select a value that might cause more conflicts than another value, but these conflicts occurred less frequently, and therefore they have a lower weighted sum. As we will show in Section 7.1, this can help considerably with getting the search algorithm out of a local optimum.

Extensions. The presented approach can be successfully applied in other search algorithms as well. For example, in the local search, we can memorize the assignment $V_x = v_x$, which was selected to be changed (re-assigned). A reason for such selection can be retrieved and memorized together with the selected assignment $V_x = v_x$ as well. Note that typically an assignment which is in a conflict with some other assignments is selected.

Furthermore, the presented conflict-based statistics can be used not only inside the solving mechanism. The constructed 'implications' together with the information about frequency of their occurrences can be easily accessed by users or by some add-on deductive engine to identify inconsistencies[1] and/or hard parts of the input problem. The user can then modify the input requirements in order to eliminate problems found and let the solver continue the search with this modified input problem.

6 Solution for Timetabling Problem

In this section we will discuss an application of the above-described algorithm for the large lecture timetabling problem at Purdue University. The modelling part will be described first, followed by the description of the algorithm.

6.1 Problem Representation

Due to the set of standardized time patterns and administrative rules in place at the university, it is generally possible to represent all meetings of a class by a single variable. This tying together of meetings considerably simplifies the problem constraints. Most classes have all meetings taught in the same room, by the same instructor, at the same time of day. Only the day of week differs. Moreover, these days and times are mapped together with the help of meeting patterns: e.g., a 2 hours × 3 days per week class can be taught only on Monday, Wednesday, Friday, beginning at five possible times (7:30, 9:30, 11:30, 1:30, 3:30).

In addition, all valid placements of a course in the timetable have a one-to-one mapping with values in the variable's domain. This domain can be seen as a subset of the Cartesian product of the possible starting times, rooms, etc., for a class represented by these values. Therefore, each value encodes the selected

[1] Actually, this feature allows discovery of all inconsistent data inputs during solution of the Purdue University timetabling problem.

time pattern (some alternatives may occur: e.g., 1.5 hour × 2 day per week may be an alternative to 1 hour × 3 day per week), selected days (e.g., a two-meeting course can be taught in Monday–Wednesday, Tuesday–Thursday, Wednesday–Friday), and possible starting times. A value also encodes the instructor and selected meeting room. Each such placement also encodes its preferences (soft constraints), combined from the preference for time, room, building and available equipment of the room. Only placements with valid times and rooms are present in a domain. For example, when a computer (classroom equipment) is required, only placements in a room containing a computer are present. Also, only rooms large enough to accommodate all the enrolled students can be present in valid class placements. Similarly, if a time slice is prohibited, no placement containing this time slice is in the class's domain.

The variable and value encodings described above leave us with only two types of hard constraints to be implemented: resource constraints (expressing that only one course can be taught by an instructor or in a particular room at the same time), and group constraints (expressing relations between several classes: e.g., that two sections of the same lecture cannot be taught at the same time, or that some classes have to be taught immediately after another).

There are three types of soft constraints in this problem. First, there are soft requirements on possible times, buildings, rooms, and classroom equipment (e.g., computer or projector). These preferences are expressed as integers:

−2 . . . strongly preferred
−1 . . . preferred
0 . . . neutral (no preference)
1 . . . discouraged
2 . . . strongly discouraged

As mentioned above, each value, besides encoding a class's placement (time, room, instructor), also contains information about the preference for the given time and room. Room preference is a combination of preferences on the choice of building, room, and classroom equipment. The second group of soft constraints is formed by student requirements. Each student can enroll in several classes, so the aim is to minimize the total number of student conflicts among these classes. Such conflicts occur if the student cannot attend two classes to which he or she has enrolled because these classes have overlapping times. Finally, there are some group constraints (additional relations between two or more classes). These may either be hard (required or prohibited), or soft (preferred), similar to the time and room preferences (from −2 to 2).

6.2 Search Algorithm

In Section 4, we described four functions which parametrize the proposed algorithm. Here we will describe their exact settings in our timetabling solver.

The quality of a solution is expressed as a weighted sum combining soft time and classroom preferences, satisfied soft group constrains and the total number of student conflicts. This allows us to express the importance of different types of soft constraints. The following weights are considered in the sum:

W_{student} . . . weight of a student conflict,
W_{time} . . . weight of a time preference of a placement,
W_{room} . . . weight of a classroom preference of a placement,
W_{constr} . . . weight of a preference of a satisfied soft group constraint.

Note that preferences of all time, classroom and group soft constraints go from -2 (strongly preferred) to 2 (strongly discouraged). So, for instance, the value of the weighted sum is increased when there is a discouraged time or room selected or a discouraged group constraint satisfied. Therefore, if there are two solutions, the better of them has the lower weighted sum of the above criteria. Moreover, additional solution parameters can be included in this comparison as well. For instance, we can also discourage empty half-hour time segments between classes (such half-hours cannot be used since all events require at least one hour) or usage of classrooms that are too large (having more than 50% excess seats).

The termination condition stops the search when the solution is complete and good enough (expressed as the number of perturbations and the solution quality described above). It also allows for the solver to be stopped by the user. Characteristics of the current and the best achieved solution, describing the number of assigned variables, time and classroom preferences, the total number of student conflicts, etc., are visible to the user during the search.

The solution comparator prefers a more complete solution (with a smaller number of unassigned variables) and a solution with a smaller number of perturbations among solutions with the same number of unassigned variables. If both solutions have the same number of unassigned variables and perturbations, the solution of better quality is selected.

If there are one or more variables unassigned, the variable selection criterion picks one of them randomly. We have compared several approaches for variable selection using domain sizes, number of previous assignments, number of constraints in which the variable participates, etc. However, there was no significant improvement in this timetabling problem in comparison with the random selection of an unassigned variable. The reason could be that it is easy to go back when a wrong variable is picked—such a variable is unassigned when there is a conflict with it in some of the subsequent iterations.

When all variables are assigned, an evaluation is made for each variable according to the above-described weights. The variable with the worst evaluation is selected because this variable promises the best improvement in optimization.

We have implemented a hierarchical handling of the value selection criteria. There are three levels of comparison. At each level a weighted sum of the criteria described below is computed. Only solutions with the smallest sum are considered in the next level. The weights express how quickly a complete solution should be found. Only hard constraints are satisfied in the first level sum. Distance from the initial solution (MPP), and a weighting of major preferences (including time, classroom requirements and student conflicts), are considered in the next level. In the third level, other minor criteria are considered. Such a criterion could be for instance a usage of a room that is too large or a number of empty half-hour time segments between classes. In general, a criterion can be used in more than one level: for example, with different weights.

The above sums order the values lexicographically: the best value having the smallest first level sum, the smallest second level sum among values with the smallest first level sum, and the smallest third level sum among these values. As mentioned above, this allows diversification of the importance of individual criteria. In general, there can be more than three levels of these weighted sums, however three of them seem to be sufficient for spreading weights of various criteria for our problem.

The value selection heuristics also allow for random selection of a value with a given probability P_{rw} (random walk, e.g., 2%) and, in the case of MPP, to select the initial value (if it exists) with a given probability P_{init} (e.g., 60%).

Criteria used in the value selection heuristics can be divided into two sets. Criteria in the first set are intended to generate a complete assignment:

1. Number of hard conflicts, weighted by $V_{\mathrm{conf},1}$ in the first level, $V_{\mathrm{conf},2}$ in the second level and $V_{\mathrm{conf},3}$ in the third level.
2. Number of hard conflicts, weighted by their previous occurrences (see conflict-based statistics section) and by $V_{\mathrm{wconf},1..3}$.

Additional criteria allow better results to be achieved during optimization:

3. Number of student conflicts caused by the value if it is assigned to the variable, weighted by $V_{\mathrm{student},1..3}$.
4. Soft time preference caused by a value if it is assigned to the variable, weighted by $V_{\mathrm{time},1..3}$.
5. Soft classroom preference caused by a value if it is assigned to the variable (combination of the placement's building, room, and classroom equipment compared with preferences), weighted by $V_{\mathrm{room},1..3}$.
6. Preferences of satisfied soft group constraints caused by the value if it is assigned to the variable, weighted by $V_{\mathrm{constr},1..3}$.
7. Difference in the number of assigned initial values if the value is assigned to the variable (weighted by $V_{\Delta\mathrm{init},1..3}$): -1 if the value is initial, 0 otherwise, increased by the number of initial values assigned to variables with hard conflicts with the value.

Let us emphasize that the criteria 3–7 are needed for optimization only, i.e. they are not needed to find a feasible solution.[2] Furthermore, assigning a different weight to a particular criterion influences the value of the corresponding objective function (see Figure 3 with comparison between criteria 3 and 4). The solver returns good results in reasonable time (e.g., in 30 minutes time limit) when the total sum of the weights used in additional criteria (3–7) in the first level corresponds to one-half of the weight $V_{\mathrm{wconf},1}$ (2). The weights in the second level usually correspond to the weights used for the solution quality comparison (W_{student}, W_{time}, W_{room}, and W_{constr}).

[2] A feasible solution must satisfy hard constraints.

7 Implementation and Experiments

The timetabling system is implemented in Java. It contains a general implementation of the iterative forward search algorithm described above. The general solver operates over variables and values with a selection of basic general heuristics, comparison, and termination functions. It may be customized to fit a particular problem (as it has been extended for Purdue University timetabling) by implementing variable and value definitions, adding hard and soft constraints, and extending the algorithm's parametric functions.

Besides the above discussed solver, the timetabling application for Purdue University also contains a web-based graphical user interface (written using Java Server Pages) which allows management of several versions of the data sets (input requirements, solutions, changes, etc.), browsing the resultant solutions (see Figure 2), and tracking and managing changes between them.

The following experiments were performed on the complete Fall 2004 data set, including 830 classes to be placed in 50 classrooms. The classes included represent 89,677 course requirements for 29,808 students. The results presented here were computed on 1GHz Pentium III PC running Windows 2000, with 512 MB RAM and J2SDK 1.4.2. We have achieved similar results with Fall 2001 and Spring 2005 data sets as well, even though they are quite different in the number of requirements (Fall 2004 is the most constrained one out of these three data sets).

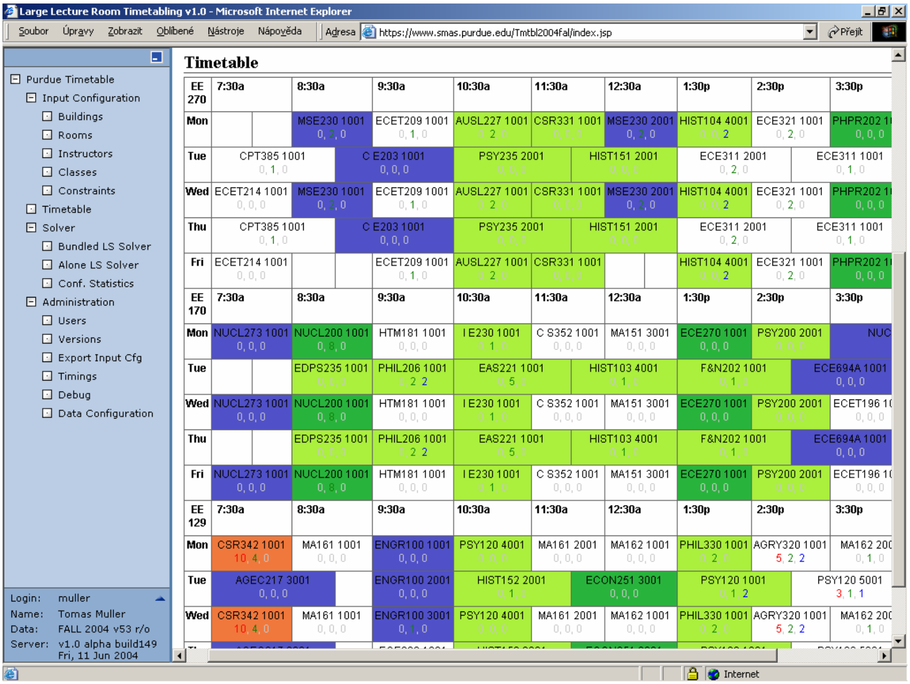
Below, we present two types of experiments. The first experiment investigates finding an initial solution This is followed by experiments on the minimal perturbation problem (i.e., where there is an existing solution plus a set of changes to be applied to it). Solving an initial problem can be seen as a special case of MPP where all variables are new and therefore have no initial values.

If not stated otherwise, the solution quality weights W_{student}, W_{time}, W_{room} and W_{constr} in the solution quality weighted sum are set to zero in the following experiments. First level weight for the weighted hard conflicts $V_{\mathrm{wconf},1}$ is set to 1, all other weights in the value selection criterion are set to zero. Also, there is no random value selection ($P_{\mathrm{rw}} = 0$). This way, by default, only the hard constraints are considered during the search. We will show how the other weights influence the search process and the overall solution quality.

7.1 Initial Problem

The experiments in Table 1 present the behaviour of the solver with respect to various settings of weights for particular criteria (the student conflicts, violated time preferences, and violated room preferences). It is important to see that the weights for particular criteria can be easily adjusted. This allows us to emphasize or suppress particular optimization criteria and it results in the corresponding change of the solution quality.

Time refers to the amount of time required by the solver to find the presented solution. *Satisfied enrolments* gives the percentage of satisfied requirements for courses chosen by students. *Preferred time* and *preferred room* correspond to



Large Lecture Room Timetabling v1.0 - Microsoft Internet Explorer

Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda | Adresa https://www.smas.purdue.edu/Tmtbl2004fal/index.jsp | Přejít

- ⊟ Purdue Timetable
  - ⊟ Input Configuration
    - ⊡ Buildings
    - ⊡ Rooms
    - ⊡ Instructors
    - ⊡ Classes
    - ⊡ Constraints
  - ⊡ Timetable
  - ⊟ Solver
    - ⊡ Bundled LS Solver
    - ⊡ Alone LS Solver
    - ⊡ Conf. Statistics
  - ⊟ Administration
    - ⊡ Users
    - ⊡ Versions
    - ⊡ Export Input Cfg
    - ⊡ Timings
    - ⊡ Debug
    - ⊡ Data Configuration

Login: muller
Name: Tomas Muller
Data: FALL 2004 v53 r/o
Server: v1.0 alpha build149
Fri, 11 Jun 2004

Timetable

EE 270	7:30a	8:30a	9:30a	10:30a	11:30a	12:30a	1:30p	2:30p	3:30p
Mon		MSE230 1001 0, 2, 0	ECET209 1001 0, 1, 0	AUSL227 1001 0, 2, 0	CSR331 1001 0, 0, 0	MSE230 2001 0, 2, 0	HIST104 4001 0, 0, 2	ECE321 1001 0, 2, 0	PHPR202 1(0, 0, 0
Tue	CPT385 1001 0, 1, 0	C E203 1001 0, 0, 0		PSY235 2001 0, 0, 0	HIST151 2001 0, 0, 0		ECE311 2001 0, 2, 0	ECE311 1001 0, 1, 0	
Wed	ECET214 1001 0, 0, 0	MSE230 1001 0, 2, 0	ECET209 1001 0, 1, 0	AUSL227 1001 0, 2, 0	CSR331 1001 0, 0, 0	MSE230 2001 0, 2, 0	HIST104 4001 0, 0, 2	ECE321 1001 0, 2, 0	PHPR202 1(0, 0, 0
Thu	CPT385 1001 0, 1, 0	C E203 1001 0, 0, 0		PSY235 2001 0, 0, 0	HIST151 2001 0, 0, 0		ECE311 2001 0, 2, 0	ECE311 1001 0, 1, 0	
Fri	ECET214 1001 0, 0, 0		ECET209 1001 0, 1, 0	AUSL227 1001 0, 2, 0	CSR331 1001 0, 0, 0		HIST104 4001 0, 0, 2	ECE321 1001 0, 2, 0	PHPR202 1(0, 0, 0

EE 170	7:30a	8:30a	9:30a	10:30a	11:30a	12:30a	1:30p	2:30p	3:30p
Mon	NUCL273 1001 0, 0, 0	NUCL200 1001 0, 8, 0	HTM181 1001 0, 0, 0	I E230 1001 0, 1, 0	C S352 1001 0, 0, 0	MA151 3001 0, 0, 0	ECE270 1001 0, 0, 0	PSY200 2001 0, 0, 0	NUC
Tue		EDPS235 1001 0, 0, 0	PHIL206 1001 0, 2, 2	EAS221 1001 0, 5, 0	HIST103 4001 0, 1, 0		F&N202 1001 0, 1, 0	ECE694A 1001 0, 0, 0	
Wed	NUCL273 1001 0, 0, 0	NUCL200 1001 0, 8, 0	HTM181 1001 0, 0, 0	I E230 1001 0, 1, 0	C S352 1001 0, 0, 0	MA151 3001 0, 0, 0	ECE270 1001 0, 0, 0	PSY200 2001 0, 0, 0	ECET196 1(0, 0, 0
Thu		EDPS235 1001 0, 0, 0	PHIL206 1001 0, 2, 2	EAS221 1001 0, 5, 0	HIST103 4001 0, 1, 0		F&N202 1001 0, 1, 0	ECE694A 1001 0, 0, 0	
Fri	NUCL273 1001 0, 0, 0	NUCL200 1001 0, 8, 0	HTM181 1001 0, 0, 0	I E230 1001 0, 1, 0	C S352 1001 0, 0, 0	MA151 3001 0, 0, 0	ECE270 1001 0, 0, 0	PSY200 2001 0, 0, 0	ECET196 1(0, 0, 0

EE 129	7:30a	8:30a	9:30a	10:30a	11:30a	12:30a	1:30p	2:30p	3:30p
Mon	CSR342 1001 10, 4, 0	MA161 1001 0, 0, 0	ENGR100 1001 0, 0, 0	PSY120 4001 0, 0, 0	MA161 2001 0, 0, 0	MA162 1001 0, 0, 0	PHIL330 1001 0, 2, 0	AGRY320 1001 5, 2, 2	MA162 200 0, 1, 0
Tue	AGEC217 3001 0, 0, 0		ENGR100 2001 0, 0, 0	HIST152 2001 0, 1, 0	ECON251 3001 0, 0, 0		PSY120 1001 0, 1, 2	PSY120 5001 3, 1, 1	
Wed	CSR342 1001 10, 4, 0	MA161 1001 0, 0, 0	ENGR100 3001 0, 1, 0	PSY120 4001 0, 0, 0	MA161 2001 0, 0, 0	MA162 1001 0, 0, 0	PHIL330 1001 0, 2, 0	AGRY320 1001 5, 2, 2	MA162 200 0, 1, 0

Internet

Fig. 2. Generated timetable at web-based graphical user interface

the satisfaction of time and room preferences respectively. 100% corresponds to a case when all classes are placed in their most preferred times or rooms, 0% means a case when the least preferred locations are used. Preferences of soft group constraints are not presented, since there are no such constraints in the Fall 2004 data set (all group constraints are either required or prohibited).

A complete solution was found on every run of all experiments in Table 1 except the experiment marked *No CBS*. Average values together with their RMS (root-mean-square) variances of the best achieved solutions from 10 different runs found within 30 minute time limit are presented.

The experiment marked *No preference* presents average solutions obtained without any preferences on the soft constraints. All solution quality weights W and value selection weights V are set to zero, except of the weight $V_{\text{wconf},1} = 1$ (weight of the weighted hard conflicts in the first level of the value selection).

The following three experiments marked *Students*, *Time* and *Rooms* are minimizing just one of the criteria: the student conflicts, violated time preferences, or violated room preferences. The *Students* experiment uses the same weights as the *No preference* experiment, but student weights are the following: $V_{\text{student},1} = 0.5$, $V_{\text{student},2} = W_{\text{student}} = 1$. Similarly, the *Time* experiment uses weights $V_{\text{time},1} = 0.5$, $V_{\text{time},2} = W_{\text{time}} = 1$ and the *Rooms* experiment weights $V_{\text{room},1} = 0.5$, $V_{\text{room},2} = W_{\text{room}} = 1$.

Table 1. Solutions of the initial problem

Test case	No preference	Students	Time	Rooms
Assigned variables [%]	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
Time [min]	0.16 ± 0.03	9.18 ± 4.47	18.79 ± 7.35	0.17 ± 0.01
Satisfied enrolments [%]	98.26 ± 0.15	$\mathbf{99.74 \pm 0.02}$	98.19 ± 0.13	98.18 ± 0.24
Preferred time [%]	62.54 ± 1.19	65.57 ± 1.53	$\mathbf{98.75 \pm 0.13}$	62.14 ± 0.94
Preferred room [%]	63.64 ± 2.29	62.96 ± 1.67	63.72 ± 1.64	$\mathbf{98.58 \pm 0.29}$

Test case	Students + Time	Students + Time + Rooms	No CBS
Assigned variables [%]	100.00 ± 0.00	100.00 ± 0.00	$\mathbf{98.42 \pm 0.20}$
Time [min]	19.96 ± 5.34	14.79 ± 4.87	24.08 ± 4.42
Satisfied enrolments [%]	$\mathbf{99.61 \pm 0.03}$	$\mathbf{99.79 \pm 0.01}$	99.52 ± 0.06
Preferred time [%]	$\mathbf{95.70 \pm 0.32}$	$\mathbf{95.02 \pm 0.37}$	94.62 ± 0.43
Preferred room [%]	62.68 ± 2.23	$\mathbf{75.30 \pm 2.30}$	83.77 ± 1.49

The experiment marked *Students + Time* equally combines student conflicts with time preferences, weights are $V_{\text{student},1} = V_{\text{time},1} = 0.25$, $V_{\text{student},2} = V_{\text{time},2} = W_{\text{student}} = W_{\text{time}} = 1$.

The next experiment (marked *Students + Time + Rooms*) most closely corresponds to reality. Here all the soft preferences are considered. Student conflicts and time preferences are weighted equally, room preferences are considered much less important. Weights of student conflicts and time preferences are the same as in the previous experiment (marked *Students + Time*). Moreover, the weights on room preferences are $V_{\text{room},2} = W_{\text{room}} = 0.2$. Note that rooms are not considered in the first level of the value selection criteria.

Finally, the last experiment (marked *No CBS*) presents average solutions obtained from the solver without conflict-based statistics. The weights on soft constraints are the same as in the previous experiment. But there is $V_{\text{conf},1} = 1$ (weight of a hard conflict) instead of $V_{\text{wconf},1} = 1$ (weight of a hard conflict weighted by CBS). $V_{\text{wconf},1}$ is set to zero. The solver was not able to find a complete solution within the given 30 minute time limit, not even when 2% random walk selection was used ($P_{\text{rw}} = 0.02$) to avoid cycling. Furthermore, there were at least five unassigned classes after three hours of running time.

Figure 3 compares several experiments giving different stress on student conflicts and time preferences. Average values from the best solutions of 10 different runs found within 30 minute time limit are presented.

Only student conflicts or time preferences are considered in the border experiments marked *students* and *time* respectively. In the middle (experiment marked *1:1*), student conflicts and time preferences are equally weighted. The experiment marked *3:1* prefers student conflicts three times as much as time preferences (i.e., weights of student conflicts are three times higher than weights of time prefer-

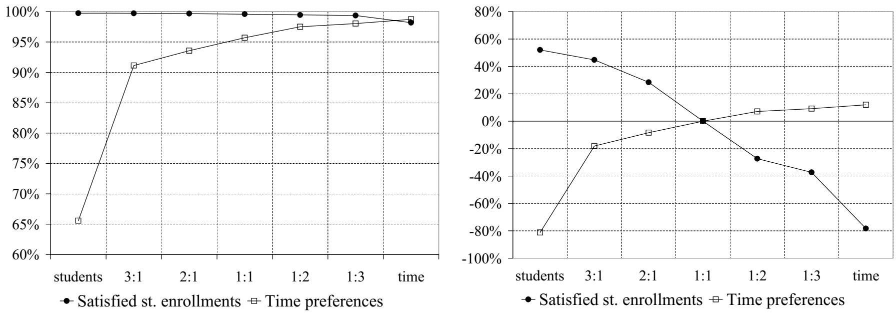


Fig. 3. Comparison of satisfied student enrolments and time preferences: average quality of the solution (left), improvement of the solution in terms of percentage of the 1:1 solution (right)

ences) and vice versa. For instance, the experiment marked *1:2* has the following weights: $V_{\mathrm{wconf},1} = 1$, $V_{\mathrm{student},1} = 0.2$, $V_{\mathrm{time},1} = 0.4$, $V_{\mathrm{student},2} = W_{\mathrm{student}} = 1$, $V_{\mathrm{time},2} = W_{\mathrm{time}} = 2$.

7.2 Minimal Perturbation Problem

The following experiments were conducted on one of the complete initial solutions computed in the previous set of experiments (column marked *Students + Time + Room* in Table 1). Input perturbations were generated such that a given number of randomly selected variables were not allowed to retain the values they were assigned in the initial solution. Therefore, these classes cannot be scheduled to the same placement as in the initial solution (either room or starting time must be different). Only variables with more than one value in their domains were used. For each number of input perturbations, 10 different sets of input perturbations (i.e., variables with initial values prohibited) were generated. The following figures show the average parameter values of the best solutions found within 10 minutes.

The aim of the first set of experiments is to find a suitable setting for P_{init} (probability of selection of an initial value) and $V_{\Delta\mathrm{init},1..3}$ (difference in the number of perturbations in value selection). In each experiment, we have executed 10 tests for each of $10, 20, 30, \ldots, 100$ input perturbations respectively (100 runs in total). The average number of assigned variables together with the average number of additional perturbations are presented in Table 2. One or a combination of the criteria is used in each experiment. The second column refers to the set of criteria described in Table 3.

Let us look at the explanation of this table. For instance, the expression $0.25_{s=1}$, $1.0_{s=2}$ in the column marked $V_{\mathrm{students,s}}$ means that $V_{\mathrm{students,1}}$ is set to 0.25 and $V_{\mathrm{students,2}}$ is set to 1. The first case ($\Delta\mathrm{init} = \mathbf{0}$) corresponds to the settings of the *Students + Time + Room* experiment. In remaining Δinit sets, we tried to decrease the importance of other value selection criteria in comparison

Table 2. Comparison of several approaches to MPP

Test case		Assigned	Number of
P_{init}	Δinit	variables [%]	perturbations
0.5	0	100.00	13.83
0.6	0	99.98	13.48
0.7	0	99.96	13.33
0.8	0	99.95	12.94
0	2	100.00	31.40
0.6	2	99.99	13.26
0	1	100.00	13.70
0.6	**1**	**100.00**	**11.90**

Table 3. Meaning of Δinit

Δinit	$V_{\Delta\text{init,i}}$	$V_{\text{student,s}}$	$V_{\text{time,t}}$	$V_{\text{room,r}}$
0	–	$0.25_{s=1}$, $1.0_{s=2}$	$0.25_{t=1}$, $1.0_{t=2}$	$0.2_{r=2}$
1	$0.5_{i=1}$	$1.0_{s=2}$	$1.0_{t=2}$	$0.2_{r=2}$
2	$1.0_{i=2}$	$0.25_{s=1}$, $1.0_{s=3}$	$0.25_{t=1}$, $1.0_{t=3}$	$0.2_{r=3}$

with the initial value delta. For $\Delta\text{init} = \mathbf{1}$, the first level value selection criterion $V_{\Delta\text{init},1}$ is used and the other optimization criteria which were placed in the first level are disabled ($V_{\text{student},1}$, $V_{\text{time},1}$ are set to zero). And the third line ($\Delta\text{init}=\mathbf{2}$) corresponds to a case when the second level value selection criterion $V_{\Delta\text{init},2}$ is used and the other optimization criteria from the second level ($V_{\text{student},2}$, $V_{\text{time},2}$, $V_{\text{room},2}$) are moved to the third level.

Let us discuss particular experiments from Table 2. In the first four experiments (marked $P_{\text{init}} = 0.5, \ldots, P_{\text{init}} = 0.8$), the minimal perturbation problem was solved only by changing the value selection criteria so that it selected the initial value with a given probability (50%, 60%, 70% and 80% respectively). Otherwise, it worked exactly as *Students + Time + Room* experiment, since all the other weights were the same. As the P_{init} probability is rising, we can see that the average number of additional perturbations is descending, but the algorithm is loosing the ability to find a complete solution in every run (in the given 10 minute time limit).

Similarly, we can see that using just the second level value selection criterion $V_{\Delta\text{init},2}$ is able to find a complete solution all the time, but the average number of additional perturbations is too high. A combination with the 60% probability of an initial value selection helps to improve the average number of additional

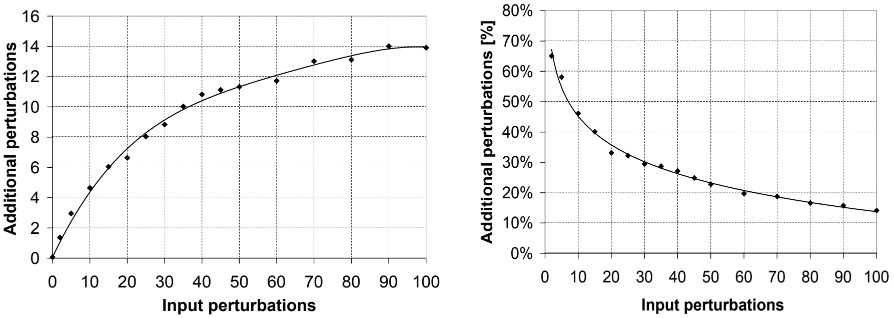


Fig. 4. Absolute number of average additional perturbations (left) and average additional perturbations in terms of percentage of the number of input perturbations (right)

perturbations, but again, there were some cases where a complete solution was not found.

Using the first level value selection criterion $V_{\Delta\mathrm{init},1}$ seems to be very promising. All the presented experiments with this criterion were able to find a complete solution. Moreover, the experiment marked $P_{\mathrm{init}} = 0.6$, $\Delta\mathrm{init} = 1$ (combining $V_{\Delta\mathrm{init},1}$ with 60% initial value selection probability) gave us the best results from the above experiments, since the average number of additional perturbations was the smallest. The following results (see Figures 4 and 5) were computed using the weights from this experiment.

Figure 4 presents the average number of additional perturbations (variables that were not assigned their initial values though not prohibited). Additional perturbations are presented wrt. the absolute number of input perturbation (i.e., up to about 13.4% of input perturbations is considered). The best solution found within 10 minutes from each experiment is taken into account. The number of additional perturbations grows with the number of input perturbations.

The graph on the left of Figure 5 shows the average quality of the resulting solutions in the same manner as presented in Table 1. Because the initial solution is (at least locally) optimal, and because the number of additional perturbations is the primary minimization criterion, it is not surprising that the quality of the solution declines with an increasing number of input perturbations. The weighting between time preferences, student conflicts, and other parameters considered in the optimization can have a similar influence as seen in the initial solutions.

Finally, the graph on the right of Figure 5 presents the average time needed to find the best solution. Note that a 10 minute time limit for finding a best solution was set. The influence of this limit is seen mostly on the right portion of the chart, where the number of input perturbations exceed 50.

Other Problems. Some MPP results of a preliminary version of the above described iterative forward search algorithm (e.g., without conflict-based statistics) on the *random placement problem* (see `http://www.fi.muni.cz/~hanka/rpp/`

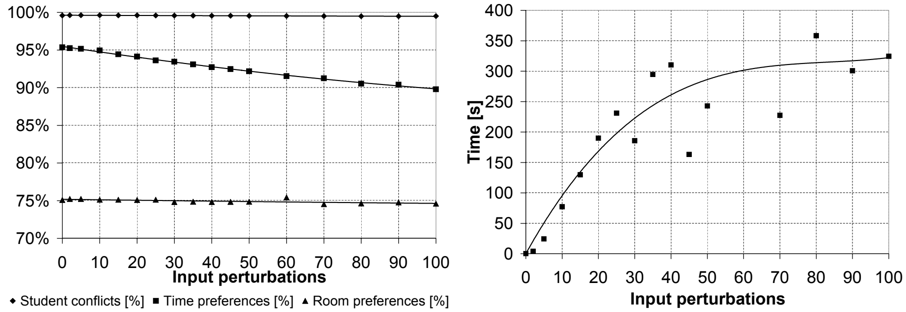


Fig. 5. Average solution quality (left), average time (right)

for details) are presented in our earlier work [1]. Here, it is compared with an algorithm combining branch-and-bound approach and the limited assignment number (LAN) search algorithm [2]. In this comparison, the iterative forward search algorithm was significantly better in its computational speed and in the number of additional perturbations than the other algorithm.

8 Conclusions

We have proposed and implemented a solution to a large-scale university timetabling problem. Our proposal includes a new iterative forward search algorithm that is extended by conflict-based statistics which we believe can be generalized to other search algorithms. Both ideas combined together suffice to solve the problem and the role of additional heuristics can be minimized. Our problem solver is able to construct a demand-driven timetable as well as incorporate dynamic aspects. The initial solution generated by our solver satisfies the course requests of more than 99% of students together with about 95% of time requirements. The automated search was able to find suitable times and classrooms for all classes. The experiments with a MPP give us very promising results as well. Within 10 minutes, the solver was able to find a complete, high-quality solution with a small number of additional perturbations.

Our future research will include extensions of the proposed general algorithm together with improvements to the implemented solver. We would like to do an extensive study of the proposed Minimal Perturbation Problem solver and its possible application to other, non-timetabling problems. We are also planning to compare our results with the previous CLP solver [15] we have implemented. We are currently extending the CLP solver with some of the features included here to present a fair comparison.

We are also working on extensions to the implemented solver to cover additional requirements and problem features required by Purdue University. The strategy for computing perturbations needs to be extended as well. For example, a change in time is usually much worse than a movement to a different classroom. The number of enrolled/involved students should also be taken into account. Another factor is whether the solution has already been published or not.

Acknowledgements

This work is partially supported by Purdue University, by the Czech Science Foundation under contract No. 201/04/1102 and by the Ministry of Education of the Czech Republic under research intent No. 0021622419. Our thanks also go to the Supercomputer Centre Brno where experiments with the search algorithms were conducted. We would like to thank our students for their assistance in solving this problem and Purdue staff who have helped in many ways. Our thanks also go to Keith Murray for his careful proofreading of the drafts of this paper.

References

1. Barták, R., Müller, T., Rudová, H.: A New Approach to Modeling and Solving Minimal Perturbation Problems. In: Recent Advances in Constraints. Lecture Notes in Artificial Intelligence, Vol. 3010. Springer, Berlin (2004) 233–249
2. Barták, R., Rudová, H.: Limited Assignments: A New Cutoff Strategy for Incomplete Depth-First Search. In: Applied Computing. ACM, New York (2005) 388–392
3. Cambazard, H., Demazeau, F., Jussien, N., David, P.: Interactively Solving School Timetabling Problems Using Extensions of Constraint Programming. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (2004) 107–124
4. Dechter, R.: *Constraint Processing.* Morgan Kaufmann, San Mateo, CA (2003)
5. Dechter, R., Frost, D.: Backjump-Based Backtracking for Constraint Satisfaction Problems. *Artif. Intell.*, **136** (2002) 147–188
6. Elkhyari, A., Guéret, C., Jussien, N.: Solving Dynamic Timetabling Problems as Dynamic Resource Constrained Project Scheduling Problems Using New Constraint Programming Tools. In: Burke, E. K., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV, Selected Revised Papers. Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 39–59
7. Guéret, C., Jussien, N., Boizumault, P., Prins, C.: Building University Timetables Using Constraint Logic Programming. In: Burke, E. K., Ross, P. (eds.): Practice and Theory of Automated Timetabling I. Lecture Notes in Computer Science, Vol. 1153. Springer, Berlin (1996) 130–145
8. Jussien, N., Lhomme, O.: Local Search with Constraint Propagation and Conflict-Based Heuristics. Artif. Intell. **139** (2002) 21–45
9. Kocjan, W.: Dynamic Scheduling: State of the Art Report. Technical Report T2002:28. SICS (2002)
10. Michalewicz, Z., Fogel, D. B.: *How to Solve It: Modern Heuristics.* Springer, Berlin (2000)
11. Miguel, I.: *Dynamic Flexible Constraint Satisfaction and its Application to AI Planning.* Springer, Berlin (2004)

12. Müller, T., Barták, R.: Interactive Timetabling: Concepts, Techniques, and Practical Results. In: Burke, E. K., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV, Selected Revised Papers. Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 58–72
13. Piechowiak, S., Ma, J., Mandiau. R.: EDT-2004: An Open Interactive Timetabling Tool. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (2004) 305–321
14. Ran, Y., Roos, N., van den Herik, J.: Approaches to Find a Near-Minimal Change Solution for Dynamic CSPs. In *Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems* (2002) 373–387
15. Rudová, H., Murray, K.: University Course Timetabling with Soft Constraints. In: Burke, E. K., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV, Selected Revised Papers. Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 310–328
16. El Sakkout, H., Wallace, M.: Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. Constraints **4** (2000) 359–388
17. Verfaillie, G., Jussien, N.: Dynamic Constraint Solving (2003). A tutorial including commented bibliography presented at CP 2003. See `http://www.emn.fr/x-info/jussien/CP03tutorial/`
18. Verfaillie, G., Jussien, N.: Constraint Solving in Uncertain and Dynamic Environments: A Survey. Constraints **10** (2005) 253–281

Feature Selection in a Fuzzy Student Sectioning Algorithm

Mahmood Amintoosi[1] and Javad Haddadnia[2]

[1] Mathematics Department, Sabzevar Teacher Training University, Iran, 397
[2] Engineering Department, Sabzevar Teacher Training University, Iran, 397
{amintoosi, haddadnia}@sttu.ac.ir

Abstract. In this paper a new student sectioning algorithm is proposed. In this method a fuzzy clustering, a fuzzy evaluator and a novel feature selection method is used. Each student has a feature vector, contains his taken courses as its feature elements. The best features are selected for sectioning based on removing those courses that the most or the fewest numbers of students have taken. The Fuzzy c-Means classifier classifies students. After that, a fuzzy function evaluates the produced clusters based on two criteria: balancing sections and students' schedules similarity within each section. These are used as linguistic variables in a fuzzy inference engine. The selected features determine the best students' sections. Simulation results show that improvement in sectioning performance is about 18% in comparison with considering all of the features, which not only reduces the feature vector elements but also increases the computing performance.

1 Introduction

The course timetabling problem essentially involves the assignment of weekly lectures to time periods and lecture room in such a way that a set of constraints satisfy. Current methods for tackling timetabling problems include evolutionary algorithms [14], [15], [17], [29], [30], genetic algorithms [12], [13], [18], [19], [20], graph-based methods [10], [11], [35], simulated annealing [2], [3], tabu search [24], [33], [34], neural networks [26], hybrid and heuristic approaches [6], [7], [28], [38], constraint logic programming [1], [9], [21], [23], [32], ant colony optimization [36] and fuzzy expert systems [5].

A particular problem related to timetabling is student sectioning. This problem is due to courses which involve a large number of students. For a variety of reasons, splitting these students into a few smaller sections is desirable: for example,

1. Room capacity requirements: when the number of students in a course is greater than every room capacity.
2. The policies of the institution: some institutions have rules about maximum capacity of courses (e.g. 50 for specialized courses and 60 for public courses).
3. A good student sectioning may reduce the number of edges in the conflict matrix [16].

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 147–160, 2005.

Most previous works related to the course scheduling problem have concentrated on timetable construction, with little regard to student sectioning.

Selim [35] introduced the idea of split vertices and made a start to determine those vertices, which should be split in order that the chromatic number may be reduced. Selim treated the problem as a conflict graph and showed how one could pick out certain vertices in the conflict graph to split, reducing the chromatic number of the graph to the desired value by increasing the number of sections in the timetable. With this idea Selim decreases the chromatic number of the conflict matrix, from 8 to 3. Thus the total number of periods needed is reduced.

The main algorithm of Laporte and Desroches [25] has the following stages:

1. Constructing student schedules without taking into account section enrollments and room capacities.
2. Balancing section enrollments.
3. Respecting room capacities.

One of its interesting features is the weight it gives to the overall quality of student schedules.

Aubin and Ferland [6] generate an initial timetable with an assignment of the students to the course sections; then an iterative procedure is used which adjusts the timetable and the grouping successively until no more improvement of the objective function can be obtained. At each iteration, two procedures are used:

1. Given the grouping generation during the preceding iteration, the timetable is modified to reduce the number of conflicts.
2. With this timetable, the grouping of students is modified to reduce the number of conflicts.

Hertz [24] used a tabu search technique for both timetabling and sectioning problems. He assumed that the numbers of students in each section are fixed. The neighborhood $N(s)$ of a solution s consists of all those grouping which can be obtained from s by exchanging the two students of two different sections of a course.

The initial student sectioning of Muller and Rudova [27] is based on Carter's [16] homogeneous sectioning and it is intended to minimize future student conflicts. They attempt to improve the solution with respect to the number of student conflicts. This is achieved via section changes during the search. Each student enrollment in a course with more than one section was processed. An attempt was made to switch it with a student enrollment from a different section. If this switch decreased the total number of student conflicts, it would be applied.

Amintoosi et al. [4] introduced a fuzzy sectioning method which decreases the average number of conflicts in a genetic timetabling program.

In this paper we concentrate on initial student sectioning prior to timetabling. A new student-sectioning algorithm is proposed. In the proposed method a fuzzy clustering, a fuzzy evaluator and a novel feature selection method are used.

A Fuzzy c-Means algorithm classifies students in a large class into smaller sections. Each student has a feature vector in fuzzy classifier. The courses taken by each student are its feature elements.

The produced clustering is evaluated with a fuzzy function, according to some criteria: size of clusters and students' schedules similarity of each section. The above parameters have been used in a fuzzy inference engine as linguistic variables for clustering evaluation.

By removing those courses that have been taken by *the most* or by *the fewest* numbers of students, the best features (courses) are selected. Appropriate values for *the most* and *the fewest* values are determined with an iteration procedure. In each iteration, courses which contain each of the following properties are removed:

- courses which have been taken by a high percentage of students, greater than a specified threshold, and
- those that have been taken by a low percentage of students, lower than another threshold.

A Fuzzy c-Means classifier classifies remaining courses. The produced clusters are evaluated by the mentioned fuzzy function. The best classification of students will be ready at the end of the above loop.

Simulation results in the average case show that about 53% of courses are essential for clustering and with these selected courses the clustering performance would be about 18% more efficient.

The reminder of this paper is organized as follows. Section 2 describes the fuzzy C-means clustering algorithm. In Section 3, the proposed method is explained in more detail and in Section 4 simulation results are considered. Section 5 concludes.

2 Fuzzy c-Means Clustering

Fuzzy c-Means (FCM) is a data clustering algorithm in which each data point is associated with a cluster through a membership degree. Most analytical fuzzy clustering approaches are derived from Bezdeck's FCM [8], [31]. This technique partitions a collection of N_T data points into r fuzzy groups and finds a cluster center in each group, such that a cost function of a dissimilarity measure is minimized [22]. The algorithm employs fuzzy partitioning such that a given data point can belong to several groups with a degree specified by membership grades between 0 and 1. A fuzzy r-partition of input feature vector $X = \{x_1, x_2, \ldots, x_{N_T}\} \subset \Re^n$ is represented by a matrix $\boldsymbol{U} = [\mu_{ik}]$, where the entries satisfy the following constraints:

$$\mu_{ik} \in [0,1]\,, \quad 1 \leq i \leq r\,, \quad 1 \leq k \leq N_T \tag{1}$$

$$\sum_{i=1}^{r} \mu_{ik} = 1\,, \quad 1 \leq k \leq N_T \tag{2}$$

$$0 < \sum_{k=1}^{N_T} \mu_{ik} < N_T\,, \quad 1 \leq i \leq r\,. \tag{3}$$

$\boldsymbol{U}$ can be used to describe the cluster structure of X by interpreting μ_{ik} as the degree of membership of X_k to cluster i. A proper partition $\boldsymbol{U}$ of X may be defined by the minimization of the following objective function:

$$J_m(U, C) = \sum_{k=1}^{N_T} \sum_{i=1}^{r} (\mu_{ik})^m d_{ik}^2 \tag{4}$$

where $m \in [1, +\infty]$ is weighting exponent called the fuzzifier, $C = \{c_1, c_2, \ldots, c_r\}$ is the vector of the cluster centres, and d_{ik} is the distance between X_k and the ith cluster. Bezdek proved that if $m \geq 1$, $d_{ik}^2 > 0$, $1 = i = r$, then $\boldsymbol{U}$ and C minimize $J_m(U, C)$ only if the entries of them are computed as follows:

$$\mu_{ik}^* = \frac{1}{\sum\limits_{j=1}^{r} (d_{ik}/d_{jk})^{\frac{2}{m-1}}} \tag{5}$$

$$c_i^* = \frac{\sum\limits_{k=1}^{N_T} (\mu_{ik})^m x_k}{\sum\limits_{k=1}^{N_T} (\mu_{ik})^m}\,. \tag{6}$$

One of the major factors that influences the determination of appropriate clusters of points is the dissimilarity measure chosen for the problem. Indeed, the computation of the membership degrees μ_{ik}^* depends on the definition of the distance measure d_{ik}, which is the inner product of norms (quadratic norms) on R^n. The squared quadratic norm (distance) between a pattern vector X_k and the center c_i of the ith cluster is defined as follows:

$$d_{ik}^2 = ||x_k - c_i||_G = (x_k - c_i)^T G (x_k - c_i) \tag{7}$$

where G is any positive definite $(n \times n)$ matrix. The identity matrix is the simplest and most popular choice of G.

The FCM algorithm consists of a series of iterations alternating between Equations (5) and (6). This algorithm converges to either a local minimum or a saddle point of $J_m(U, C)$. FCM is used to determine the cluster centers c_i and the membership matrix $\boldsymbol{U}$ for a given r value as follows:

Step1: Initially the membership matrix is constructed using random values between 0 and 1, such that constraints (1)–(3) are satisfied.

Step2: For each cluster i $(i = 1, 2, \ldots, r)$ the fuzzy cluster centres c_i are calculated using Equation (6).

Step3: For each cluster i, the distance measures d_{ik} are computed using Equation (7).

Step4: The cost function in Equation (4) is computed and if either it is found to be below a certain tolerance value, or its improvement over the previous iteration $(\mathrm{d}J_m)$ is below a certain threshold, then it is stopped and the clustering procedure is terminated.

Step5: A new $\boldsymbol{U}$ using Equation (5) is computed and steps 2–5 are repeated.

By using the above fuzzy clustering procedure, the students in large classes are divided into r clusters. Clustering evaluation is done with a fuzzy function. This fuzzy evaluation is based on a fuzzy inference engine. In the next section the proposed method is explained.

3 The Proposed Method

The aim is to allocate students of a course into smaller sections, satisfying the following criteria:

1. Student course selections must be respected.
2. Section enrollments should be balanced, i.e. all sections of the same course should have roughly the same number of students;
3. Section capacities and policies of institute should not be exceeded.
4. Student schedules in each section would be the same as each other (as much as possible).

A fuzzy c-Means algorithm is used for student sectioning [4]. This algorithm satisfies criterion 1. Other criteria are evaluated at the evaluation phase. A fuzzy inference engine evaluates the produced clustering. With a well-defined set of rules [4], the criteria 2–4 are considered.

Removing those courses that have been taken by the most or the fewest numbers of students achieved the best feature elements. The simulation results show that about 53% of features are important for classification; in addition, the clustering performance with selected features is better than the performance in the case that all features were considered.

The proposed algorithm contains three basic parts as follows:

- method of data representation,
- fuzzy clustering and fuzzy evaluation,
- feature selection method.

The following sections explain the basic parts of the proposed method.

3.1 Data Representation

In the proposed method each student has a feature vector. The courses taken by each student are its feature elements, represented by a bit array. Suppose that P is the number of all courses and V_i is the list of taken courses by student i. As shown in Equation (8), V_i is the feature vector of the ith object (student):

$$V_i = (V_{i_1}, \ldots, V_{i_P}) \ , \qquad V_{i_j} = \begin{cases} 1 & \text{if student } i \text{ has taken lesson } j \\ 0 & \text{otherwise} \, . \end{cases} \tag{8}$$

Table 1 shows an example for five students. Column 2 contains the selected courses list from three total courses, which are taken by each student. Column 3 shows the corresponding feature vector for each student. If sectioning of A is desirable, students 1 and 2 will be in Section 1, and others remain in the second section. Column 4 displays the sectioning results.

Table 1. List of courses taken by five students and their corresponding feature vectors

Student	Courses taken	Feature vector	Section no.
1	A, B	110	1
2	A, B	110	1
3	A, C	101	2
4	A, C	101	2
5	A, C	101	2

3.2 Fuzzy Clustering and Fuzzy Evaluation

In our algorithm Fuzzy C-Mean has been used as classifier. The input of the classifier is the students' feature vectors, as explained in the previous section. For simplicity, the number of clusters is assumed to be 2.

Clustering evaluation has been done with a fuzzy function. Rates of section balancing and similarity of students' schedules in each section (criteria 1 and 2) are its inputs and its output is the clustering performance. Two lingual variables "Density" and "N1PerN2" (N1/N2) are defined as inputs of a fuzzy inference engine. Density of clusters is the sum of the common courses of all student pairs (a, b) such that students a, b lie in the same section. By dividing the mentioned summation with its maximum value, the value of "Density" will be normalized."N1PerN2" represents the section's balancing rate. It is supposed that N1 is the size of the smaller section and hence, the range of this variable is between 0 and 1. Since the number of students in each section should be as equal as possible, the suitable values of N1PerN2 are close to 1. Figure 1 shows the membership functions of the mentioned variables.

The output of the fuzzy inference engine is named "Performance" and has the following values: Bad, NotBad, Medium, Good and Excellent. Our Fuzzy rules were defined as follows:

Fuzzy Rules

Rule 1:
if (Density is *High*) and (N1PerN2 is *Suitable*) then (Performance is Excellent)
Rule 2:
if (Density is *High*) and (N1PerN2 is *Middle*) then (Performance is Good)
Rule 3:
if (Density is *High*) and (N1PerN2 is *UnSuitable*) then (Performance is Bad)
Rule 4:
if (Density is *Med*) and (N1PerN2 is *Suitable*) then (Performance is Good)
Rule 5:
if (Density is *Med*) and (N1PerN2 is *Middle*) then (Performance is Medium)
Rule 6:
if (Density is *Med*) and (N1PerN2 is *UnSuitable*) then (Performance is Bad)

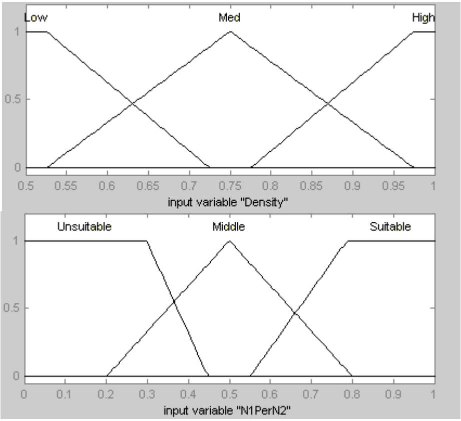


Fig. 1. Membership functions of our linguistic variables: "Density" (upper), "N1PerN2" (lower)

Rule 7:
if (Density is *Low*) and (N1PerN2 is *Suitable*) then (Performance is NotBad)
Rule 8:
if (Density is *Low*) and (N1PerN2 is *Middle*) then (Performance is Bad)
Rule 9:
if (Density is *Low*) and (N1PerN2 is *UnSuitable*) then (Performance is Bad)

The rules are defined such that the influence of unsuitable sections sizes is more than the effect of students' schedules similarity. Rules 3, 6 and 9 reflect this. Hence the decision surface of our rules' database shown in Figure 2 has an asymmetric face.

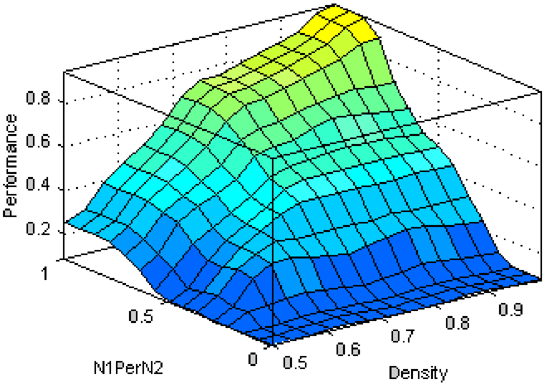


Fig. 2. Decision surface of our rules' database

3.3 Feature Selection Method

Feature selection plays an important role in classification problems. Advances in feature selection not only reduce the dimension of feature vector but also reduce the complexity of classifier. The most important rule in feature selection is reducing the feature elements as far as possible such that their class discrimination remains [37].

As you can see in Table 1, the first feature (course A) is common between all vectors (students). Removing it should not influence the clustering results. In our problem it seems that removing the following courses is a good idea:

1. courses that *the most* number of students have been taken;
2. courses that *the fewest* number of students have been taken.

Removing those courses that none of the students or all of them have taken is done in a pre-processing stage. One problem is to specify the appropriate threshold values for *the most* and *the fewest* parameters. An exhaustive search procedure finds them as follows: the percentage of students that have taken each course is determined and these values are finite. The number of such courses is equal to or less than the number of all courses. If P is the number of all courses taken by the students of the class, a procedure with worst time complexity $O(P^2)$ will find the appropriate values for *the most* and *the fewest*. Before entering a loop, the percentage of students that have taken each course, and the percentage that have not taken each course, are determined. Each value of these two lists can be a threshold for *the most* (T_1) and *the fewest* (T_2) parameters, respectively. In each iteration, those courses for which the percentage of students which have taken them is greater than T_1, or for which the percentage of students that have not taken them is greater than T_2, will be removed.

The fuzzy C-Means algorithm classifies students based on these selected courses (features). After that, the fuzzy function evaluates the produced clusters based on two criteria: balancing sections and students' schedules similarity of each section. After the last iteration the best classification of students will be arrived at. The following pseudo-code illustrates the overall procedure. In this algorithm *Clustering()* is a classifier function and *ClusteringEvaluation()* is an evaluator function.

```
AllCourses = Set of all courses that students of this
course have taken; //(All Features)
List1 = Percentage of students that have taken each
       course, sorted in non increasing order;
Thresholds1 = Distinct elements of List1;
List2 = Percentage of students that have not taken each
       course, sorted in non increasing order;
Thresholds2 = Distinct elements of List2;

for i: = 1 to length(Thresholds1) do
begin
```

```
        MaxSet = Set of Courses that percentage of students
                which have taken them > Thresholds1[i];
        for j: = 1 to length(Thresholds2) do
        begin
            MinSet = Set of Courses that percentage of students
                which have not taken them > Thresholds2[j];
            SelectedFeatures = AllCourses – (MaxSet $\cup$ MinSet);
            Clusters = Clustering(SelectedFeatures);
            if ClusteringEvaluation(Clusters) is better than
                previous clusters' performance then
            begin
                BestClusters = Clusters;
                $T_1$ = Thresholds1[i];
                $T_2$ = Thresholds2[j];
            end; // end if
        end; // end for j
    end; // end for i
```

4 Simulation Results

The information used for simulation is taken from the Mathematics department at Sabzevar University. Students and courses are randomly selected with the following characterization:

- total number of students: 210;
- number of courses: 38.

The simulator program has been written with Matlab on a Windows platform. Figure 3(a) shows the feature vectors for a course with 67 students and total 31 features. Figure 3(b) highlights the selected features. As can be seen in the last row of Table 2 (related to this course) the number of selected features (NSF column) is 16 out of 31. With these selected features the performance is better by about 35% (P1/P2 column) than with consideration of all features.

At each iteration of the feature selection algorithm, performance is evaluated with the selected features. The values of the fuzzy function's inputs are varying with every set of selected courses (features). Figure 4 demonstrates the inputs and output values of fuzzy evaluator function in a total of 179 iterations for the mentioned course with 67 students. The best solution is shown with a circle on iteration no 87 in Figures 4 and 5. The values of the "Density", "N1PerN2", "Performance" and the number of selected features are 0.73, 0.97, 0.75 and 16 respectively. The first iteration (values: 0.78, 0.60, 0.56 and 31) corresponds with the case in which all features have been considered in clustering. As can be seen, performance increases by about 35% ($0.75/0.56 = 1.35$).

The simulation results on 12 courses are shown in Table 2. As can be seen, the performance with the proposed method (P1) is always better than or equal to the performance when all features are considered (P2) in clustering. All of the features are required only in three examples (Examples 3, 10 and 11). In the average case, 53% of features are required and performance is increased by about 18%.

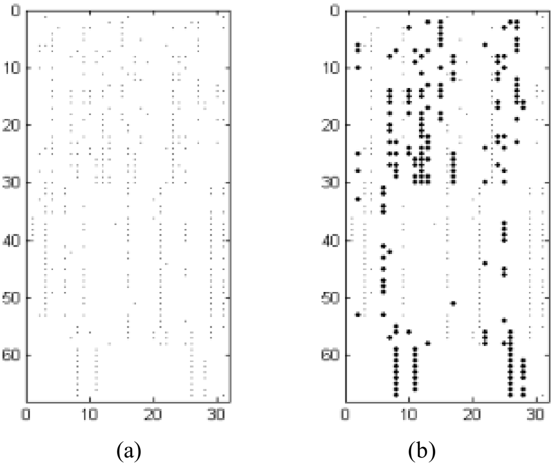


Fig. 3. (a) Feature vectors for a course with 67 students and 31 features (courses). (b) The selected features (columns in the figure) are highlighted.

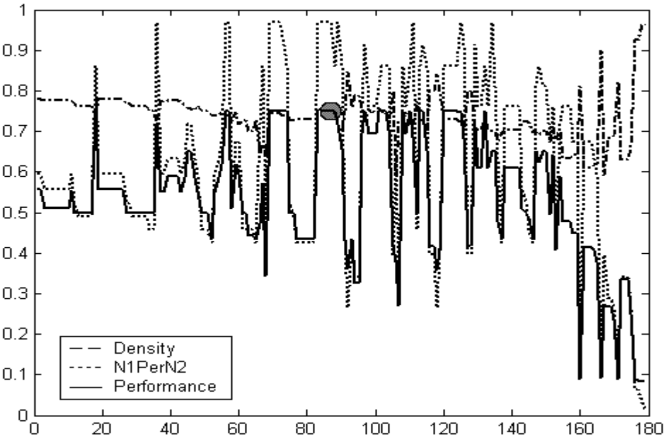


Fig. 4. The values of inputs and output of fuzzy evaluator function in the Feature Selection Algorithm. Dashed, dotted and solid lines represent the value of "Density", "N1PerN2" and "Performance", respectively. The best solution is shown with a circle on iteration no. 87.

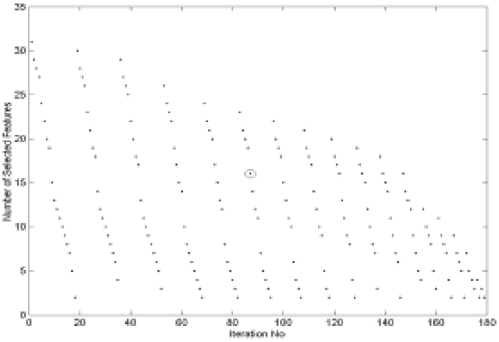


Fig. 5. The best number of selected features is 16 and demonstrated with a circle on iteration no. 87. The point formation depicts the outer and the inner loops in the proposed feature selection algorithm.

Table 2. The simulation results for 12 courses. P1 is the best performance achieved (with selected features), P2 is the performance taking into account all the courses. NSF is the best number of selected features, NTF is the number of total features, T_1 and T_2 are the values of *the most* and *the fewest* parameters.

LessonNo	N1PerN2	P1	P2	P1/P2	NSF	NTF	NSF/NTF	T_1	T_2
1	0.8	0.76	0.49	1.56	5	18	0.28	1	0.44
2	0.95	0.5	0.49	1.03	27	34	0.79	0.33	0.95
3	0.8	0.75	0.75	1	29	29	1	1	0.97
4	0.76	0.54	0.52	1.03	26	34	0.76	1	0.93
5	0.75	0.65	0.58	1.13	3	22	0.14	1	0.64
6	0.56	0.39	0.24	1.61	2	11	0.18	0.5	0.57
7	0.7	0.63	0.57	1.1	7	30	0.23	1	0.71
8	0.86	0.65	0.51	1.28	10	31	0.32	0.33	0.77
9	0.76	0.69	0.64	1.08	6	34	0.18	1	0.71
10	0.89	0.66	0.66	1	33	33	1	1	0.97
11	0.79	0.58	0.58	1	31	31	1	1	0.98
12	0.97	0.75	0.56	1.35	16	31	0.52	0.25	0.9
Average	0.80			1.18			0.53		

5 Concluding Remarks

In this paper a novel course selection method and a fuzzy evaluation function for the student sectioning problem is proposed. Our aim is to allocate students of a course to smaller sections—prior to timetabling—so as to satisfy the following criteria:

- student course selections must be respected,
- section enrollments should be balanced,

– section capacities and rules of institution should not be exceeded,
– student schedules in each section should be the same as each other (as far as possible).

Firstly, with a Fuzzy c-Means algorithm, students in a large class have been classified. Each student has a feature vector in the fuzzy classifier. The courses taken by each student are his/her features. In contrast to the usual graph-based sectioning, not only is the number of courses common to two students important, but also the number of courses that is not taken by both of them is significant. Secondly, the produced clusters are evaluated with a fuzzy function, according to two criteria: balancing sections and students' schedules similarity of each section. By removing those courses that the most students or the fewest students have taken, the best features (courses) are selected. Simulation results in the average case show that about 53% of courses are essential for clustering. The best classification of students corresponds with these selected features. Clustering performance with these selected features is increased by about 18% compared with considering all of the features.

References

1. Abdennadher, S., Marte, M.: University Course Timetabling Using Constraint Handling Rules. Appl. Artif. Intell. **14** (2000) 311–325
2. Abramson, D.: Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms. Manage. Sci. **37** (1991) 98–113
3. Abramson, D.: Simulated Annealing Cooling Schedules for the School Timetabling Problem. Asia-Pacific J. Oper. Res. **16** (1999) 1–22
4. Amintoosi, M., Sadooghi Yazdi, H., Haddadnia, J.: Fuzzy Student Sectioning. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 421–425
5. Asmuni, H., Burke, E. K., Garibaldi, J. M.: Fuzzy Multiple Ordering Criteria for Examination Timetabling. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 51–66
6. Aubin, J., Ferland, J. A.: A Large Scale Timetabling Problem. Comput. Oper. Res. **16** (1989) 67–77
7. Banks, D., Beek, P., Meisels, A.: Heuristic Incremental Modeling Approach to Course Timetabling. In: Proc. 12th Canadian Conf. on Artif. Intell. (1998) 16–29
8. Bezdek, J. C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum, New York (1981)
9. Boizumault, P., Gu'eret, C., Jussien, N.: Efficient Labeling and Constraint Relaxation for Solving Timetabling Problems. In: Proc. Workshop on Constraint Languages and Their Use in Problem Modeling. International Logic Programming Symposium (1994) 116–130
10. Burke, E. K., Elliman, D. G., Weare, R. F.: Extensions to a University Exam Timetabling System. In: Proc. IJCAI-93 Workshop on Knowledge-Based Production, Planning, Scheduling and Control (1993) 42–48
11. Burke, E. K. Elliman, D. G., Weare, R. F.: A University Timetabling System based on Graph Colouring and Constraint Manipulation. J. Res. Comput. Educ. **27** (1994) 1–18

12. Burke, E. K., Elliman, D. G., Weare, R. F.: The Automation of the Timetabling Process in Higher Education. J. Educ. Technol. Syst. **23** (1995) 257–266
13. Burke, E .K., Elliman, D. G., Weare, R. F.: Specialised Recombinative Operators for Timetabling Problems. In: Fogarty, T. C. (ed.): AISB Workshop on Evolutionary Computing. Lecture Notes in Computer Science, Vol. 993. Springer, Berlin (1995) 75–85
14. Burke, E. K., Newall, J. P.: A Multi-Stage Evolutionary Algorithm for the Timetable Problem. In: IEEE Trans. on Evolutionary Computation **3** (1999) 63–74
15. Burke, E. K., Newall, J. P., Weare, R. F.: Initialization Strategies and Diversity in Evolutionary Timetabling. Evol. Comput. **6** (1998) 81–103
16. Carter, W. M.: A comprehensive course timetabling and student scheduling system at the University of Waterloo. In: Burke, E. K., Erben, W. (eds): PATAT 2000—Proceedings of the 3rd International Conference on the Practice And Theory of Automated Timetabling (2000) 64–82
17. Corne, D., Ross, P., Fang, H.L.: Fast Practical Evolutionary Timetabling. In: Lecture Notes in Computer Science, Vol. 865. Springer, Berlin (1994) 250–263
18. Erben, W.: A Grouping Genetic Algorithm for Graph Colouring and Exam Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 397–421
19. Erben, W., Keppler, J.: A genetic algorithm solving a weekly course-timetabling problem. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 198–211
20. Fang, H.: Genetic Algorithms in Timetabling and Scheduling. Ph.D. Dissertation. Department of Artificial Intelligence, University of Edinburgh, (1994)
21. Goltz, H. J., Küchler, G., Matzke, D.: Constraint-Based Timetabling for Universities. In: Proc. INAP'98, 11th Int. Conf. on Applications of Prolog (1998) 75–80
22. Haddadnia, J., Ahmadi, M.: A Fuzzy Hybrid Learning Algorithm for Radial Basis Function Neural Network with Application in Human Face Recognition. Pattern Recognition **36** (2003) 1187–1202
23. Henz, M., Wurtz, J.: Constraint-Based Timetabling—A Case Study. Appl. Artif. Intell. **10** (1996) 439–453
24. Hertz, A.: Tabu Search for Large Scale Timetabling Problems. Eur. J. Oper. Res. **54** (1991) 39–47
25. Laporte, G., Desroches, S.: The Problem of Assigning Students to Course Sections in a Large Engineering School. Comput. Oper. Res. **13** (1986) 387–394
26. Mausser, H., Magazine, M. J., Moore, J. B.: Application of an Annealed Neural Network to a Timetabling Problem. INFORMS J. Comput. **8** (1996) 103–117
27. Muller, T., Rudova, H.: Minimal Perturbation Problem in Course Timetabling. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 283–304
28. Newall, J. P.: Hybrid Methods for Automated Timetabling, Ph.D. Dissertation. Department of Computer Science, University of Nottingham, UK (1999)
29. Paechter, B., Luchian, H., Petriuc, M.: Two Solutions to the General Timetable Problem Using Evolutionary Methods. In: Proc. 1st IEEE Conf. on Evolutionary Computation (1994) 300–305
30. Ross, P., Corne, D., Fang, H.L.: Successful Lecture Timetabling with Evolutionary Algorithms. In: Eiben, A. E., Manderick, B., Ruttkay, Z. (eds.): Applied Genetic and other Evolutionary Algorithms: Proc. ECAI'94 Workshop. Springer, Berlin (1995)

31. Rowski, B. L., Bezdek, J. C.: C-Means Clustering with the L_1 and L_∞ Norms. IEEE Trans. Syst., Man Cybernet. **21** (1991) 545–554
32. Rudova, H., Murray, K.: University Course Timetabling with Soft Constraints. In: Burke, E., De Causmaecker, P. (eds.): PATAT 2002—Proceedings 4th International Conference on the Practice and Theory of Automated Timetabling 73–89
33. Schaerf, A.: Tabu Search Techniques for Large High-School Timetabling Problems. Technical Report CS-R9611. CWI, Amsterdam, The Netherlands (1996)
34. Schaerf, A.: Local Search Techniques for Large High School Timetabling Problems. In: IEEE Trans. Syst., Man Cybernet. A **29** (1999) 367–377
35. Selim, S. M.: Split Vertices in Vertex Colouring and Their Application in Developing a Solution to the Faculty Timetable Problem. Comput. J. **31** (1988) 76–82
36. Socha, K., Knowles, J., Sampels, M.: A MAX-MIN Ant System for University Course Timetabling Problem. In: Ant Algorithms Third International Workshop. Springer, Berlin (2002) 1–13
37. Theodoridis, S., Koutroumbas, K.: Pattern Recognition. Academic, New York (1999)
38. Willemen, R.J.: School Timetable Construction, Algorithms and Complexity, Ph.D. Dissertation. Technische Universiteit Eindhoven (2002)

A Column Generation Scheme for Faculty Timetabling

Andrea Qualizza[1] and Paolo Serafini[1,2]

[1] Department of Mathematics and Computer Science,
University of Udine, Italy
[2] CISM, Udine, Italy

Abstract. In this paper we deal with the problem of building a timetable for the courses of a university faculty. We propose an integer linear programming approach based on column generation. Each column is associated with a weekly timetable of a single course. The constraints referring to classroom occupancy and the non-overlapping in time of courses are in the integer linear programming matrix. The constraints and preferences related to a single course timetable are embedded in the column generation procedure. Generating a column for a course amounts to selecting the currently best time slots in the week. The interaction between the column generation procedure and the branch-and-bound method is also discussed. Some computational results are shown.

1 Introduction

Building a timetable for a university faculty is a common task which has been carried out in many different ways. The reason for this diversity is perhaps that each faculty has its own peculiarities which suggest home-made timetabling methods. Although the core of a timetable is an assignment problem, there are always additional constraints which make the problem difficult and not suitable to a general purpose algorithm. Only some of the work done in designing faculty timetables has emerged into the literature. General characteristics of the problem can be found in the papers [10], [11].

The methods fall generally into two categories: heuristic methods and integer linear programming methods. The latter cannot be classified as exact methods because very rarely is the branch-and-bound tree fully explored due to the size of most problems. For a review of recent timetabling methods, especially heuristics, see [9].

In the integer linear programming formulation the problem is usually modeled with binary variables expressing the fact that a certain course has been assigned to certain time slots and to certain classrooms, and constraining the variables accordingly. This has been the approach taken by several authors, as in [6], [3].

However, the integrality relaxation of this model does not provide a strong bound and this has obvious drawbacks in terms of computing time. Besides, the need of introducing more realistic preferences over the set of variables associated with a single course as a whole, instead of just the preference sum for each time

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 161–173, 2005.

slot, cannot be practically done. This makes it sometimes very difficult to "drive" a solution to a better one.

Therefore this approach is sometimes coupled with heuristic techniques which on the one hand reduce the complexity of the model and on the other hand allow the user to friendly interact with the procedure to obtain acceptable solutions. See for instance the SAPHIR system described in [4] and based on [1], where the authors also faced a problem feature which is not required in our model, namely the grouping of students in different course sections. The rigidity of the syllabi in Italian universities, with very few elective courses, results in a straightforward grouping task.

In this paper we propose an original approach which is essentially based on an integer linear programming formulation, but, instead of the usual assignment binary variables, we use binary variables for each weekly course timetable. Due to the exponential number of different course timetables the formulation requires a column generation scheme. The use of column generation procedures is in general recommended because part of the combinatorial structure of the problem is already embedded in the constraint matrix, thus providing stronger bounds (see the next section). The power of column generation techniques has been always recognized. See for instance [2], [7]. Recently, in [8] this approach was experimented (real data were used to test the model) for the timetable problem in high schools, which however has a quite different structure to that in academic faculties.

The problem we face consists of courses, classrooms and time slots. Courses must be assigned to both classrooms and time slots by respecting constraints of non-simultaneous use of the same classrooms in the same time slot and non-overlapping in time of certain groups of lectures. These are the main constraints we have to take care of. There are preferences on the time slots due to teaching reasons and lecturers' preferences as well. The lecturers may also express their preferences on the whole set of time slots. However, these preferences cannot be fully arbitrary and must be known in advance for them to be taken care of during the column generation phase.

It is convenient to decompose the problem by first considering classroom types instead of single classrooms, with the idea that classrooms of the same type are interchangeable. The main integer linear programming model considers classroom types. Once a timetable is computed, courses are easily assigned to single classrooms.

In our main model a column is associated with a weekly timetable of a single course. The matrix rows take care of the constraints referring to classroom occupancy and non-overlapping in time of some courses.

It turns out that generating a column for a course can be computed very quickly because it amounts to picking the best numbers in an array. Once a fractional solution of the integrality relaxation is found we compute an integral solution by resorting to an integrality solver on the generated columns. This may already provide a good solution.

If we want to improve the solution (or if we need to find a solution at all because there was none with the generated columns) we have to start a branch-and-price procedure. However, the column generation scheme conflicts with the branch-and-bound method. There are some subtle issues connected to this point. We are confronted with an NP-hard problem and we show how dynamic programming techniques can be used to design a pseudopolynomial algorithm to generate columns under the additional requirements that some variables are fixed to zero.

The approach proposed in this paper has two main advantages: the first one derives from the combinatorial properties of the model and, as already remarked, results in a better bound for the branch-and-bound procedure; the second one consists in the possibility of better "controlling" the structure of the weekly schedule, as we shall see later.

The main features of the model are discussed in Sections 2–4. A small example is provided in Section 5 to better show how columns are generated. The issues related to the branch-and-price procedure are described in Section 6. In Section 7 we briefly describe the computational results obtained by applying the method to the real data of our faculty. Finally, we show in Section 8 how courses are assigned to classrooms.

2 The Integer Linear Programming Model

We assume that the classrooms can be partitioned into sets of classrooms of the same type. Classrooms of the same type are interchangeable, i.e. a course can be assigned to any classroom of a certain type. Hence it is simpler to first assign a course to a classroom type and later, via a simple assignment problem, assign the course to a specific classroom. Therefore in this phase we only consider classroom types. We address the problem of assigning courses to classrooms in Section 8. Let K be the set of classroom types and let n_k be the number of classrooms of type $k \in K$.

Let C be the set of courses. Usually a course is ideally suited to a certain classroom type. However, it may be convenient, in case of unavailability, to assign it to a classroom of a different type, if available and feasible. For instance a course with few students should be placed in a small classroom, but it may also be assigned to a large classroom, although this is less preferred in general. Let $K(c)$ be the set of feasible classroom types for course c.

The timetable we consider is weekly. The week is partitioned in a set of time slots. The time slots do not necessarily have the same duration, although equal time slots (e.g. two hours or one hour each) are preferable in the column generation phase. In this paper we only consider equal time slots. Let H be the set of time slots. Let $d(c)$ be the required number of time slots for course c.

A typical feature of faculty timetabling is that certain courses must not be taught in the same time slot. The most obvious case concerns courses taught by the same person. Besides, there are always courses which should be attended by the same group of students and therefore must be scheduled in different times.

Let us define as $C_q \subset C$, $q \in Q$, the sets of non-overlapping courses (with Q an abstract index set). Conversely, let $Q(c) := \{q \in Q : c \in C_q\}$, i.e. the list of non-overlapping groups to which c belongs.

Let $P(c)$ be the set of timetable patterns for the course c. By "pattern" we mean an assignment of all the required hours per week for the course to definite time slots and definite classroom types. The number of possible patterns for each course is exponential, but we will generate only a subset of patterns. In order to constrain the patterns we need to define the following matrices:

$$a_{(kh)(jc)} = \begin{cases} 1 & \text{if course } c \text{ is assigned the time slot } h \text{ in a classroom of type } k \\ & \text{for the pattern } j \in P(c) \\ 0 & \text{otherwise} \end{cases}$$

$$a'_{(h)(jc)} = \begin{cases} 1 & \text{if course } c \text{ is assigned the time slot } h \\ & \text{for the pattern } j \in P(c) \\ 0 & \text{otherwise}\,. \end{cases}$$

Clearly $a'_{(h)(jc)} = 0$ if and only if $a_{(kh)(jc)} = 0$ for each $k \in K$ and

$$\sum_{h \in H} a'_{(h)(jc)} = \sum_{h \in H} \sum_{k \in K(c)} a_{(kh)(jc)} = d(c) \qquad j \in P(c), \quad c \in C\,.$$

We introduce the following variables:

$$x_{jc} = \begin{cases} 1 & \text{if pattern } j \in P(c) \text{ is used for course } c \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The constraints are as follows: we require that

$$\sum_{c \in C} \sum_{j \in P(c)} a_{(kh)(jc)}\, x_{jc} \le n_k \qquad k \in K, \quad h \in H \tag{2}$$

to avoid simultaneous use of more than n_k classrooms of type k. We require that

$$\sum_{c \in C_q} \sum_{j \in P(c)} a'_{(h)(jc)}\, x_{jc} \le 1 \qquad h \in H, \quad q \in Q \tag{3}$$

to impose non-overlapping of courses in the same group q. We require that

$$\sum_{j \in P(c)} x_{jc} = 1 \qquad c \in C \tag{4}$$

to impose that a course is assigned to exactly one pattern. The number of rows of the problem is given by $|H| \cdot |K| + |H| \cdot |Q| + |C|$, which can be large but not intractable. We consider as objective function the maximization of a preference

$$\sum_{c \in C} \sum_{j \in P(c)} r_{jc}\, x_{jc} \tag{5}$$

where

$$r_{jc} = \sum_{h \in H} \sum_{k \in K(c)} a_{(kh)(jc)} \, s_{khc}$$

and s_{khc} is the preference of using the time slot h and the classroom type k for the course c.

Therefore the integer linear programming problem consists in the maximization of (5) subject to (1)–(4). The approach to solve it is via branch-and-bound with column generation (i.e. branch-and-price). The integrality constraint is relaxed to $x_{jc} \geq 0$ and the patterns (i.e. the columns) are generated until optimality of the integrality relaxation is reached. At this point, unless the solution is integer, the branch-and-price procedure starts.

If we compare this model to the usual model in which binary variables are associated to course-time slots assignments (assignment model) we may observe that each feasible solution of the integrality relaxation of (1)–(4) can be easily turned into a feasible solution of the integrality relaxation of the assignment model, whereas the converse is not true in general (if for instance the assignment variables have different values for the same course in different time slots). Therefore, the integrality relaxation of the model of this paper yields a better bound than the relaxation of the assignment model.

3 Column Generation

Let us define the dual variables w_{kh}, v_{hq}, u_c for the constraints (2)–(4) respectively. Then the dual constraints are

$$\begin{aligned} &\sum_{h \in H} \sum_{k \in K} a_{(kh)(jc)} \, w_{kh} + \sum_{h \in H} \sum_{q \in Q(c)} a'_{(h)(jc)} \, v_{hq} + u_c \\ &\quad \geq \sum_{h \in H} \sum_{k \in K} a_{(kh)(jc)} \, s_{khc} \, , \qquad j \in P(c) \, , \quad c \in C \, , \end{aligned}$$

i.e.

$$\sum_{h \in H} \Big(\sum_{k \in K} a_{(kh)(jc)} \, (w_{kh} - s_{khc}) + \sum_{q \in Q(c)} a'_{(h)(jc)} \, v_{hq} \Big) + u_c \geq 0, \quad j \in P(c), \, c \in C \, .$$

So, in order to generate a pattern j for the course c, we have to minimize, with respect to a and a',

$$\sum_{h \in H} \Big(\sum_{k \in K} a_{(kh)(jc)} \, (w_{kh} - s_{khc}) + \sum_{q \in Q(c)} a'_{(h)(jc)} \, v_{hq} \Big) \, . \qquad (6)$$

Let us define

$$\hat{w}_{hc} = \min_{k \in K(c)} w_{kh} - s_{khc}$$

$$\hat{v}_{hc} := \sum_{q \in Q(c)} v_{hq}$$

and

$$t_{hc} := \hat{w}_{hc} + \hat{v}_{hc} \, .$$

Then minimizing (6) is equivalent to minimizing, for each c,

$$\sum_{h \in H} t_{hc} \, a'_{(h)(jc)} \, . \tag{7}$$

Minimizing (7) can be subject to some constraints or preferences related to the particular course. Let us consider some relevant cases.

The simplest case is the one without constraints, i.e. any set of $d(c)$ time slots is feasible for course c. In the sequel, to ease the notation, we drop the dependence of d on c. In this case minimizing (7) can be done by selecting the d minimum values of t_{hc}, $h \in H$. This computation can be carried out with complexity $\mathrm{O}(|H| \log d)$. It is clear that we deal with fixed and generally small values of $|H|$ and d and consequently, it is seems out of place to provide asymptotic bounds. Yet it is useful to realize that the algorithm for a column generation is a fast one.

Quite often we are not allowed to assign more than one time slot per day to a course. In this case the minimization of (7) is carried out by selecting the d minimum values of t_{hc} on different days. This is simply carried out by taking the best values of t_{hc} for each day, let us denote them by $\hat{t}_1$, $\hat{t}_2$, $\hat{t}_3$, $\hat{t}_4$, $\hat{t}_5$, and then selecting the d best values out of them. Here we need just to scan all $|H|$ values and then to scan at most twice the $\hat{t}_i$ values.

Sometimes teachers prefer to teach on consecutive days without any particular preference for the actual days. Then the pattern is generated by considering the best value among $(\hat{t}_1 + \hat{t}_2 + \hat{t}_3)$, $(\hat{t}_2 + \hat{t}_3 + \hat{t}_4)$, $(\hat{t}_3 + \hat{t}_4 + \hat{t}_5)$ (if for instance $d = 3$).

Another preference expressed sometimes by teachers consists in having all classes either all in the morning or all in the afternoon. This can be easily carried out by considering the d best values of t_{hc} of the morning hours and the d best values of t_{hc} of the afternoon hours and by selecting the better solution.

Finally, in case the time slots do not consist of two consecutive hours (as is anyway advisable in general) but of single hours, one typical requirements is that hours for the same course come out in pairs as much as possible. This type of constraint is particularly nasty in the usual formulation. But here it does not make the pattern generation too much harder. Suppose we have to allocate five hours and four of them must be in pairs. Furthermore, let us assume that no more than two hours per day can be taught. Then we compute the best values $\hat{t}_1$, $\hat{t}_2$, $\hat{t}_3$, $\hat{t}_4$, $\hat{t}_5$ as before. We also compute all values $t_{hc} + t_{h'c}$ with h and h' consecutive hours and we take the best of these values for each day. Let $\tilde{t}_1$, $\tilde{t}_2$, $\tilde{t}_3$, $\tilde{t}_4$, $\tilde{t}_5$ be these values. Now we have to select three different indices i, j, k such that $\tilde{t}_i + \tilde{t}_j + \hat{t}_k$ is maximum. Although implementing this computation is not straightforward, the computation itself is quick.

Let M_c be the minimum obtained for the course c (no matter the particular rule to generate the pattern for that course). If $M_c + u_c \geq 0$ the optimality condition is satisfied for the course c whereas if $M_c + u_c < 0$ the pattern obtained by minimizing (7) has to be inserted into the matrix.

4 Initialization

Since finding a feasible timetable is by itself NP-hard, we cannot initialize the matrix with a set of feasible patterns. It is more convenient to introduce artificial variables z_c to the equality constraints (4), which become

$$\sum_{j \in P(c)} x_{jc} + z_c = 1\,, \quad c \in C \tag{8}$$

and the original objective function is replaced by

$$\min_{c \in C} z_c\,, \quad \text{i.e. } \max_{c \in C} -z_c\,.$$

The only difference in the column generation procedure is that the values s are zero. The initial solution is taken as $z_c = 1$ and $x_{jc} = 0$. Due to the null value of x_{jc}, any pattern can be used to fill up the matrix initially. However, we may even think of starting without any pattern at all and generate all of them. Indeed, no matter which are the initial patterns, the initial values for the dual variables are $w_{hk} = 0$, $v_{hq} = 0$, $u_c = -1$, and therefore the first generated patterns can be any. In order to speed up the computation it is advisable to use as objective function a weighted sum of the original objective and the artificial one.

We recall that we solve the relaxed problem and therefore the initial solution can be fractional. In other words we may find an initial fractional feasible solution even if there is no feasible integer solution.

5 An Example

We limit ourselves to show one single column generation, because everything else is standard. The example is a small instance for illustration purposes. Let us suppose that there are four time slots ($H = 4$) and five courses c_1, c_2, c_3, c_4, c_5 with $d(c_1) = 2$, $d(c_2) = 2$, $d(c_3) = 1$, $d(c_4) = 1$, $d(c_5) = 2$. The courses are grouped into two non-overlapping sets $C_1 = \{c_1, c_2\}$ and $C_2 = \{c_3, c_4, c_5\}$. There are two classroom types ($K = \{k_1, k_2\}$) and two classrooms for each type, i.e. $n(k_1) = n(k_2) = 2$. The relationship between courses and classrooms is defined as $K(c_1) = \{k_1\}$, $K(c_2) = K(c_5) = \{k_2\}$, $K(c_3) = K(c_4) = \{k_1, k_2\}$. We assume that the ideal classroom type for courses c_3 and c_4 is k_1. They can fit type k_2 but in this case their preferences (columns 3 and 4 of table below) are reset to 0. The preferences are

$$s_{hc} = \begin{pmatrix} 0 & 2 & 1 & 2 & 2 \\ 2 & 0 & 1 & 0 & 2 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 1 & 2 & 1 & 0 \end{pmatrix}\,.$$

Let us suppose that three columns have been generated for each course, so that the matrix A (constraints (2)) is (the columns in boldface correspond to the current solution)

$$\begin{array}{r} k_1\left\{\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right. \\ k_2\left\{\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right. \end{array}
\begin{pmatrix}
1&0&0&0&0&0&1&0&0&0&\mathbf{1}&0&0&0&0\\
0&1&\mathbf{1}&0&0&0&0&0&0&1&0&0&0&0&0\\
1&0&\mathbf{1}&0&0&0&0&0&1&0&0&1&0&0&0\\
0&1&0&0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0\\
0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&1&0&0&0&0&0&0&0&1&\mathbf{1}\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&\mathbf{1}\\
0&0&0&1&\mathbf{1}&0&0&0&0&0&0&0&1&0&0
\end{pmatrix}.$$

The matrix A' (constraints (3)) is

$$\begin{array}{r} C_1\left\{\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right. \\ C_2\left\{\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right. \end{array}
\begin{pmatrix}
1&0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0&0&0\\
0&1&\mathbf{1}&1&0&1&0&0&0&0&0&0&0&0&0\\
1&0&\mathbf{1}&0&0&0&0&0&1&0&0&1&0&0&0\\
0&1&0&1&\mathbf{1}&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&\mathbf{1}&0&0&1&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&1&\mathbf{1}\\
0&0&0&0&0&0&0&0&1&0&0&1&1&0&\mathbf{1}\\
0&0&0&0&0&0&0&\mathbf{1}&0&0&0&0&1&0&0
\end{pmatrix}$$

and the assignment matrix (constraints (4)) is

$$\begin{pmatrix}
1&1&\mathbf{1}&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&\mathbf{1}&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&\mathbf{1}&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&\mathbf{1}&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&\mathbf{1}
\end{pmatrix}.$$

The objective function coefficients are

$$(\,1\;4\;\mathbf{3}\;1\;\mathbf{3}\;1\;1\;\mathbf{2}\;1\;0\;\mathbf{2}\;1\;1\;4\;\mathbf{3}\,)$$

so that the current solution value is 13. The computed dual variables at this stage are

$$w = 0\,, \quad v_{h1} = (0 \;\; 2 \;\; 0 \;\; 1)\,, \quad v_{h2} = (1 \;\; 0 \;\; 0 \;\; 0)\,, \quad u_c = (1 \;\; 2 \;\; 2 \;\; 1 \;\; 3)\,.$$

So we compute the following values for the course c_1:

$$\hat{w}_{h1} = \min_{k \in K(c_1)} w_{kh} - s_{khc} = w_{1h} - s_{1hc} = (0 \quad -2 \quad -1 \quad -2)\,,$$

$$\hat{v}_{h1} := v_{h1} = (0 \;\; 2 \;\; 0 \;\; 1)$$

and

$$t_{h1} := \hat{w}_{h1} + \hat{v}_{h1} = (0 \;\; 0 \quad -1 \quad -1)\,.$$

We have to allocate two time slots for course c_1 so that the minimum is in selecting time slots 3 and 4 with minimum value $M_1 = -2$. Since $M_1 + u_1 = -1 < 0$, the optimality condition is not satisfied for c_1 and the pattern $(0, 0, 1, 1)$ has to be generated.

For the course c_3 we have

$$\hat{w}_{h3} = \min_{k \in K(c_3)} w_{kh} - s_{kh3} = \min\{w_{1h} - s_{h3}\,;\, w_{2h}\} = -s_{h3} = (-1 \ \ -1 \ \ 0 \ \ -2)$$

$$\hat{v}_{h3} := v_{h2} = (1 \ \ 0 \ \ 0 \ \ 0)$$

and

$$t_{h3} := \hat{w}_{h3} + \hat{v}_{h3} = (0 \ \ -1 \ \ 0 \ \ -2)\,.$$

In this case we have to allocate only one time slot and the best way to do it is to allocate the fourth time slot. So $M_3 = -2$. Since $M_3 + u_3 = 0$ the optimality condition is satisfied and there is no need to generate columns for c_3. We omit the similar computations for the courses c_2, c_4 and c_5.

6 Branch-and-Price Strategy

We first solve the relaxed problem by generating columns until optimality is reached. If we end up with a fractional solution, we invoke a ILP routine on the generated columns to get a first incumbent. Then we start a branch-and-bound search. The branching is done by setting the fractional variables to 0 and to 1. These additional constraints however conflict with the column generation scheme. While fixing a variable to 1 poses no problem, there are problems in fixing a variable to 0. Indeed there is no way to prevent generating again columns whose corresponding variables are forced to 0.

We circumvent the problem as follows. Let us suppose that we are solving a subproblem in the branch-and-bound tree for which $(K - 1)$ variables have been set to 0. If we compute the first K minima in (7), we are sure that among those minima there is the minimum of (7) with the additional requirement of excluding the $(K - 1)$ columns associated to the variables set to 0.

However, computing the first K minima in (7) is not a straightforward problem. Let us consider the case when (7) is minimized without constraints. There is an array of values $T := \{t_1, t_2, \ldots, t_{|H|}\}$ (in general unrestricted in sign). Let $J \subset \{1, 2, \ldots, |H|\}$. The value of the subset J is defined as $\sum_{j \in J} t_j$. We ask if there are at least K distinct subsets of $\{1, 2, \ldots, |H|\}$ with value at most $-u_c$. This is a variant of the K-th LARGEST SUBSET problem, which is known to be NP-hard (see for instance [5]). Our problem is a variant with subsets of equal cardinality and values unrestricted in sign. It is not hard to see that the variant is NP-hard as well. However, it may be solved pseudopolynomially, for instance in the following way. Let $T_j := \{t_1, t_2, \ldots, t_j\}$ and $I_j := \{1, 2, \ldots, j\}$. Define $L(i, j)$ to be a list of the values of the K best subsets of I_j with i elements (the list may have less than K values if K subsets do not exist) and $L^*(i, j)$ to be the list of the corresponding subsets. Then the following dynamic programming recursion holds:

$$L(i,j) = \min\{L(i,j-1) \; ; \; t_j + L(i-1,j-1)\} \tag{9}$$

where $t_j + L(i-1,j-1)$ means that the value t_j is added to each value of the list $L(i-1,j-1)$ and the "min" operation is actually a merge operation extracting the best K values from the two lists. The same merge operation is carried out on $L^*(i,j-1)$ and $L^*(i-1,j-1) \cup j$. This operation has complexity $\mathrm{O}(K)$ on sorted lists and produces a sorted list. The meaning of (9) is that $L^*(i,j)$ is obtained by merging the lists which do not contain j and those which do. By the optimality principle the latter are optimal if the subsets with one element less up to $j-1$ are optimal.

The recursion is initialized as

$$L(0,j) = \{0\}, \quad L^*(0,j) = \{\emptyset\}, \quad j := 0, \ldots, |H|$$

and is computed for increasing values of i and j (note that $L(i,j)$ and $L^*(i,j)$ are undefined for $i > j$). The complexity is $\mathrm{O}(K\,|H|\,d)$. In our case due to the small values of d, H and K this computation is quite fast.

The other cases of minimizing (7) can be taken care of in similar ways. For instance if we consider the case of allowing at most one time slot per day, then we denote the time slots as (jk) (jth time slot of day k) and define $L(i,k)$ to be a list of the values of the K optimal subsets up to the day k with i elements and $L^*(i,k)$ to be the lists of the same subsets. Then the recursion is as follows:

$$\begin{aligned} L(i,k) = \min\{L(i,k-1)\,, t_{(1k)} + L(i-1,k-1))\,, \\ t_{(2k)} + L(i-1,k-1) \ldots t_{(pk)} + L(i-1,k-1)\} \end{aligned}$$

(p is the number of time slots per day) with complexity $\mathrm{O}(K\,|H|\,d)$ as before ($L^*(i,j)$ is computed accordingly).

7 Computational Results

We have applied the model to the actual data of one teaching period of our faculty. There are 63 courses, 25 time slots (five days, five time slots per day), four classroom types and 25 groups of non-overlapping courses. This can be considered a medium size model. The starting LP model has 788 rows and 168 columns. We use CPLEX routines to solve the model.

Solving the LP relaxation requires 731 columns to be generated. The fractional optimum has a value of 1443. At this point the CPLEX MIP routine is called to solve the ILP problem with the currently generated columns. The MIP routine returns the first incumbent with value 1411. The computation time up to this point is 225 seconds, of which 173 seconds are spent on the MIP routine.

As expected, the LP relaxation provides a strong upper bound (it is a maximization problem). The gap is $(1443 - 1411)/1443 = 0.02217$, i.e. around 2%.

Since the objective function is an artificial one, in the sense that the preferences are numbers which reflect in an imprecise way the real preferences of teachers and students, we might consider that any solution within a certain gap is acceptable. This has the obvious implications that the branch-and-bound tree does not grow too much. Indeed if we consider acceptable a gap within 3% we might just take the first incumbent and stop the computation.

If, in contrast, we continue the computation to the very end, we need to build a branch-and-bound tree with 556 nodes and depth 53 and generate 534 more columns and we eventually find an integer solution of value 1443. This shows that the gap provided by the relaxed model is actually zero for this instance. This is quite remarkable. It raises, however, the need of finding a better incumbent to prune more efficiently the branch-and-bound tree and also to look for better strategies to explore the tree in order to find the optimum as soon as possible. This is the subject of future research.

We also point out that it is advisable to implement a derived model in which constraint violations are allowed at the price of high penalty values in the objective function. This does not introduce any new feature in the column generation scheme. If there is no feasible solution (which unfortunately may happen) then the artificial variables associated with the penalty values do not vanish and it is possible to detect which constraints are responsible for the infeasibility, thereby suggesting new requirements on the timetable.

Once an initial solution is available this can be modified manually by the user either directly or by fixing part of the timetable and running the model again. We are currently experimenting with this successive phase of timetable building.

8 Assigning Courses to Classrooms

Once an integer solution is found to the main problem we have a complete assignment for each time slot of each course to a certain classroom type. We know that the constraint on the number of available classrooms for each type is satisfied by the solution. It is therefore a trivial task to assign a definite classroom for each course if we do not ask for more. In other words, as long as we consider each time slot independent of the others, we may simply assign in a greedy way the classrooms for each time slot. However, we usually would like to have all lectures of the same course in the same classrooms. Although this is not a compulsive requirement, it is a desired property of a timetable. Sometimes, it is more than a soft requirement. If there are many students in a class, it may be very annoying and time consuming having them moving around between lectures.

So it is reasonable to require that certain lectures stay in the same classroom as much as possible. Let the solution of the main model be represented by a set of triples (c, h, k). Each triple states that course c is taught in time slot h in classroom of type k. Clearly the problem we face is decomposed into classroom types, so we may just consider pairs (c, h), taking for granted the classroom type.

Let Z be the set pairs (c, h). We partition Z into sets $Z_1, Z_2, \ldots, Z_f$ with the idea that pairs in Z_i should be in the same classroom as much as possible. The partition can be specified manually by taking into account for instance courses with the same groups of students.

Let R be the set of classrooms. Let us define variables

$$x_{chr} := \begin{cases} 1 & \text{if the pair } (c,h) \text{ is assigned to classroom } r \\ 0 & \text{otherwise} \end{cases}$$

and variables

$$y_{jr} := \begin{cases} 1 & \text{if a pair in } Z_j \text{ is assigned to classroom } r \\ 0 & \text{otherwise} . \end{cases}$$

Then we may write the following constraints:

$$\sum_r x_{chr} = 1 \qquad (c,h) \in Z$$

stating that each pair must be assigned to a classroom,

$$\sum_{c:(ch) \in Z} x_{chr} \leq 1 \qquad r \in R, \quad h \in H$$

stating that for each time slot and each classroom there can be at most one course, and

$$x_{chr} \leq y_{jr} \qquad r \in R, \quad (ch) \in Z_j$$

to set y variables consistently with x variables. Then we minimize the number of classroom changes within each set Z_j by minimizing either $\sum_{jr} y_{jr}$ or $\max_{jr} y_{jr}$. Due to the symmetry of the problem and also for practical reasons it is advisable to introduce preferences q_{jr} for the assignments between sets Z_j and classroom r so that we actually minimize $\sum_{jr} q_{jr} \, y_{jr}$.

References

1. Aubin, J., Ferland, J. A.: A Large Scale Timetabling Problem. Comput. Oper. Res. **16** (1989) 67–77
2. Barnhart C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P ., Vance, P.: Branch-and-Price: Column Generation for Solving Huge Integer Problems. Oper. Res. **46** (1998) 316–329
3. Daskalaki, S., Birbas, T., Housos, E.: An Integer Programming Formulation for a Case Study in University Timetabling. Eur. J. Oper. Res. **153** (2004) 117–135
4. Ferland, J. A., Fleurent, C.: SAPHIR: A Decision Support System for Course Scheduling. Interfaces **24** (1994) 105–115
5. Garey, M. R., Johnson, D. S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)

6. Hultberg, T. H., Cardoso, D. M.: The Teacher Assignment Problem: A Special Case of the Fixed Charge Transportation Problem. Eur. J. Oper. Res. **101** (1997) 463–473
7. Maculan N., de Mendonça Passini, M., de Moura Brito, J. A., Loiseau, I.: Column-Generation in Integer Linear Programming. RAIRO Oper. Res. **37** (2003) 67–83
8. Papoutsis K., Valouxis, C., Housos, E.: A Column Generation Approach for the Timetabling Problem of Greek High Schools. J. Oper. Res. Soc. **54** (2003) 230–238
9. Schaerf, A.: A Survey of Automated Timetabling. Artif. Intell. Rev. **13** (1999) 87–127
10. de Werra, D.: An Introduction to Timetabling. Eur. J. Oper. Res. **19** (1985) 151–162
11. de Werra, D.: The Combinatorics of Timetabling. Eur. J. Oper. Res. **96** (1997) 504–513

School Timetabling

Decomposition and Parallelization of Multi-resource Timetabling Problems

Petr Šlechta

Czech Technical University in Prague, Department of Cybernetics,
Technická 2, 166 72 – Prague 6, Czech Republic
pslechta@ra.rockwell.com

Abstract. This paper presents a model of generalized timetabling problem and a decomposition algorithm which can decompose large problems into independent smaller subproblems—the search for a feasible solution can then be easily parallelized. The timetabling problem consists in fixing a sequence of meetings between teachers and students in a prefixed period of time (typically a week), satisfying a set of constraints of various types [9]. Course timetabling is a multi-dimensional NP-complete problem [4]. In this paper we present the multi-resource timetabling problem (MRTP), our model for the generalized high-school timetabling problem. The MRTP is a search problem—a feasible solution is searched. The main contribution of this paper is the Decomposition Algorithm for MRTP to parallelize the search, so the whole MRTP can be easily distributed and parallelized. Our approach was applied to real-life instances of high-school timetabling problems. Results are discussed at the end of this paper and parallelized search is compared with the centralized one.

1 Introduction

The traditional high-school timetabling problem is defined as follows [9], [12] (the terminology from the citations is used in this section; Section 2 defines the terminology which is used in the rest of this paper):

We have m classes $c_1, \ldots, c_m$, n teachers $t_1, \ldots, t_n$, and p periods $1, \ldots, p$. We are also given a non-negative integer matrix R_{mxn}, called requirements matrix, where r_{ij} is the number of lectures given by teacher t_j to class c_i. The problem consists in assigning lectures to periods in such a way that no teacher or class is involved in more than one lecture at a time.

This definition does not reflect the following requirements [9]:

1. some lectures require special rooms (e.g. music education lecture requires a music room);
2. some lectures may be given to more than one class (e.g. gymnastic lesson may involve two classes together);
3. some constraints on timetables may be defined for teachers, classes, and rooms (e.g. some teacher may be unavailable at some time).

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 177–189, 2005.

All the requirements described above are generalized and considered in our multi-resource timetabling problem (MRTP) model, which is described in Section 2.

It is well known that timetabling problems can be particularly difficult to solve, especially when dealing with particularly large instances [2]. Academic timetabling problems generally exhibit a specific structural property: courses tend to be grouped into natural clusters of dense interaction patterns. In high schools, each grade within the academic or vocational streams produces a cluster. In universities, clusters tend to form around required courses within each program and year. The number of edges between clusters is relatively sparse [3].

Our motivation was to find an algorithm which can split large problems into smaller (preferably independent) instances that can be handled by conventional algorithms (for example by backtracking). We believe that a lot of timetabling problems have "hidden" internal structure that can be followed when the problem is decomposed.

Several decomposition techniques can be found in the literature [3], [2], [7], [1]. Some of the techniques split a large problem into smaller ones, which can be easily handled, but must be solved in a serial way. (Problem P is split into P1 and P2, P1 is solved, then P2 is solved. P2 depends on P1 because assignments done for P1 influence assignments that can be applied to P2.) Such approaches can be found in [3] and [2]. Another possibility is to fix some of the variables to decrease the complexity of the problem (see [7] for details). Another way is to use autonomous agents forming coalitions to split the search space and to parallelize the search (see [1]).

We have developed a decomposition algorithm that splits the whole problem into independent components which can be solved in a parallel way (possible assignments for subproblem P1 do not influence possible assignments for P2 and vice versa). MRTP is transformed into a non-oriented graph, color marking is applied to the graph, and finally the decomposition algorithm discovers how to decompose and parallelize the search. The algorithm is described in Section 3.2.

Our approach was applied to real-life instances. An example of search parallelization is given in Section 3.5. In Section 4 we discuss some results and we compare parallelized search with the centralized one.

2 The Multi-resource Timetabling Problem Model

In the MRTP model we treat all teachers, classes, and rooms as *resources*. Each resource has defined some constraints on its timetable.

All lessons are treated as *events*. Each event requires some number of resources (e.g. a lesson of mathematics requires one teacher and one class, a gymnastic lesson may involve four resources: one teacher, two classes, and one special room).

Hence, the MRTP can be described by a set of resources and a set of events.

Definition 1 (MRTP). An instance of the MRTP is described by the tuple $INPUT = \langle E, R \rangle$, where $R = \{r_1, \ldots, r_m\}$ is a set of m resources and $E = \{e_1, \ldots, e_n\}$ is a set of n events to be scheduled.

Definition 2 (Resource). Each resource r_i is described by the tuple $\langle S_i, C_i \rangle$, where S_i is a timetable associated with the resource r_i, and C_i are constraints for the resource r_i.

Definition 3 (Event). Each event e_i is described by the tuple $\langle S_i, C_i, R_i \rangle$, where S_i is a timetable associated with the event e_i, C_i are constraints for the event e_i, and $R_i = \{r_1, \ldots, r_{m_i}\}$ is a set of m_i resources required by the event e_i.

To schedule an event, all required resources must be properly scheduled to the same time slot with respect to all event constraints and all constraints of the required resources. Each event requires at least one resource (it may require more than one resource) for its completion. At one time, each resource can be occupied by one event at most. Event assignment is described more formally in the following definition.

Definition 4 (Event assignment). Each event $e_i = \langle S_i, C_i, R_i \rangle$, for which $S_i = [s_{i1}, \ldots, s_{iL}]$, $C_i = [c_{i1}, \ldots, c_{iL}]$, $R_i = \{r_{i1}, \ldots, r_{i,mi}\}$, and $r_i = \langle S_j, C_j \rangle$, L is length of the schedule, can be scheduled to the slot x if all of the following conditions are satisfied:

1. The slot x of event's timetable is not occupied and is not forbidden by any event's constraint.
2. All required resources have free slot x in their timetables.
3. None of the constraints of all required resources restricts us to use the slot x.

Our goal is to schedule properly all events E from the input tuple *INPUT*. The following definition describes what we mean by feasible solution of MRTP.

Definition 5 (Solution of MRTP). A solution of Multi-Resource Timetabling Problem can be described by the set $OUTPUT = \{S_1, \ldots, S_m\}$, where S_i is a timetable for resource r_i from the tuple *INPUT*. All events from the *INPUT* tuple must be scheduled according to Definition 4.

2.1 Example of MRTP

In this section we use the MRTP model from the previous section to describe a simple high school timetabling problem. The problem is very simple and is only used to illustrate how the model can be used:

$$\begin{aligned} INPUT = \langle E, R \rangle\,, \quad R &= \{\mathrm{A, B, C, John, Bill, Ray, Joe}\}\,, \\ E &= \{\mathrm{M1, M2, F1, F2, H1, H2, H3, A1, A2, Ph1}\}\,. \end{aligned}$$

A, B, and C are classes (disjunctive groups of students); John, Bill, Ray, and Joe are teachers. M1, M2, F1, F2, H1, H2, H3, A1, A2, and Ph1 are subjects which should be scheduled. For simplicity we do not introduce any constraints on timetables in this example. Also the timetables are simplified: each timetable consists of three days, each day has three time slots.

Detailed descriptions of all events are summarized in Table 1.

Table 1. Description of all events from the problem defined in this section

Event name	Event description	No of instances	Required resources
M1	Mathematics	1	A, John
M2	Mathematics	2	B, Joe
F1	Physics	2	A, John
F2	Physics	2	B, Joe
H1	History	1	A, Ray
H2	History	1	B, Bill
H3	History	1	C, Ray
A1	Art	1	C, Ray
Ph1	Philosophy	1	C, Bill
A2	Art	1	C, Bill

Table 2. One possible solution of the problem defined in this section: timetables for the classes A, B, and C. Timetables for the teachers can be easily derived.

sA

H1	F1	
F1		
M1	H1	

sB

M2		
M2		
H2	F2	

sC

A2		
A1	Ph1	
H3		

$$OUTPUT = \{ \text{sA, sB, sC, sJohn, sBill, sRay, sJoe} \}\,.$$

One possible solution to the problem described above is shown in Table 2. We present only timetables for the classes A, B, and C. Timetables for the teachers (sJohn, sBill, sRay, and sJoe) can be easily derived.

3 Solution Approach

To parallelize the search we have developed a Decomposition Algorithm (DA) which transforms the MRTP into a non-oriented graph, applies color marking to the graph, and discovers how to decompose and parallelize the search. Subproblems that are small enough (have acceptable complexity) are solved by backtracking.

The decomposition algorithm is described in the following listing:

```
function solveMRTP(MRTP $p$) {
  $solutions$ := {};
  $p$ := sortEvents($p$);
  $g$ := MRTP2graph($p$);
  for $i$ := 1 to $noOfDecompositionsToTry$ {
    $edges$ := findEdgesToRemove($g$, $i$);
```

```
      (p1, p2) := satisfyEvents(edges);
      if hasGoodComplexity(p1)
        sol1 := solveMRTP(p1);
      else
        sol1 := solveLocally(p1);
      if hasGoodComplexity(p2)
        sol2 := solveMRTP(p2);
      else
        sol2 := solveLocally(p2);
      sol := integratePartialSolutions(sol1, sol2);
      solutions := solutions + {sol};
    }
    bestSolution := findBestSolution(solutions);
    return bestSolution;
  }
```

3.1 MRTP as Non-oriented Graph

The MRTP is transformed to a non-oriented graph as follows:

1. Each resource is represented as one graph node.
2. Each event is transformed into a set of edges. The set of edges forms complete subgraph under all nodes representing resources required by the event to its completion.

Figure 1 shows a graph for one event which requires four resources for its completion (such event does not appear in our example). The example from Section 2.1 transformed into non-oriented graph is shown in Figure 2.

3.2 Decomposition

Parallelization of the MRTP is based on splitting of the whole problem into independent subproblems. Splitting of the problem into independent subproblems corresponds to splitting the graph into isolated subgraphs (the subgraphs are not connected by any edge).

In most cases we have to remove some edges from the graph to split it. There are usually more sets of edges that can be removed from the graph to split it. Our goal is to select the proper set of edges which is small enough and which splits the graph into two subgraphs of approximately same size. We have developed a color marking algorithm (CMA) which marks the graph with two colors and discovers the proper set of edges which should be removed from the graph. The CMA is described in Section 3.3.

Once we have found the proper set of edges to be removed, we can start with decomposition of the problem. The removed edges correspond to events which must be scheduled to split the MRPT into two subproblems which are independent. These two subproblems may be solved in parallel and the found solutions may be easily integrated.

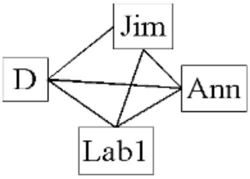


Fig. 1. Graphical representation of one event which requires four resources for its completion (teachers Jim and Ann do teach together class D in room Lab1)

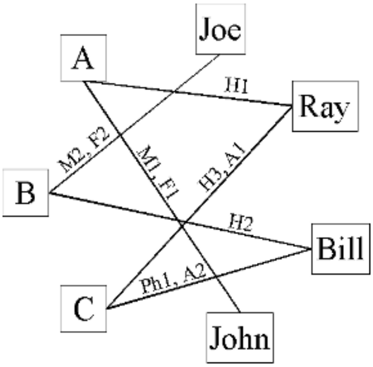


Fig. 2. Graphical representation of the problem from Section 2.1

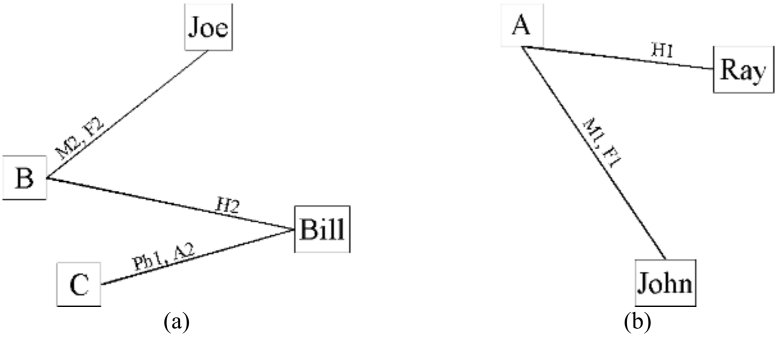


Fig. 3. The graph from Figure 2 divided into two isolated subgraphs

The selected event must be scheduled carefully (with respect to emerging subproblems) otherwise the subproblems will be restricted too much (inappropriate placement of the event reduces too much the set of all possible assignments that can be applied to the subproblem).

The graphical representation of example from Section 2.1 is in Figure 2. If we remove the edge "H3, A1" the graph is split into two independent subgraphs, see Figure 3.

If we schedule events H3 and A1, the remaining events are split into two independent sets:

$$G_1 = \{\mathrm{M2}, \mathrm{F2}, \mathrm{H2}, \mathrm{Ph1}, \mathrm{A2}\}, \quad G_2 = \{\mathrm{M1}, \mathrm{F1}, \mathrm{H1}\}\,.$$

All resources are also split into two sets:

$$G_{1r} = \{\mathrm{B}, \mathrm{C}, \mathrm{Bill}, \mathrm{Joe}\}, \quad G_{2r} = \{\mathrm{A}, \mathrm{John}, \mathrm{Ray}\}\,.$$

By scheduling an event from set G_1 only resources from set G_{1r} may be affected. The same holds for sets G_2 and G_{2r}. The problem was split into two independent subproblems.

3.3 Color Marking Algorithm

This section describes the CMA designed to discover the set of edges which should be removed to split the whole graph into isolated subgraphs.

The CMA is based on the following idea: We remove one edge from the graph. One node, to which the removed edge was connected, will obtain red color and the second one will obtain blue color. Then the color is propagated through the edges and the color loses its intensity. When the color propagation is finished, each node has a portion of blue and red color. The node is finally colored by the color of which the node has bigger portion. At the end, red nodes form the first subgraph and blue nodes form the second one (see Figure 4).

If the graph has z edges we start the CMA z-times to obtain all possible color markings of the graph. From all these color markings we select the most appropriate based on its rating. The rating is composed of the two components: (a) number of edges which were removed from the graph (smaller number is preferred), and (b) difference between sizes of two subgraphs (the size of both subgraphs should be similar). When an edge is removed from the graph, the corresponding event has to be scheduled. This decision is made locally—a partial schedule is formed.

The CMA is briefly described in the following listing:

```
for each node $n$ in the graph {
    $n$.red_portion = 0;
    $n$.blue_portion = 0;
}
for each edge $e$ in the graph {
    remove $e$ from the graph;
    $node1$ = the 1^{st} node to which $e$ was connected;
    $node1$.red_portion = 100;
    $node1$.blue_portion = 0;
    $node2$ = the 2^{nd} node to which $e$ was connected;
    node2.red_portion = 0;
    node2.blue_portion = 100;
    repeat {
      changed = false;
      for each node $n$ in graph {
        call routine propagate_color($n$);
      }
    } until (changed == false);
    count a rating for the colored graph;
  }
  select the partitioning with the best rating;

  routine propagate_color(node $n$) {
    no_of_edges = number of all edges connected to the node $n$;
    red_portion_to_propagate = $n$.red_portion / no_of_edges;
    blue_portion_to_propagate = $n$.blue_portion / no_of_edges;
    for each edge $e$ connected to the node $n$ {
      neighbor_node = node to which the selected edge $e$ is connected
      if (red_portion_to_propagate > neighbor_node.red_potion) {
        neighbor_node.red_portion = red_portion_to_propagate;
        changed = true;

      }
      if (blue_portion_to_propagate > neighbor_node.blue_potion) {
        neighbor_node.blue_portion = blue_portion_to_propagate;
        changed = true;
      }
    }
  }
```

3.4 Examples of Graph Decomposition

All possible decompositions of the problem from Section 2.1 are illustrated in Figure 5. Decompositions of some other problems are shown in Figure 6. These problems are not described in this paper and the graphs are only to illustrate the decomposition of larger problems.

3.5 Example of Search Parallelization

The complete decomposition of the problem from Section 2.1 is shown in Figure 7. At the beginning, we start with all events in one set. This set is received by the first scheduler, which schedules events H3 and A1 locally. After this the problem splits into two independent subproblems (events M2, F2, H2, Ph1, A2 represent the first subproblem, and events M1, F1 and H1 form the second subproblem). Then each subproblem is decomposed recursively as shown in Figure 7.

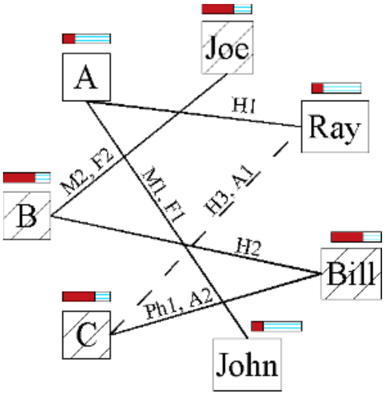


Fig. 4. The colored graph from Figure 2 (assigned portion of red and blue color is above each node). The edge which must be removed to split the graph into two subgraphs is dashed.

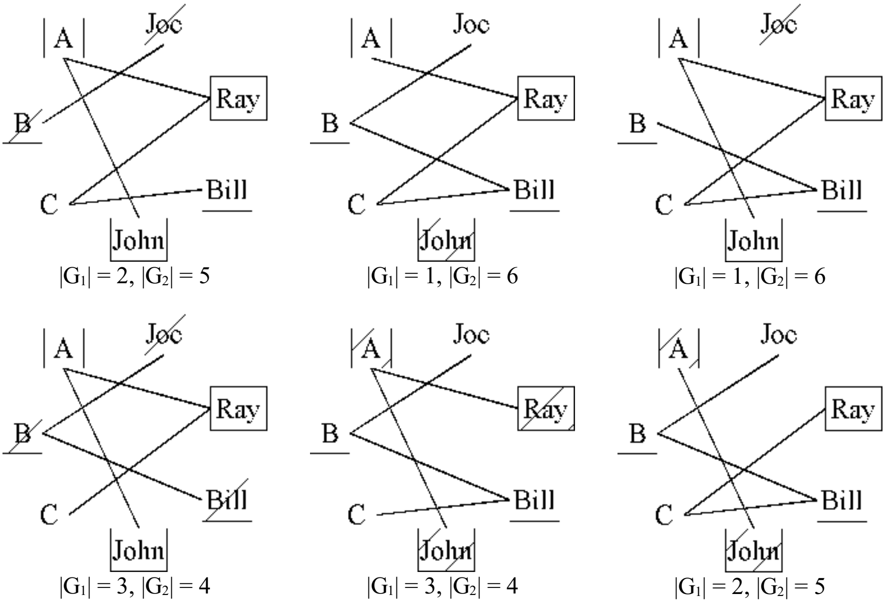
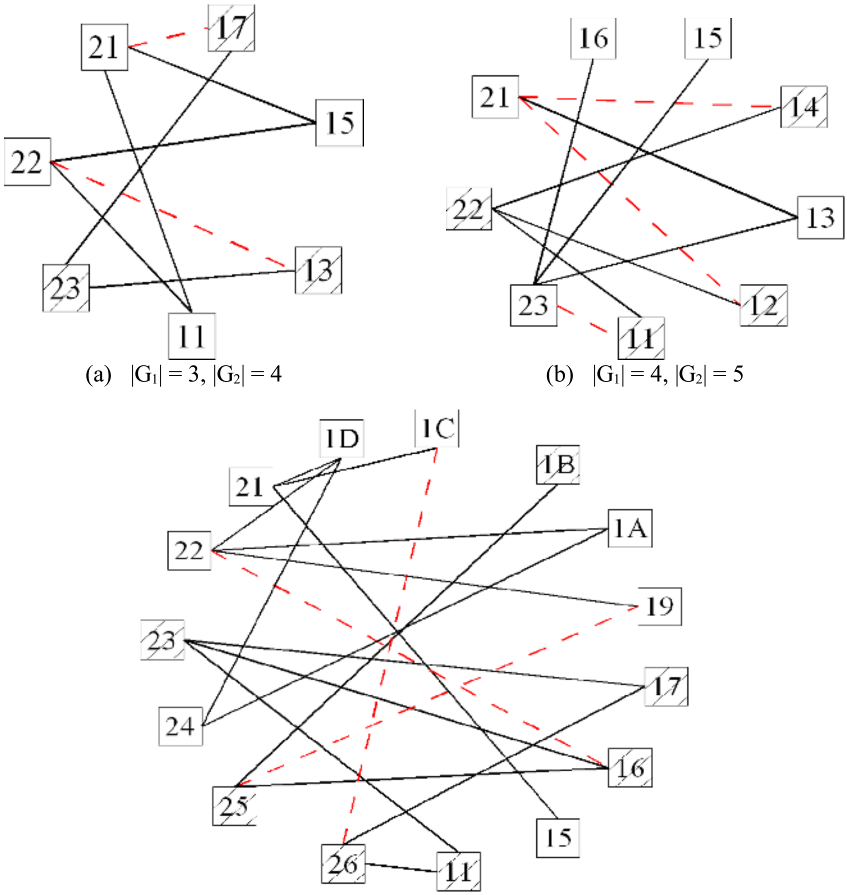


Fig. 5. Possible decompositions of the graph from Figure 2. Hatched nodes form the first subgraph, nodes without hatching form the second subgraph.

The parallelization of the search is shown in Figure 8. The problem is split into two subproblems: one was dedicated to the machine M2 and the second one was dedicated to the machine M4. The machine M2 (resp. M4) decomposed the given subproblem into two new subproblems: one was solved by machine M2 (resp. M4) itself, the second one was dedicated to the machine M3 (resp. M5).



(a) $|G_1| = 3, |G_2| = 4$

(b) $|G_1| = 4, |G_2| = 5$

(c) $|G_1| = 7, |G_2| = 8$

Fig. 6. Some problems represented as graphs and decomposition of them. If dashed edges are removed, each graph is split into two subgraphs, one being formed by the hatched nodes, the second by the remaining nodes.

4 Results

We have implemented a distributed scheduler written in Java 1.4 [10]. Some tests were performed to verify and exemplify the usability and performance of our approach. We run all these tests on PCs with Pentium III 866 MHz processor, 256 MB RAM, under Microsoft Windows 2000.

We would like to present results for some real-life problems which were treated. Each problem description contains the complexity of the problem and the search times (for centralized and parallel search). Results are summarized in Table 3.

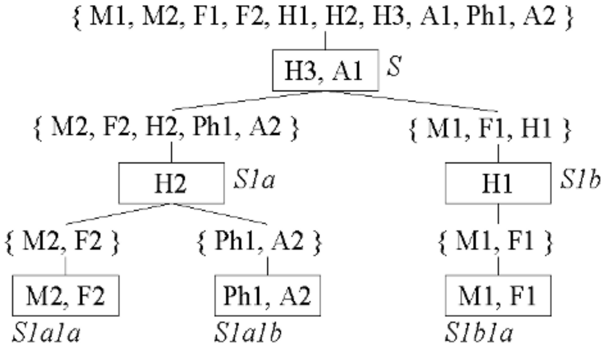


Fig. 7. Decomposition of the problem from section 2.1. Boxes represent tasks which were scheduled locally by schedulers to split the given problem into subproblems.

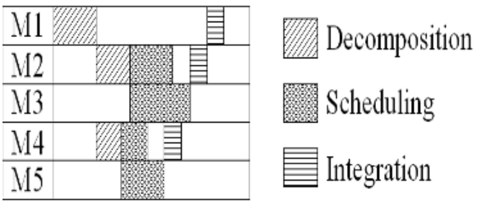


Fig. 8. Parallelization of the search. Hatched boxes represent decomposition, scheduling, and integration of found solutions.

For example, problem number 3 is the problem of construction of timetable for a high school with nine classes, 36 teachers (some of them are external teachers), and 158 subjects. All the information needed for timetable construction was released by the school management.

Thanks to our decomposition technique, we were able to parallelize the whole search which resulted in the faster searching. We hope to optimize the Decomposition Algorithm in the future.

We wanted to compare our algorithm with another algorithm based on decomposition, but were not able to find any results that were directly comparable. There are some interesting papers about decomposition algorithms [5], [6], [3], [2], but they deal with other problems which cannot be directly compared with our MRTP. We agree with paper [11], which concludes that

> Although the STP (School Timetabling Problem) is a classical optimization problem, there is still no set of test-problems with the particular characteristics described in ... (at least, none that we know of to date) that can be used as benchmark.

Table 3. Some treated real-life problems. Comparison of parallelized and centralized search.

	Complexity		Search time (s)	
Problem	Resources	Events	Centralized	Parallel
1	40	56	13.6	11.2
2	43	111	24.8	21.4
3	45	158	36.2	28.6

5 Conclusions

We have developed a model for a class of timetabling problems called the multi-resource timetabling problems. We have developed a decomposition algorithm which allows us to parallelize the search. The approach was successfully evaluated, and it has the following advantages:

1. The whole problem can be split into smaller independent subproblems which can be solved in parallel.
2. Integration of found solutions is very simple and straightforward because the all subproblems are independent. There is no need for communication or synchronization between two subproblems during the search.
3. The parallel search allows us to use modern grid computing techniques to solve our problem in a short time.

References

1. Bárta, J., Štepánková, O., Pechoucek, M.: Distributed Branch and Bound Algorithm in Coalition Planning. In: Multi-Agent Systems and Applications II. Lecture Notes in Artificial Intelligence, Vol. 2322. Springer, Berlin (2002)
2. Burke, E. K., Newall, J. P.: A Multi-Stage Evolutionary Algorithm for the Timetable Problem. IEEE Trans. on Evolutionary Computation **3** (1999) 63–74
3. Carter, M. W.: A Decomposition Algorithm for Practical Timetabling Problems. Technical Paper 83-06. Department of Industrial Engineering, University of Toronto (1983)
4. Carter, M. W., Laporte, G.: Recent Developments in Practical Course Timetabling. In: Burke, E., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 3–19
5. Friha, L., Queloz, P., Pellegrini, C.: A Decomposition Method for Hospital Scheduling Problems. In: Workshop on Industrial Constraint-Directed Scheduling (1997)
6. Goltz, H.-J., John, U.: Methods for Solving Practical Problems of Job-Shop Scheduling Modelled in CLP(FD). In: Proc. Practical Application of Constraint Technology (PACT'96) (1996)
7. Hansen, P., Mladenovic, N., Perez-Brito, D.: Variable Neighborhood Decomposition Search. J. Heuristics **7** (2001) 335–350
8. Marík, V., Štepánková, O., Lažanský, J., et al. Umelá Intelligence, Vol. 3. Academia, Praha (2001)

9. Schaerf, A.: A Survey of Automated Timetabling. CWI Report (1995) ISSN 0169-118X
10. The Source for Java Technology, http://java.sun.com. Sun Microsystems, Inc. (2003)
11. Souza, M. J. F., Maculan, N., Ochi, L. S.: A GRASP-Tabu Search Algorithm to Solve a School Timetabling Problem. 4th Metaheuristics Int. Conf. (MIC'2001) (2001)
12. de Werra, D.: An Introduction to Timetabling. Eur. J. Oper. Res. **19** (1985) 151–162

Interactively Solving School Timetabling Problems Using Extensions of Constraint Programming

Hadrien Cambazard[1], Fabien Demazeau[2],
Narendra Jussien[1], and Philippe David[1]

[1] École des Mines de Nantes, LINA CNRS,
4 rue Alfred Kastler, BP 20722, F-44307 Nantes Cedex 3, France
{hcambaza, jussien, pdavid}@emn.fr
[2] ISoft, Chemin de Moulon,
91190 Gif sur Yvette, France
demazeau@isoft.fr

Abstract. Timetabling problems have been frequently studied due to their wide range of applications. However, they are often solved manually because of the lack of appropriate computer tools. Although many approaches mainly based on local search or constraint programming seem to have been quite successful in recent years, they are often dedicated to specific problems and encounter difficulties in dealing with the dynamic and over-constrained nature of such problems.

We were confronted with such an over-constrained and dynamic problem in our institution. This paper deals with a timetabling system based on constraint programming with the use of explanations to offer a dynamic behaviour and to allow automatic relaxations of constraints. Our tool has successfully answered the needs of the current planner by providing solutions in a few minutes instead of a week of manual design. We present in this paper the techniques used, the results obtained and a discussion on the effects of the automation of the timetabling process.

1 Introduction

Timetabling problems have been frequently studied because of their wide range of applications. Scheduling activities occur in any human organization and become quickly critical for large administrations. However, timetabling problems are often solved manually because of the lack of appropriate computer tools. Although many approaches mainly based on local search or constraint programming seem to have been quite successful in recent years, they are often highly dedicated to specific problems. This is due to the wide variety of problems which differ for example for education timetabling between schools, universities, engineering schools and more specific educational institutions. Among those problems, two main difficulties are encountered: first, timetabling problems are often over-constrained and optimization criteria are hard to define. The optimization objective is often hard to express for the scheduling office itself because of the

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 190–207, 2005.

confusion induced by the manual process of resolution. Second, they are intrinsically dynamic: activities, resources or constraints are sometimes unknown or can often change at the last moment.

We were confronted with such an over-constrained and dynamic problem in our institution. We took the opportunity to experiment with PaLM [14], our explanation based constraint solver, in a real-world situation. The system is based on constraint programming with the use of explanations [15] to offer a dynamic behaviour (the problem is not solved again from scratch when constraints are added or removed) and to allow an automatic relaxation of constraints in case of over-constrained problems. The tool has already been introduced for solving dynamic RCPSP (Ressource Constrained Project Scheduling Problems) in [9]. Experiments on *academic* problems such as the RCPSP encouraged us to apply the technique in a real world situation. Our tool has successfully answered the needs of the planner by providing solutions in a few minutes instead of a week of manual computation.

This paper is organized as follows. Section 2 introduces the timetabling problem we solved. Formal models are then presented in Section 3: a linear and a constraint formulation are given. Section 4 presents the basics of explanation-based constraint programming and our results are discussed in Section 5.

2 Problem Description

The problem was met in a French engineering school: the École des Mines de Nantes (EMN). EMN offers a four-year programme which consists of a two-year common core followed by two specialized years. There are nine possible specializations in various areas of study like computer science, production and logistics, environment or management (a complete list is provided in Table 1). EMN tries to offer as much choice as possible to its students.

In this paper, we report our work on the third-year timetabling problem. All students are supposed to choose a major and a minor (among the nine possible specializations) and a set of open courses. The idea is to let them finalize their choices for their final year where either the major or the minor becomes the only set of courses that they follow. Notice that the choice of a specialization implies a fixed set of required courses. The task is to design a weekly timetable which will be in use for a complete semester. Each course available for the studied semester is to be assigned to a half-day slot during the week. The main problem is to determine which courses will happen simultaneously. The aim is to allow as many students as possible to attend all the required courses of their two choices and as many as possible of their free choices. Each option (an option is a topic or an area of study chosen by a student to specialize in during his last year) is defined by two sets of required courses (depending if the topic is a major or a minor).

2.1 Static Part of the Problem

The built timetable must verify a given set of constraints. Both hard and soft constraints can be defined. Hard constraints are as follows:

- topic availability: required courses of each topic have to be assigned non-overlapping slots;
- pre-requisite: some courses are pre-requisite for others and must therefore take place in different slots;
- room availability: a maximum number of courses is allowed simultaneously because of room limitations.

The provided timetable must meet all these hard constraints. Moreover, soft constraints are highly desirable for a useable timetable:

- teachers' availability: teachers can specify their availability on given slots during the week;
- major/minor selection: for each student, the major and minor required courses should be scheduled in non-overlapping slots;
- free course selection: additional courses chosen by a given student must fall in different slots.

Although no precise preference function is defined, the person in charge first tries to satisfy teachers as much as possible (many teachers come from other schools or universities) and to leave students free to attend their major/minor choice. Secondly, the objective is to fulfil the other students' requests in order to provide each of them with a conflict-free schedule. The main problem consists in optimizing both objectives:

1. minimize the number of violated pairs of options (major/minor);
2. minimize the number of violated pairs of courses (violated pairs of options and courses will be denoted as conflicts in the latter).

For the first semester, the priority is given to the number of pairs of options available and for the second semester, it consists in minimizing the number of conflicts between courses. (Optimization was not required at the beginning by the planner and became possible once the satisfaction of hard constraints had been successfully taken into account.)

Typical instances for the first semester of the third year in EMN involve 120 students, 30 different courses, nine main topics and seven courses per student. The 30 courses have to be assigned in seven timeslots. Notice that, even if the number of courses and students is quite small compared to typical universities, the student schedule is complete. 72 pairs of options are possible, three courses are required for a major and two for a minor. Around 30 pairs among the 72 are effectively chosen by students and the distribution of option over the pairs is varied. We cannot predict *a priori* that any two options will never be together. For example, environment and computer science options are not incompatible (a particular instance can be seen in Table 1). Therefore, the problem is often over-constrained, only considering the soft constraints concerning major/minor selection.

The *student scheduling problem* [5] (SSP) has the same objective: providing conflict-free schedules. Students need to be assigned to courses in the classical SSP (the timetable is already done) whereas our problem consists in assigning courses

Table 1. The allocation of minor and major choices in a particular instance (the pair (ACC, QRS) has been chosen by two students for instance). Options are the following: Computer systems engineering (CSE), Computer science as an aid to decision-making (CSAD), Organization and management of information technologies (OMIT), Automatic control and industrial computation (ACC), Operations management in production and logistics (OMPL), Quality and reliability of systems (QRS), Environmental engineering (EG), Energetic system engineering (ESE), Nuclear and associated technologies (NT).

Minor Major	OMIT	CSE	EG	CSAD	QRS	OMPL	ACC	NT	ESE
OMIT	–	7	1	4	1	10	0	0	0
CSE	4	–	0	13	0	0	2	0	0
EG	1	0	–	0	2	0	0	3	5
CSAD	0	6	2	–	0	1	1	0	0
QRS	0	0	0	0	–	7	0	0	1
OMPL	2	0	4	0	13	–	2	0	1
ACC	4	3	1	0	2	9	–	1	3
NT	1	0	4	0	0	0	0	- -	2
ESE	0	0	4	0	0	0	0	1	–

to timeslots (the timetable is built according to student's requests). A similar problem is solved in [22] where a feasible and personalized weekly timetable is assigned to every student of a Spanish Engineering School. But in this case again, several groups have been already done for every subject, every group has a fixed timetable and is already scheduled to lecturers. Actually, our problem can be seen as a specific *demand driven timetabling* problem [5] where the number of satisfied course requests is to be maximized.

2.2 Dynamic Part of the Problem

One of the main requirements from the scheduling office was the possibility to quickly change a previous solution in order to take unexpected events into account. Constraints concerning the students are supposed to be quite static, even if the timetabling can be computed whereas some student choices are still missing. The dynamic part of the schedule is more related to logistic problems and unexpected events concerning:

– the link with timetabling of other years (the four years share common resources: rooms, equipment for practical sessions but also teachers);
– the requirement of the teachers (insiders and specially outsiders).

It is for instance usual that the availability of an outside contributor to a course remains unknown until the last moment. Moreover, as teachers are involved in courses for different years whose timetables are not scheduled at the same period, unexpected constraints concerning teachers occur frequently. The dynamic changes considered here focus on the addition of constraints (teacher availability or logistic needs). The literature on dynamic timetabling is not very rich. However, our dynamic changes are related to the minimal perturbation problem which consists in incorporating the changes, along with the initial solution, as a new problem whose solution must be as close as possible to the previous one. [18] proposes a local search method on partial feasible assignments guided by conflicts statistics to solve the minimal perturbation problem. Our approach is quite different but also pragmatic (see Section 4.3). However, recent dedicated complete methods can also be found in [1].

3 Formal Models

The input data for a given timetable are the following:

- $\mathit{Courses} = \{1, 2, \ldots, n\}$;
- $\mathit{Options} = \{1, 2, \ldots, o\}$;
- $\mathit{Students} = \{1, 2, \ldots, m\}$;
- $\forall i \in \mathit{Options}, \mathit{Requisite}1_i \subset \mathit{Courses}$;
- $\forall i \in \mathit{Options}, \mathit{Requisite}2_i \subset \mathit{Courses}$;
- $\forall s \in \mathit{Students}, \mathit{Choices}_s \subset \mathit{Courses}$
- R is the maximum number of free rooms in a each timeslot;
- NP is the number of time periods in a week;
- $w_{i_1 i_2}$ denotes the conflicts associated to a pair of courses (i_1, i_2): for example, the number of students that took the pair (i_1, i_2).

Let *Courses* be the set of courses, *Options* be the set of options, $\mathit{Requisite}1_i$ be the first set of required courses for option i and $\mathit{Requisite}2_i$, the second set of required courses for the same option i. For each student s, the inputs are his major and minor $O1_s$, $O2_s$ and his free choice of courses $\mathit{FreeCourses}_s \subset \mathit{Courses}$. Therefore, the complete set of courses, $\mathit{Choices}_s$, corresponding to a student s is equal to $\mathit{Requisite}1_{O1_s} \cup \mathit{Requisite}2_{O2_s} \cup \mathit{FreeCourses}_s$.

3.1 A Linear Programming Model

A linear model can be expressed using Boolean variables x_{ij} which take the value 1 if the course i is placed in the timeslot j and 0 otherwise:

$$\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, \mathit{NP}\} \qquad x_{ij} = \{0, 1\}\,.$$

Integrity Constraints Each course must be placed exactly once:

$$\forall i \in \{1, \ldots, n\}, \qquad \sum_j x_{ij} = 1\,.$$

Hard Constraints

Option validity: $\forall o \in Options\,, \forall j \in \{1,\dots,NP\}\,, \sum_{i \in Requisite1_o} x_{ij} \leq 1\,.$
Linked courses: $\forall i_1, i_2$ *pre-requisite*, $\forall j \in \{1,\dots,NP\}\,, x_{i_1 j} + x_{i_2 j} \leq 1\,.$
Room availability: $\forall j \in \{1,\dots,NP\}\,, \sum_{i \in Courses} x_{ij} \leq R\,.$

Soft Constraints

Teacher availability: teacher of course i *is absent at period* p, $x_{ip} = 0\,.$
Option choice: $\forall s \in Students\,,$
$\forall j \in \{1,\dots,NP\}\,, \quad \sum_{i \in \left\{ \substack{Requisite1_{O1_s} \\ \cup Requisite2_{O2_s}} \right\}} x_{ij} \leq 1\,.$
Course choice: [1] $\forall s \in Students,$
$\forall j \in \{1,\dots,NP\}\,, \quad \sum_{i \in Choices_s} x_{ij} \leq 1\,.$

This model corresponds to the problem as it was first expressed by the scheduling office. These requirements were modified to consider an optimization model which minimizes the number of conflicts once the problem has been proved to be over-constrained. An optimization function could be added using y_{i_1,i_2} Boolean variables to express the fact that the courses i_1 and i_2 are placed in the same timeslot. An optimization function could be written (with appropriate channelling constraints to link x and y variables):

$$\min \sum_{(i_1,i_2) \in Courses \times Courses} y_{i_1 j_2} \times w_{i_1,i_2}\,.$$

3.2 A Constraint Programming Model

Constraint programming (CP) techniques have been widely used to solve scheduling problems. A *constraint satisfaction problem* (CSP) consists of a set V of variables defined by a corresponding set of possible values (the domain D) and a set C of constraints. A solution for the problem is an assignment of a value to each variable such that all the constraints are simultaneously satisfied. The constraints are handled through a propagation mechanism which allows the reduction of the domains of variables and the pruning of the search space. The propagation mechanism coupled with a backtracking scheme allows the search space to be explored in a complete way.

CP seems a good approach to our problem, as almost all the constraints described here fall within difference constraints (except room availability) i.e. courses that cannot be scheduled into the same period. We can therefore enforce our model by using global constraints like the *alldifferent* [19] or *global cardinality* [2]

[1] This constraint includes the previous one but the distinction will be made for relaxation.

[2] This is denoted by $gcc(X, lb, ub)$ where X is a set of variables, and lb and ub are two sets of integers where lb_i and ub_i gives the minimal and maximal number of times the value i must appear in X.

[20] constraints. Such constraints constitutes an efficient way to handle recurrent patterns or sub-problems, they contain complex algorithms able to efficiently prune large portions of the search space.

The constraint model proposed uses n integer variables x_i denoting the timeslots in which courses i are scheduled.

$$\forall i \in \{1, \ldots, n\}, \qquad \mathit{domain}(x_i) = [1, \ldots, \mathit{NP}] \,.$$

Hard Constraints These can be simply written as follows:

$$\begin{array}{ll} \textit{Option validity:} & \forall o \in \mathit{Options}\,, \mathit{alldifferent}(x_i | i \in \mathit{Requisite}1_o) \\ \textit{Linked courses:} & \forall i, j \textit{ linked courses}, \quad x_i \neq x_j \\ \textit{Room availability:} & \mathit{gcc}(\{x_i | i \in \mathit{Courses}\}, \{0 | (1, \ldots, \mathit{NP})\}, \{R | (1, \ldots, \mathit{NP})\}) \,. \end{array}$$

Soft Constraints

$$\begin{array}{l} \textit{Teacher availability: teacher of course } i \textit{ is absent at period } p \,, x_i \neq p \\ \textit{Option choice:} \quad \forall s \in \mathit{Students}\,, \mathit{alldifferent}\left(x_i | i \in \begin{smallmatrix} \mathit{Requisite}1_{O1_s} \\ \cup \mathit{Requisite}2_{O2_s} \end{smallmatrix} \right) \\ \textit{Course choice:} \quad \forall s \in \mathit{Students}\,, \mathit{alldifferent}(x_i | i \in \mathit{Choice}_s) \,. \end{array}$$

The problem is highly symmetric. In fact, disregarding the teachers' availability constraints, two timeslots are interchangeable. A partial assignment of at least two timeslots is equivalent to exchanging the slots. When the search algorithm has proven that one assignment is inconsistent, it is a waste of effort to try the equivalent permuted assignment.

The problem can be seen as a generalized assignment problem (GAP). It is a well-known NP-complete combinatorial optimization problem which consists of assigning a set of tasks (courses) to a set of resources (half-day timeslots). But it can also be considered as a weighted CSP which minimizes the weighted sum of unsatisfied constraints (by considering a weight for each soft constraint). The weight w_{i_1,i_2} can be associated to each elementary difference $x_{i_1} \neq x_{i_2}$. Coefficients can be added to distinguish minor choice and free course choices. However, our goal was firstly to provide a solution to hard constraints and, as requested by the planner, to quickly re-compute a solution in case of changes (often concerning teachers) or to simulate different *scenarii* with unknown data.

The dynamic and over-constrained aspect of the problem led us to consider constraint programming with specific enhancements: explanations.

4 Explanations for Constraint Programming

Solving dynamic constraint problems has led to different approaches. Two main classes of methods can be distinguished: proactive and reactive methods. On the one hand, proactive methods propose to build robust solutions that remain solutions even if changes occur. On the other hand, reactive methods try to reuse as much as possible of previous reasonings and solutions found in the past. Reactive methods avoid restarting from scratch and can be seen as a form of learning. One of the main methods currently used to perform such learning in dynamic constraint solving is a justification technique that keeps trace of inferences made by the solver during the search. Such an extension of classical constraint programming has been recently introduced. It is called explanation-based constraint programming (*e-constraints*) and it has already proved its interest in many applications [14] including dynamic constraint solving. One can refer to [9] to find an example of using explanations to solve timetabling problems. We recall in this section what explanation-based constraint programming is and how it can be used.

4.1 Definition

An explanation records information to justify a decision of the solver as a reduction of domain or a contradiction. It is made of a set of constraints C' (a subset of the original constraints of the problem) and a set of decisions, dc_1, $dc_2, \ldots, dc_n$, taken during search (e.g. the branching part of a branch-and-bound algorithm such as $x = a$ or $x < y$).

An explanation of the removal of value a from variable v will be written as follows:

$$C' \wedge dc_1 \wedge dc_2 \wedge \cdots \wedge dc_n \Rightarrow v \neq a\,.$$

An explanation e_1 is said to be more precise than e_2 if and only if $e_1 \subset e_2$. The more precise the explanation, the more relevant the learning about the inference is. When a domain is emptied, a contradiction is identified. An explanation for this contradiction is computed by uniting each explanation of each removal of value of the variable concerned. At this point, intelligent backtracking algorithms that question a relevant decision appearing in the conflict are conceivable [4]. By keeping in memory a relevant part of the explanations involved in conflicts, a learning mechanism can be implemented [16]. Notice that a *nogood* is associated with any explanation: $dc_1 \wedge dc_2 \wedge \cdots \wedge dc_n \wedge v = a$.

4.2 Computing Explanations

During propagation, constraints are awaken (like agents or daemons) each time a variable domain is reduced (this is an event) possibly generating new events (value removals). A constraint is fully characterized by its behaviour regarding the basic events such as value removal from the domain of the variables and domain bound updates. Explanations for events are computed when the events are generated.

Explanations for Basic Constraints. It is easy to provide explanations for basic constraints. The following example shows how to compute them.

Example 1. Let us consider a two-variable toy problem: x and y with the same set of possible values $[1, 2, 3]$. Let us state the constraint $x > y$. The resulting sets of possible values are $[2, 3]$ for x and $[1, 2]$ for y. An explanation for this situation is the constraint $x > y$. Now, let us suppose that we choose to add the constraint $x =$

2. The only resulting possible value for x is 2. The explanation of the modification is the constraint $x = 2$. The other consequence is that the remaining value for y is 1. The explanation for this situation is twofold: a direct consequence of the constraint $x > y$ and also an indirect consequence of constraint $x = 2$.

Precise Explanations for Global Constraints. Computing a precise explanation for global constraints may not be easy because it is necessary to study the algorithms used for propagation. However, there always exists a generic explanation: the current state of the domains of each variable of the constraints. [21] describes how to provide precise explanations for the *all-different* constraint involved in our timetabling problem.

4.3 Explanations for Handling Dynamic Problems

Incremental constraint addition to a problem is a well-known issue in classical constraint programming solvers: it is often the usual way constraints are added to the constraint system. However, incremental constraint *retraction* is not so easy. Several extensions have been proposed to handle dynamic retraction of constraints: some of them [2], [12] analyse the reduction operators to be able to determine the past effects of a constraint and so incrementally retract it; others, following [3], store information to achieve that determination.

Explanations (or simplifications) may be used as past effect determination tools [6], [15].

Dynamic Constraint Retraction. Dynamic constraint retraction of a constraint c can be achieved through the following steps in an explanation-based system:

1. *Disconnecting.* The first step is to remove the retracted constraint c from the constraint network (no further propagation).
2. *Setting back values.* The second step is to undo the past effects of the constraints, both the direct (each time the constraint operators have been applied) and indirect (further consequences of the constraint through operators of other constraints) effects of that constraint. We can easily put back in its domain, each value whose explanation contains the constraint. Past events are recorded in explanations.
3. *Re-propagate.* It is finally necessary to re-propagate in order to reach a consistent state. As our explanations are neither minimal nor unique, many ways exist to deduce the removal of a value and only one of those ways is stored in the current explanation. Therefore a new propagation is necessary.

Re-propagation of Constraints. Although determining the past effects of a constraint c is quite easily done by referring to the explanations that do contain reference to c, efficient re-propagation is not provided in classical constraint solvers. Indeed, another event needs to be handled: value restoration (or bound restoration). Instead of calling a general local consistency operator for handling this new event, it is possible to design new operators dedicated to this re-propagation phase. See [7] for more details.

4.4 Solving Over-Constrained Problems

Computing explanations for conflicts and for incrementally removing a constraint from a constraint system leads to over-constrained problems.

Consider the classical enumeration process used to solve CSP as a dynamic sequence of constraint additions (assigning variables) and retractions (backtracks). We can apply a simple strategy [15]: solve the complete constraint system *as usual* and, if necessary (i.e. once the problem has been proven over-constrained), use the conflict explanation to identify the next constraint system to consider (using a *comparator* [23] that takes into account the user's preferences) and perform the constraint modification (retraction(s) and/or addition(s)) incrementally still using explanations. This process is iterated as long as no suitable solution is found. The use of the comparator guarantees the optimality of the solution. Section 5.3 explains in more detail how this process leads to an optimization with respect to soft constraints.

Notice that search is made in the space of possible relaxations in opposition to the space of possible assignments in more classical approaches. We can carry this idea further and imagine a local search process in the space of possible relaxations (see Section 5.3).

5 Solving Dynamic Timetabling Problems

5.1 An Interactive Tool

Our system has been implemented using *PaLM* [14] and the Java Swing API to design the graphical interface. *PaLM* is launched as a background process and waits for dynamic events: addition, retraction of constraints or asking of a new solution (during search, the graphical user interface is locked). As the success of the tool was largely dependent on a user-friendly interface as well as the ability to solve the problem, we paid particular attention to the planner's needs and her manual process.

A screenshot of the implemented system can be seen in Figure 1. One can see a classic timetable representation with seven timeslots and 30 courses. Each line corresponds to a timeslot. Boxes in the left column give information on the complete timeslot (number of conflicts and number of involved students). Inside a timeslot, courses are ordered according to the number of involved students (from warm to cold colours, dark to light on the picture). Each course corresponds to a box with the following information: conflicts with every other course in the same slot, number of involved students, reference to the course and the kind of dynamic constraint added on it. Tool Tip texts help to know the purpose of each number by simply pausing with the cursor over it. Constraints can be added using a mouse left click on the course or the concerned timeslot. A new window allows the constraint and other courses or timeslots involved to be selected. The current state of dynamic constraints added since the beginning is also maintained, allowing easy removals.

The automated timetabling system implemented met the requirements of the scheduling office by proposing the following features:

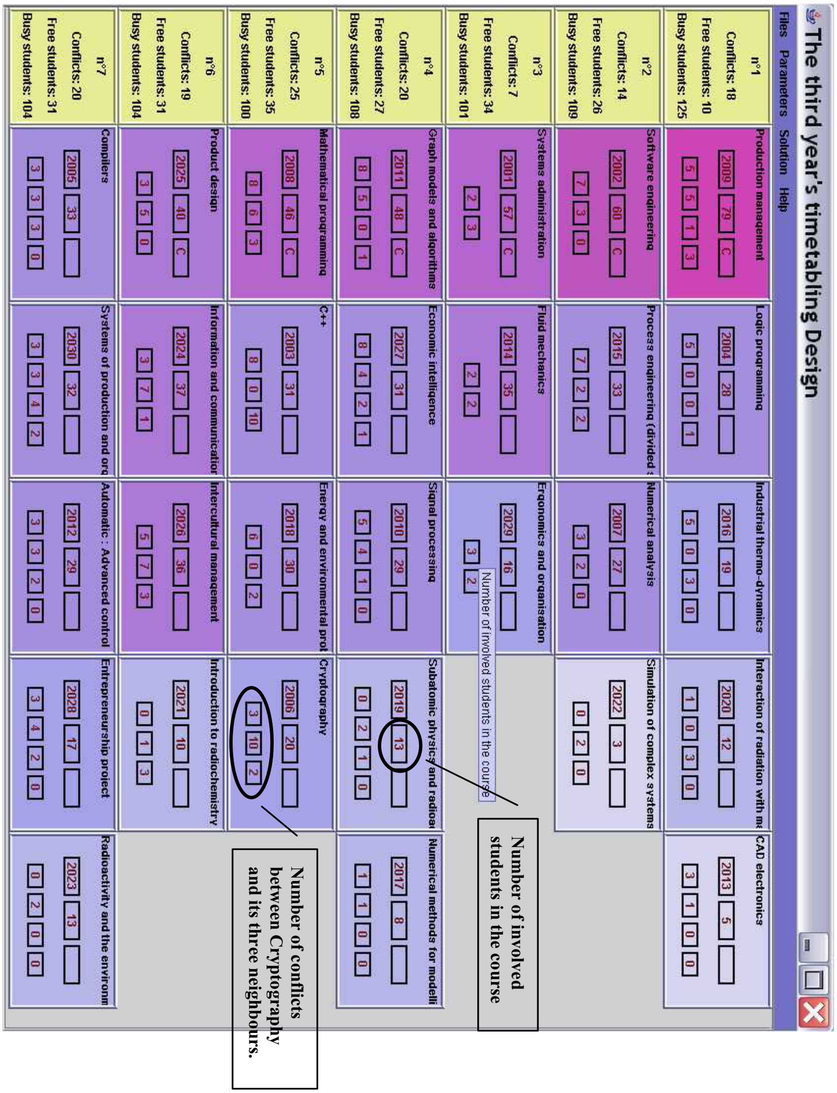
- A dynamic behaviour. Three kinds of constraints can be added or removed: equality, difference and *alldifferent*, either between courses or courses with timeslot. By dynamically adding/retracting constraints or changing their weight, the planner is able to perform simulations on the solution, to evaluate its robustness and to *manually* build (in interaction with the tool) good solutions.
- A complete visualization of conflicts in each timeslot and for each course with its neighbours offers the possibility to make judicious *dynamic* changes to a solution. Previous indicators used by the planner in the manual process, such as the number of inactive students in a timeslot, have been included. Additional statistics on a solution are accessible through a menu to evaluate the results in more details.

5.2 Results

The first results were of great interest for the planner:

- The tool is able to provide better solutions quickly. A solution was found manually in a week when 10 solutions are now obtained in about 10 minutes. Moreover, solutions have a better quality than those found by hand even if no optimization is done. Nevertheless, the planner is forced to pre-allocate some courses to allow a quick computation of a solution. This is a simple way to reduce the symmetries of the problem. A good way of doing this consists in setting the required courses of the most chosen option in different slots, as they will not be found together in any conceivable good solution.

  We compared the result during the start-up year of the tool and noticed that conflicts were reduced by around 20% in both semesters. Moreover, the number of students able to follow their minor choice increased by 3% (this corresponds to a reduction of 10% of the number of unsatisfied students concerning their minor). The tool was good enough to be accepted by the scheduling office and, since then, six semesters have been scheduled with it.
- Unexpected events, or unknown data can be taken into account quickly before the start of the courses without changing the original solution too much. The system does not provide any guarantee on the perturbation. It does not ensure one obtains the minimal perturbation. However, the stability of a solution has been analysed for scheduling problems in [8], [10], [11] using the same technique and proved to be quite effective.
- A simulation tool: the tool is able to be quickly used by the planner to simulate different situations. Common-sense assumptions were made to make the manual scheduling process easier and are now questioned and simulated with the tool. For example, Human social science courses were often put together in the last timeslot considering that each student is forced to choose one course in this field. However, best solutions do not necessarily respect such assumptions. Dynamic abilities intend here to take advantage of the know-how of the planner as much as possible because she remains a central element in the process of resolution.


The third year's timetabling Design
Files Parameters Solution Help

n°1 Conflicts: 18 Free students: 10 Busy students: 125	Production management 2009 79 C 5 5 1 3	Logic programming 2004 28 5 0 0 1	Industrial thermo-dynamics 2016 19 5 0 3 0	Interaction of radiation with ma 2020 12 1 0 3 0	CAD electronics 2013 5 3 1 0 0
n°2 Conflicts: 14 Free students: 26 Busy students: 109	Software engineering 2002 60 C 7 3 0	Process engineering (divided : 2015 33 7 2 2	Numerical analysis 2007 27 3 2 0	Simulation of complex systems 2022 3 0 2 0	
n°3 Conflicts: 7 Free students: 34 Busy students: 101	Systems administration 2001 57 C 2 3	Fluid mechanics 2014 35 2 2	Ergonomics and organisation 2029 16 Number of involved students in the course 3 2		
n°4 Conflicts: 20 Free students: 27 Busy students: 108	Graph models and algorithms 2011 48 C 8 5 0 1	Economic intelligence 2027 31 8 4 2 1	Signal processing 2010 29 5 4 1 0	Subatomic physics and radioa 2019 13 0 2 1 0	Numerical methods for modelli 2017 8 1 1 0 0
n°5 Conflicts: 25 Free students: 35 Busy students: 100	Mathematical programming 2008 46 C 8 6 3	C++ 2003 31 8 0 10	Energy and environmental prot 2018 30 6 0 2	Cryptography 2006 20 3 10 2	
n°6 Conflicts: 19 Free students: 31 Busy students: 104	Product design 2025 40 C 3 5 0	Information and communicatior 2024 37 3 7 1	Intercultural management 2026 36 5 7 3	Introduction to radiochemistry 2021 10 0 1 3	
n°7 Conflicts: 20 Free students: 31 Busy students: 104	Compilers 2005 33 3 3 3 0	Systems of production and org 2030 32 3 3 4 2	Automatic : Advanced control 2012 29 3 3 2 0	Entrepreneurship project 2028 17 3 4 2 0	Radioactivity and the environm 2023 13 0 2 0 0

Number of involved students in the course

Number of conflicts between Cryptography and its three neighbours.

Fig. 1. A timetable with seven timeslots and 30 courses. Light boxes on the left give information on the complete timeslot (number of conflicts and number of students concerned). In a timeslot, courses are ordered according to the number of students concerned (from warm to cold colours). Each course corresponds to a box with the following information: conflicts with every neighbours, number of involved student and the kind of added manual constraints. Tool Tip Texts help to know the purpose of each number.

5.3 Post-optimization Processing

When solving an instance by hand, the planner focused on the satisfaction of hard constraints and teacher wishes. As hard constraints are successfully taken into account with the current system, the planner can focus more on student wishes. Once he/she is experienced with the tool, the time gained is used to optimize the solution through a simulation process. We noticed how the planner brought her know-how into play to interact with the tool using its dynamic abilities and to focus on the optimization objective. This was a new aspect of her work, as she had been more focused on hard constraints in the past. It was therefore, for us, a second step of resolution.

Her manual process can be seen as a *manual* local search around the first solution given by the system and we came naturally to the point of automating the technique as a local search.

User Preferences. Preferences are taken into account during the relaxation of constraints. When the problem has been proved over-constrained, we use the current contradiction explanation and the user preferences to identify the constraint to be relaxed. The use of a simple comparator (a comparator is a partial order on configurations) [23] that considers the constraint with the minimal weight ensures that the heaviest weight among the relaxed constraints is minimal in any solution. It guarantees the optimality of the solution according to this comparator. A configuration is defined by two sets of constraints (A, R) where A is the set of active constraints and R the set of relaxed constraints. A configuration C is said to be preferred to C' according to a given comparator. In the previous case, two configurations are compared on the heaviest weight of their constraints.

The solutions provided in this way are close to the planner's objective and first solutions have been shown to be good solutions. However, a more precise objective could be the minimization of the weighted sum of relaxed constraints.

Such a comparator needs all the contradiction explanations computed during the search to determine (each time a new contradiction occurs) the new set (of constraints) of minimal weight that covers every contradiction explanation. This leads to two main problems:

- the number of contradiction explanations can be exponential;
- finding the set of minimal weight covering the whole set of explanations is an NP-hard problem (set covering problem).

To overcome those problems and try to provide better solutions, we decided to perform local search techniques. This choice was directed by the well-known ability of local search techniques to efficiently improve an initial solution generated using a heuristic method. However, a very different approach for performing the search has been tried using the dynamic abilities of our system.

Local Search in the Space of Configurations. Local search techniques can be seen as methods which iteratively apply simple moves to a solution. Among the wide variety of techniques, tabu search is a local metaheuristic which avoids

```
input: a configuration $(A, R)$ and the constraint problem pb,
begin
    while time limit not reached do
        for each ct in $R$ do
            add ct to $A$ and changes its weight to 100
            $(A', R') \leftarrow$ searchOneSolution(pb) in less than N new decisions
            if (conflict$(A', R')$ < conflict$(A, R)$) then
                bestMove $\leftarrow$ ct
            endif
            restore previous configuration $(A, R\ )$ by:
                    removing ct from $A'$
                    adding all $c \in A \backslash A'$
        endfor
        update tabu list:
                store bestMove and its weight in the list and if the size is exceeded:
                remove from the list the first constraint and restore its weight
        add bestMove to $A$ with a weight of 100
        $(A, R\ ) \leftarrow$ searchOneSolution(pb)
    endwhile
end
```

Fig. 2. Tabu search in configurations space.

entrapment in local minima. It has already proved its use in solving timetabling problems [13]. We use here one of the simplest forms of tabu search without achieving an effective balance of intensification and diversification. But instead of performing the search on complete or partial assignments as usual, we perform the search in the space of configurations using the dynamic abilities of an explanation-based constraint solver.

Let (A, R) be the configuration reached at the end of the first step of the search. As the use of a simple comparator gives good results, we start from this point a local search in a second step of resolution. The tabu search is performed by trying to add a constraint from R to A. The neighbour function (a move) consists therefore in adding a relaxed constraint to the set of active constraints. To avoid cycling, the weight of the constraint is dynamically changed to an infinite value (in our case, 100). Adding a constraint with an infinite weight to A can lead to very difficult sub-problems. Therefore, to reduce the computational effort of each move, we limited the number of extensions (or backtracks, or repairs) allowed to find a new solution. Once the constraint of R leading to the best solution has been added to A, we store it in a tabu list with its original weight. As long as the constraint appears in the list, its weight stays infinite in the problem, forbidding its relaxation. When the constraint leaves the list, it takes back its original weight and becomes relaxable.

Local search methods traditionally encounter difficulties in taking hard constraints into account. Notice that the solution provided here after each move satisfies hard constraints because arc-consistency is maintained after each retraction

Table 2. Results obtained on three years. Instances X1 correspond to a first semester problem and instances X2 to a second semester.

	First solution			Tabu (5 min)		
Instances	Opt	Conflicts	Obj. fct	Opt	Conflicts	Obj. fct
A1	6	162	282	5	138	238
B1	11	151	391	13	140	380
C1	7	160	320	5	133	273
Improvement:			10.3%			
A2	–	54	54	–	49	49
B2	–	97	97	–	90	90
C2	–	117	117	–	103	103
Improvement:			9.7%			

or addition of constraint. When no solutions are found for every constraint of R, we can decide to increase the number of extensions allowed or to stop the search.

The neighbourhood of a configuration (A, R) is defined by transferring a constraint from R to A, the computation of a consistent state with this move leads to relaxation of some constraints from A to R. Therefore, the configuration reached (A', R') can be far from the original one. The limitation of the number of extensions to re-compute a solution from the previous one ensures remaining in a close neighbourhood of (A, R) and limiting the time needed to compute a move. Moreover, after proving that a move is not reachable in the limit of time allowed, it can be removed from the neighborhood until the current configuration has been changed enough. It is a waste of time of proving at each step that a move does not not belong to the current neighbourhood.

Results on Optimization. The tabu is able to improve the first solutions by around 10% in both semesters. It compensates one main problem of the simple comparator used by the solver to relax constraints: all the constraints of one level can be relaxed whereas only one constraint of a higher level needs to be relaxed to get a solution (this is called a flooding phenomenon). Table 2 summarizes the conflict on major/minor choices (option choices in column *Opt*), free choices in column *Conflicts* and the value of the optimization function considered to perform optimization.

6 Discussion

The tool can successfully solve the problem even if optimization is still an open question. Of course, as mentioned before, it is still necessary to pre-affect some courses by hand to reduce symmetries and to obtain an efficient resolution. We would like to emphasize that the tool is unable to replace the planner's work.

On the contrary, although much progress has been made in this research field, timetabling problems remain very difficult and specific. Computer tools still need to be open and interactive to take advantage of the know-how of the planner. In our case, the tool has significantly changed her work and opened new perspectives. Common-sense assumptions, needed when the problem is solved by hand (to reduce its complexity) can now be questioned and others evaluated. By executing thankless work, the tool gives freedom to the planner who can analyse the problem without any reductionist assumptions. Therefore, her understanding and way of doing it have considerably changed with the tool which gives efficient and relevant information about the problem. Only the planner knows what makes a *good* solution, has in mind the complete problem with its human dimension and is used to students and teachers' behaviour. That is why an objective function is so hard to define clearly and why the know-how of the planner is so critical.

We have tried to develop an efficient tool based on recent technologies in order to answer precise needs, which is a quite different objective compared to the resolution of very hard and academic problems.

7 Conclusions and Further Work

We have presented in this paper a practical application of an explanation-based system to solve a school timetabling problem. The system has proved its efficiency when it was accepted by the scheduling office. First, our intention was to show the interest of using explanation techniques to solve dynamic and over-constrained timetabling problems. Second, we presented a different local search technique that tries to take advantage of both constraint programming for satisfying hard constraints and local search for its performance in an optimization context. We plan in the future to keep working on the problem itself as well as the tool:

- Our next step would be to improve the system with *user-friendly* explanations [17] to answer to legitimate questions of the planner such as: Why is the problem over-constrained? Contradictions could be explained to the user by providing the explanation computed for the contradiction in a high level representation.
- Further experiments have to be done concerning the optimization process. After the local search phase, we intend to restart the search with the new upper bound. Efficient lower bounds and branching scheme will therefore be needed. A more precise study on the problem has to be carried out to evaluate the quality of the first solution found.

Although work remains to understand the problem, these first results show that explanation constraint programming can be successfully integrated in a real-world situation.

Acknowledgements

The authors would like to thank Valérie Kirion, in charge of timetabling of the third year students, for her help and collaboration.

References

1. Barták, R., Müller, T., Rudová, H.: A New Approach to Modeling and Solving Minimal Perturbation Problems. In: Apt, K. R., Fages, F., Rossi, F., Szeredi, P., Vancza, J. (eds.) Recent Advances in Constraints. Lecture Notes in Computer Science, Vol. 2809. Springer, Berlin (2004) 233–249
2. Berlandier, P., Neveu, B.: Arc-Consistency for Dynamic Constraint Problems: A RMS Free Approach. In: Proc. ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications (1994)
3. Bessière., C.: Arc Consistency in Dynamic Constraint Satisfaction Problems. In: Nat. Conf. on Artificial Intelligence—AAAI'91 (1991)
4. de Backer, B., Béringer, H.: Intelligent Backtracking for CLP Languages: An Application to CLP($\mathcal{R}$). In: Saraswat, V., Ueda, K. (eds.) ILPS'91: Proc. Int. Logic Programming Symposium (San Diego, CA). MIT Press, Cambridge, MA (1991) 405–419
5. Cheng, E., Kruk, S., Lipman, M.: Flow Formulation for the Student Scheduling Problem. In: Burke, E. K., De Causmaecker, P. (eds.) Practice and Theory of Automated Timetabling IV. Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 299–309
6. Debruyne, R.: Arc-Consistency in Dynamic CSPs is No More Prohibitive. In: 8th Conf. on Tools with Artificial Intelligence (TAI'96) (1996) 299–306
7. Debruyne, R., Ferrand, G., Jussien, N., Lesaint, W., Ouis, S., Tessier, A.: Correctness of Constraint Retraction Algorithms. In: FLAIRS'03: 16th Int. Florida Artificial Intelligence Research Society Conf. (St Augustine, FL). AAAI Press, Menlo Park, CA (2003) 172–176
8. Elkhyari, A.: Outils d'Aide à la Décision Pour les Problèmes d'Ordonnancement Dynamique. Ph.D. Thesis. École des Mines de Nantes, France (2003) in French
9. Elkhyari, A., Guéret, C., Jussien, N.: Solving Dynamic Timetabling Problems as Dynamic Resource Constrained Project Scheduling Problems Using New Constraint Programming Tools. In: Burke, E. K., De Causmaecker, P. (eds.) Practice and Theory of Automated Timetabling IV. Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 39–59
10. Elkhyari, A., Guéret, C., Jussien, N.: Constraint Programming for Dynamic Scheduling Problems. In: Hiroshi Kise, H. (ed.) ISS'04 Int. Scheduling Symp. (Awaji, Hyogo, Japan). Japan Society of Mechanical Engineers (2004) 84–89
11. Elkhyari, A., Guéret, C., Jussien, N.: Stable Solutions for Dynamic Project Scheduling Problems. In: PMS'04 Int. Workshop on Project Management and Scheduling (Nancy, France, April 2004) 380–384
12. Georget, Y., Codognet, P., Rossi, F.: Constraint Retraction in CLP(FD): Formal Framework and Performance Results. Constraints **4** (1999) 5–42
13. Hertz, A.: Tabu Search for Large Scale Timetabling Problems. Eur. J. Oper. Res. **54** (1991) 39–47
14. Jussien, N.: E-Constraints: Explanation-Based Constraint Programming. In: CP01 Workshop on User-Interaction in Constraint Satisfaction (Paphos, Cyprus, December, 2001)
15. Jussien, N., Boizumault, P.: Best-First Search for Property Maintenance in Reactive Constraints Systems. In: Int. Logic Programming Symp. (Port Jefferson, NY). MIT Press, Cambridge, MA (1997) 339–353
16. Jussien, N., Lhomme, O.: Local Search with Constraint Propagation and Conflict-Based Heuristics. Artif. Intell. **139** (2002) 21–45

17. Jussien, N., Ouis, S.: User-Friendly Explanations for Constraint Programming. In: ICLP'01 11th Workshop on Logic Programming Environments (WLPE'01, Paphos, Cyprus, December, 2001)
18. Müller, T., Rudová, H.: Minimal Perturbation Problem in Course Timetabling. In: Proceedings of the 4th International Conference on the Practice And Theory of Automated Timetabling (PATAT'04, Pittsburgh, PA) 283–303
19. Régin, J.-C.: A Filtering Algorithm for Constraints of Difference in CSPs. In: AAAI 94, 12th Nat. Conf. on Artificial Intelligence (Seattle, WA) 362–367
20. Régin, J.-C.: Generalized Arc Consistency for Global Cardinality Constraint. AAAI/IAAI (1996) 209–215
21. Rochart, G., Jussien, N., Laburthe, F.: Challenging Explanations for Global Constraints. In: CP03 Workshop on User-Interaction in Constraint Satisfaction (UICS'03, Kinsale, Ireland, September) 31–43
22. Santiago-Mozos, R., Salcedo-Sanz, S., DePrado-Cumplido, M., Bousoño-Calzón, C.: A Two-Phase Heuristic Evolutionary Algorithm for Personalizing Course Timetables: A Case Study in a Spanish University. Comput. Oper. Res. **32** (2005) 1761–1776
23. Wilson, M., Borning, A.: Hierarchical Constraint Logic Programming. J. Logic Program. **16** (1993) 277–318

A Tiling Algorithm for High School Timetabling

Jeffrey H. Kingston

School of Information Technologies,
The University of Sydney,
2006, Australia
jeff@it.usyd.edu.au

Abstract. This paper presents a tiling algorithm for high school timetabling. The meetings are grouped into small, regular clusters called tiles, each of which is thereafter treated as a unit. Experiments with three actual instances show that tiling, coupled with an alternating path algorithm for assigning resources to meetings after times are fixed, produces good, comprehensible timetables in about ten seconds.

1 Introduction

As a recent survey makes clear [1], the problem of automatically constructing timetables for high schools remains far from solved. This paper offers a new approach based on grouping meetings together into small clusters called *tiles*. Although there are some drawbacks in doing this, there are significant advantages: the resulting timetable is comprehensible to the timetable planner; assignment of teachers to classes is simplified; and run times are reduced to about ten seconds, freeing the timetable planner to explore alternative scenarios quickly.

After a description of the high school timetabling problem (Section 2), this paper offers an overview of tiles and the tiling algorithm (Section 3). Sections describing the phases of the algorithm follow. Results are presented for three instances taken from a high school in Sydney, Australia (Section 8).

2 Specification

High school timetabling problems vary from place to place. The problem described here is the one occurring in Australian high schools.

In high school timetabling problems, the meetings are timetabled on a weekly or fortnightly *cycle*. Time is partitioned into *periods* of equal length. One common pattern is a week of 40 periods, each 40 minutes in length, spread over five days. Adjacent periods may be adjacent in time, or separated by a meal break or by the end of a day.

Unlike university timetabling, where each student follows an individual timetable, high school students are grouped together into *student groups* (often called *classes*, but that word can also mean the meetings they attend, and is so used here). The members of one student group (typically 30 students) follow the same

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 208–225, 2005.

timetable and may be treated as a unit. Four or five student groups, containing all the students of a certain age group, make one *form* (also called a *year*). Australian high schools have six forms, called Year 7, Year 8, and so on to Year 12.

A typical high school has 50 or more teachers. They are partitioned into *faculties*: subject areas, such as English, History, Mathematics, and so on. Some teachers have qualifications spanning more than one faculty, and some meetings (such as Sport) are taught by teachers in several faculties. Within faculties the teachers have further specialties, and our model allows any number of *capabilities* (such as Drama, or Senior History) to be defined and granted to arbitrary subsets of the teachers. A teacher may have many capabilities, not just one. Teachers have *quotas* specifying their workload. A common example would be a teacher whose quota is 30 periods per week (out of the maximum of 40), with a preference for at most 7 periods on any one day (out of the maximum 8). Head teachers or teachers with other duties have smaller quotas, and there are part-time teachers who are available only on nominated days.

Although most rooms are ordinary classrooms, there are specialised rooms such as Science laboratories (possibly also usable as ordinary classrooms) and Computer laboratories. Again, our model allows any number of these capabilities to be defined and granted to arbitrary subsets of the rooms. Rooms could also have quotas or be unavailable at certain times, for example for maintenance, although there are no cases of this in our data.

Student groups, teachers, and rooms (collectively,*resources*) play fundamentally the same role in timetabling: each must be assigned to meetings, avoiding clashes. However, there are differences in detail. Student group resources attend something at every period of the week, and are always preassigned to meetings—there is never a request to *choose* a student group, as there is with teachers and rooms. Teachers differ from rooms in that it is important for the same teacher to attend all of the periods allocated to each class meeting. One does not want Ms Smith to take some English class for three of its six periods, and Mr Brown to take it for the other three. That would be a *split assignment*, and it is permissible but undesirable.

A *meeting* is an entity in which a set of resources meet together at a set of times. The times and resources may be *preassigned*, meaning fixed in advance to particular values, or they may be left open to the solver to choose. Meetings may request any number of times, and may request blocks of adjacent times not separated by breaks (these are called *double periods*, *triple periods*, etc). It is preferred for these times to be spread evenly through the week, and that undesirable times (such as the last period on any day, when the students are tired and restless) should not be concentrated in one meeting. Student group resources are always preassigned. In our data, teachers and rooms are usually *not* preassigned. Instead, meetings request resources with given capabilities: a History teacher, a Science laboratory, or whatever.

In traditional class–teacher timetabling [6], each meeting contains one teacher and one student group, but our meetings typically request more resources than this. For example, in the junior years the students in each form may be grouped by ability for Mathematics, meaning that the Mathematics classes of that form must run simultaneously, leading to one large meeting requesting four or five student groups, Mathematics teachers, and rooms. In the higher years there are *electives*: sets of meetings planned to run simultaneously so that students can choose one. There may be seven or eight teachers, with varying capabilities, plus rooms in such meetings.

Occasionally there are *composite classes*, in which students from different forms study a specialised subject together. Although the students follow different curricula, the common subject matter makes such a class feasible. Composite classes are created when the school wants to offer some subject as an elective, but there are too few interested students in any one form to justify it. Their effect on timetabling is to cause two electives from different forms (those containing the specialised option) to be merged.

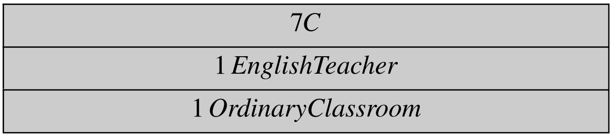
Our data also contain several kinds of staff meetings, requesting various preassigned subsets of the teachers, but no student groups or rooms. These meetings are equivalent to free time for the purposes of calculating teachers' daily and weekly quotas, but they can be involved in clashes like other meetings.

The objective is to assign times, teachers and rooms with the desired capabilities to the slots, avoiding clashes and not overloading any teachers. These two requirements are hard constraints and they dominate the problem. If necessary, a resource assignment may be split between two teachers as described above, and as a last resort the smaller of the two fragments may be occupied by a teacher not qualified for the requested capability. Meetings must receive the number of times they request, but the block structure and spread through the week of these times are soft constraints.

3 Overview

In this section we define tiles and present an overview of our algorithm. Following sections then explain its phases in more detail.

Consider a typical meeting, such as the English class of student group 7C in the *bghs98* instance. This meeting requests 6 times including two double periods, student group resource 7C, one English teacher, and one ordinary classroom. We can think of this meeting as a 3 × 6 rectangle:



Its *width* is the number of times requested, although sometimes we give a sequence of numbers for the width, defining the required block structure. For example, width 2 2 1 1 requests six times including exactly two double periods.

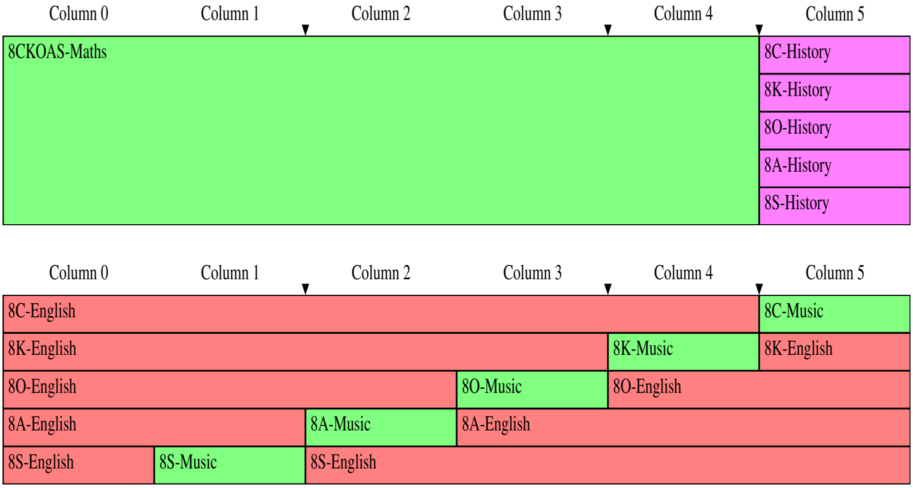


Fig. 1. Two examples of tiles from the *bghs98* instance. Each has width 2 2 1 1, as marked by the wedges. Each smaller rectangle represents one meeting, including appropriate resources (not shown).

Its *height* is the number of resources required, although again we often give more detail—a list of the resources required—and call that the height.

Several meetings may be grouped together into a larger entity of a specific width and height, which we call a *tile*. For example, suppose we decide to run the English classes of the five Year 7 student groups simultaneously. This produces a tile of width 6 and height 15, containing the five Year 7 student group resources, five English teachers, and five ordinary classrooms.

Figure 1 contains two other examples of tiles. The students are grouped by ability for Mathematics, so the five Mathematics classes must run simultaneously and are combined into one large meeting in the input data. The adjacent History classes do not have to run simultaneously, but fitting them neatly alongside Mathematics forces them to. The second tile illustrates a construction, well known to manual timetablers, called the *runaround*. There are only two Music teachers and two Music rooms, so the five Music classes cannot run simultaneously. By interleaving them among other meetings as shown, the tile demands only one of each at any one time.

Our recipe for producing comprehensible timetables may now be stated: first place all of the meetings neatly into tiles, then timetable the tiles so as to ensure that for each pair of timetabled tiles T_i and T_j, T_i and T_j either contain exactly the same set of times, or else have no times in common. In our test instances there are 40 times in the week, and the timetable is planned around an ideal pattern of six classes each six periods wide, plus four periods of sport and optional religious instruction, so it is natural to build six six-period tiles and one four-period tile in each form. This common set of widths, or *major columns*, could be inferred, but it forms such a basic part of the timetable planner's thinking that we have chosen instead to require it to be given as part of the input data. When timetabled, the four-period tiles must all contain the same set of four times, if the rule laid down just above is to be satisfied; but the six-period ones may be timetabled together in whatever way utilizes resources best. The reader might care to look ahead to Figure 3, which shows a timetable following this pattern.

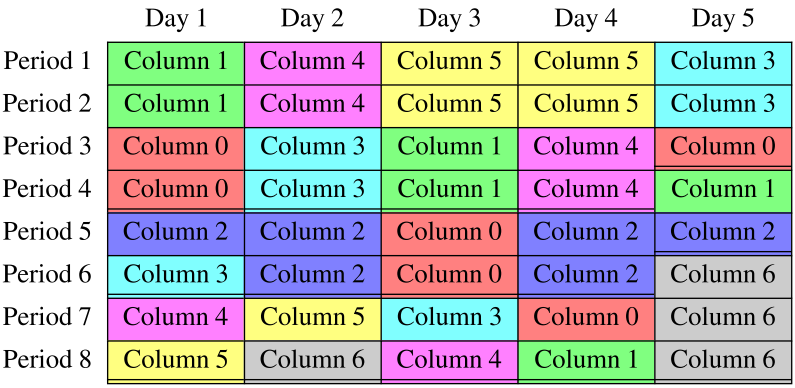
Our algorithm proceeds in four phases, each the subject of a following section. First, specific times during the week are assigned to the major columns (Section 4). Next, tiles are created and meetings inserted into them and timetabled within them (Section 5). Third, the tiles are timetabled against each other by placing them into the major columns (Section 6), using an algorithm that first timetables the Year 12 tiles, then the Year 11 tiles, and so on, building up layers until the timetable is complete. Finally, specific resources are allocated to the meetings' resource slots (Section 7).

4 Column Layout

The first step in our algorithm is to assign times to each of the major columns. Figure 2 gives a typical example of what is wanted: each column spread evenly through the week, with its blocks of times not interrupted by meal breaks.

This is an easy problem in practice so we will be brief. A tree search is used which first attempts to give each column one period on each of a set of days that is spread evenly through the week. Once this is achieved the search continues downwards, with columns making requests to days for their single periods to be exchanged for larger blocks, until every column has the block structure it requires. Each day maintains a small bin packing of the blocks it has promised to columns into the intervals between meal breaks, solved exhaustively as each request arrives.

Columns may have preassigned times, propagated from classes. For example, if some class requests Mon1 and Mon2, then a preassignment of these times to



	Day 1	Day 2	Day 3	Day 4	Day 5
Period 1	Column 1	Column 4	Column 5	Column 5	Column 3
Period 2	Column 1	Column 4	Column 5	Column 5	Column 3
Period 3	Column 0	Column 3	Column 1	Column 4	Column 0
Period 4	Column 0	Column 3	Column 1	Column 4	Column 1
Period 5	Column 2	Column 2	Column 0	Column 2	Column 2
Period 6	Column 3	Column 2	Column 0	Column 2	Column 6
Period 7	Column 4	Column 5	Column 3	Column 0	Column 6
Period 8	Column 5	Column 6	Column 4	Column 1	Column 6

Fig. 2. A typical layout of a week of 40 times into six columns of width 2 2 1 1 plus one of width 3 1. Double lines indicate meal breaks.

some column will occur. Time preassignments are rare and we assume that they do not overconstrain this problem.

In practice this algorithm finds layouts like the one in Figure 2 with virtually no backtracking. It has produced larger (two-week) layouts with ease. At present it does not try to equalize the number of morning and afternoon times granted to each column (a common soft requirement), but this could easily be added in a final stage which permutes blocks of times within days.

5 Tile Layout

The second phase of our algorithm builds, for each form, a set of tiles holding the meetings of that form. This task is usually trivial in the senior forms, where the set of widths of all meetings equals, or almost equals, the set of tile widths, but non-trivial in the junior years, which typically contain many small meetings. For example, the Year 8 meetings from the *bghs98* instance have widths

2 2 1	*English*	2 2 1	*Mathematics*	2 2 1	*Science*
2 2	*Languages*	2 2	*Health*	1 1 1	*Geography*
1 1 1	*History*	2	*Art*	2	*Sport*
2	*Technology*	2	*Design*	1 1	*Music*

and must be packed into the usual six tiles of width 2 2 1 1 and one of width 3 1. We wish to minimize the number of meetings split across two tiles, creating a bin packing problem.

Some of these entries represent single meetings (e.g. Mathematics) while others represent a set of five meetings, one for each student group (e.g. English). For comprehensibility, even in this second case we prefer to place all the meetings for one subject into the same tile.

Our current algorithm uses a tree search which first searches for a packing that does not split any meetings over two tiles. If that fails, it splits the smallest meeting in two in all possible ways and tries again. If that fails it splits the two smallest meetings in two in all possible ways, and so on. It usually works quite well (Section 8) but during development has occasionally entered on a long, fruitless search. We plan to replace it with a heuristic method which we believe will do just as well in practice, so instead of describing our current algorithm further we now present some considerations of importance to any tile layout algorithm.

Although we place all the meetings for a single subject within one tile, it is not always possible for them to run simultaneously there, because resources may not be sufficient, and this leads to the runarounds already mentioned (Section 3). The key quantity is the *minimum runaround width*, the minimum number of times that a set of meetings must spread through if its demands are to be satisfied. For example, five one-period Music classes must spread through at least three times if two Music teachers are all that are available. In general, the minimum runaround width is the maximum, over all resource demands made by the meetings, of

$$\left\lceil \frac{\mathit{Width\ of\ one\ meeting} \times \mathit{Number\ of\ occurrences\ of\ demand}}{\mathit{Number\ of\ resources\ available\ to\ satisfy\ demand}} \right\rceil .$$

Each set of meetings is classified as *vertical* (meaning that the meetings must run simultaneously, as in the case of Mathematics where the input data demands it), *runaround* (meaning requiring a runaround, because the minimum runaround width is greater than each meeting's width), or *easy*, meaning that either layout will work. For example, in Figure 1, Mathematics is vertical, Music is runaround, and the others are easy.

For a set of meetings to be timetabled within one tile it is of course necessary, to begin with, that the total width of all meetings containing any one student group resource should not exceed the tile width. Beyond this, there must be room for the runaround meetings to spread out in. When several sets of runaround meetings lie in the same tile, they interleave with each other but still occupy width equal to their total width, so this total must be at least as large as every minimum runaround width. Easy meetings may be co-opted into the runaround, as English is in Figure 1, to help achieve this *runaround condition*, which is sufficient as well as necessary for a timetable to exist, provided that the resource demands of the different sets of meetings do not interact.

Some tiles contain meetings from several forms, and consequently when a tile layout algorithm begins it may find that some of the tiles it is given are not empty. It must check for each of its sets of meetings whether resources are sufficient to permit them to enter such tiles. In particular, when some major column's width is unique, as is the case for the width 4 column of the *bghs98* instance, we only ever create one tile for that column, and that tile holds meetings from every form.

There is an artificial *fixed form* of *fixed tiles* holding meetings with preassigned times. Each fixed tile is permanently assigned to a particular major column, and since the times assigned to columns are known at this point, any meetings with preassigned times can find their way via the columns to the fixed tiles they belong in. These fixed tiles also record resource unavailabilities at particular times, converted in the usual manner into artificial meetings occupying those resources at those times. If a form contains a meeting with one or more preassigned times, that meeting will have already been assigned to the corresponding fixed tile when the form's tile layout begins, and that fixed tile will be one of the tiles handed to the tile layout algorithm.

To summarize, the aim of a tile layout algorithm is to assign its sets of meetings to tiles, avoiding violations of the runaround condition and resource sufficiency problems, minimizing the number of meetings that are split in two, and paying attention to block structure. Some of the meetings it is given may have already been assigned to some of its tiles, and these preassignments must be respected. We believe that a heuristic algorithm that assigns the widest meetings first, keeping vertical and runaround meetings apart as far as possible, looking ahead to avoid resource sufficiency traps, and splitting one meeting in a best-fit manner whenever it gets stuck, will do all this very well.

After meetings are allocated to tiles, they are timetabled within them so as to satisfy resource limits and time block structure requests as far as possible. For single-form tiles this is a small search problem easily solved to optimality; for large multi-form tiles, see the remarks at the end of Section 6.

Although it has not occurred yet in our data, it is quite possible for the minimum runaround width of some set of meetings to exceed the tile width. In that case the runaround must spread over more than one tile, or perhaps it could trigger *part-form tiling*, where the student group resources of one form are partitioned into two parts, each of which is then treated as a separate form. At present our algorithm always partitions the Year 7 and Year 8 forms into two part-forms, using a simple clustering algorithm to decide which student groups to place in each partition. These decisions could be automated, or optionally taken from the timetable planner. Most of the meetings in the higher forms are vertical, so there is nothing to gain from part-form tiling those forms.

6 The Main Timetabling Phase

After all the meetings have been allocated to tiles, and timetabled within them, the next step is to timetable the tiles against each other; that is, to assign the tiles to columns. We call this the *main timetabling* phase.

Underlying any main timetabling algorithm will be a test, probably called many times, for determining whether a given set of tiles is *compatible*: able to run simultaneously without exceeding resource limits. We consider this compatibility testing problem first.

The simplest way to test a set of tiles for compatibility is to merge their meetings into one large tile of the same width and timetable it. This is likely to be too slow when many calls on the test are made.

Three faster compatibility tests have been tried. Each gives an upper bound on the number of unassignable tixels (a *tixel* is one resource at one time) that will result from placing the tiles into the same major column. All three methods assume that the individual tiles have been timetabled, and never redo these individual timetables.

The simplest method uses a worst-case measure of the demand for resources made by each tile. If demand varies at different columns of the tile, we take for each type of demand the maximum over the columns. For example, if the five classes do Mathematics simultaneously for some of the tile's times, and History simultaneously for the rest, then the tile's demand will include five Mathematics teachers *and* five History teachers, but only five rooms.

Now form a bipartite graph with one left-hand node for each resource of the instance, and one right-hand node for each resource demand made by each tile in the set being tested. Edges join demand nodes to all resources qualified to satisfy that demand. Find a maximum matching in this graph. The desired upper bound is the number of unmatched demand nodes, multiplied by the common tile width to give a result in tixels.

The second method is a refinement of the first, in which the demand nodes are weighted by the number of tixels that would be deficient if the node remained unmatched. For example, suppose a tile requires two Computer laboratories at two of its times, one Computer laboratory at two of its times, and no Computer laboratories at the other two times—a total of six tixels altogether. This would be represented by two Computer laboratory demand nodes, the first weighted 4 and the second 2. The maximum matching is now required to minimize the weight of the unmatched nodes. For example, if one of our two Computer laboratory nodes was unmatched, it would be the weight 2 one, reflecting the fact that withholding one laboratory from this tile would cost 2 unassigned tixels. If both were unassigned the cost would be 6 tixels. The total weight of unmatched nodes gives a more refined measure of incompatibility, and finding maximum node-weighted matchings is not much harder than finding unweighted ones.

The test we currently use is a yet further refinement. It is considerably slower than the first two, but still fast enough for our purposes. Each tile is assumed as before to be timetabled in one fixed way. The test works by combining the tiles one by one into a larger tile, so let us suppose that $k$tiles have been taken and we now wish to add in the $(k{+}1)$st.

For each column (individual time) of the combined tile, find the demands made on resources by that column. Do the same for each column of the tile to be added. Take the first column of the combined tile and the first column of the tile to be added, merge their demands together into one bipartite graph, find a maximum matching, and count the number of unmatched nodes. This is the number of unallocated tixels that would result if these columns were aligned. Do this for every combination of one column from the combined tile and one from the incoming tile—if the tile width is W, a total of W^2 tests.

Now build a complete bipartite graph whose left-hand nodes are the columns of the combined tile, and whose right-hand nodes are the columns of the incoming tile. Weight the edge connecting a pair of nodes by the outcome of the test on the corresponding columns. Find a maximum matching of minimum cost in this graph, giving a permutation of the columns of the incoming tile that minimizes the number of unallocated tixels. Permute the timetable of the incoming tile according to this matching, and merge the tile into the combined tile. Repeat until all tiles are merged. The total weight of the last min-cost matching will then be an upper bound on the number of unallocatable tixels if these tiles are timetabled together.

This last test has the advantage when tiles with tall, thin demands meet. For example, in the *bghs98* instance there is a Year 9 tile that requires (among other things) five Physical Education teachers simultaneously for two of its six times, and a Year 10 tile that also requires five Physical Education teachers for two simultaneous times. There are five Physical Education teachers altogether. The first test would rate the incompatibility of these two tiles at 30 tixels (five teachers times six times), the second at 10 tixels (five teacher nodes of weight two each), while the third recognises that there are no unallocatable tixels at all.

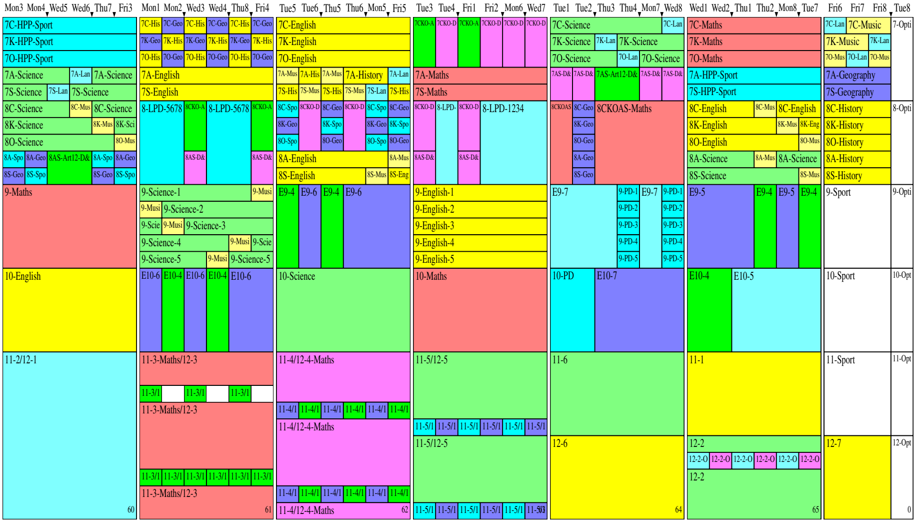


Fig. 3. A timetable for the *bghs98* instance described in Section 8, created by the algorithm of Section 6. Each row contains the meetings attended by one student group. Each narrow column represents one of the 40 times of the week; these are grouped into seven major columns, within which traces of the tiles that were placed in those columns are clearly visible. The tile at bottom left is an example of a multi-form tile; it spans Years 11 and 12. One can also see part-form tiles for Year 7 at the top, one set for student groups 7C, 7K, and 7O, the other for 7A and 7S.

Several algorithms for the main timetabling phase have been implemented and tested, including a set covering algorithm (which generates many sets of compatible tiles and then tries to select some sets which contain every tile exactly once), various tree search algorithms, and an augmenting path algorithm inspired by the algorithm of Section 7. In no case did any of these other algorithms outperform the algorithm about to be described.

Take each form in turn and assign its tiles to suitable columns, beginning with the fixed form and ending with the part-form forms, which have small height and so make good fillers of cracks. Suppose that k forms have been allocated to columns in this way and we now wish to allocate the $(k+1)$st form. Test each tile in the $(k+1)$st form for compatibility with each column. To be considered even minimally compatible the tile must have the same width as the column and not have been previously assigned to any other column (a tile may be in multiple forms, in which case it will be assigned a column along with the first of its forms that is timetabled, and must not be assigned to some other column afterwards). Build a bipartite graph in which the left-hand nodes are the columns and the right-hand nodes are the tiles of the current form. Edges join tiles to those columns with which they are minimally compatible. These edges are weighted by the number emerging from the compatibility test of this tile with this column. Find a minimum cost maximum matching in this graph; its cost will be an upper bound on the number of unassignable tixels created by adding in these tiles. Make the assignments of tiles to columns indicated by this matching and proceed to the next form. An example of a timetable created by this algorithm appears in Figure 3.

The algorithms for testing tiles for compatibility and for the main timetabling are essentially the same, only operating at different scales. Both algorithms are weighted versions of the meta-matching algorithm of [2], without the look-ahead tests employed there.

Should some kind of tree search seem indicated in future, one interesting one we have tried is based on finding an alternating cycle of minimum cost in the min-cost flow at each level. Applying this cycle gives the maximum matching of second-minimum cost, giving two matchings of each form so that a small binary tree of alternative assignments may be searched.

After the tiles' columns are fixed, each column receives a final timetabling which attempts to give its meetings the block structures they require, while minimizing unallocated tixels. This is a general timetabling problem, and despite being limited to one column it can still be challenging. Our current algorithm embarks on a long tree search which is terminated early to keep running time down. It is slow and unreliable and needs to be replaced by an algorithm based on (but not limited to) the matchings found when constructing the column, so we will say no more about it here. Instead we offer a method of reducing the problem size which should be useful to any algorithm for timetabling multiple forms within one column.

Consider a meeting that happens to stretch the full column width. Its resource demands naturally affect the feasibility of timetabling the column, but because they are constant at every position, they cannot influence any meeting in the column to choose one position over another.

Next consider the set of all meetings in a column containing a given student group resource, and suppose that together they stretch to the full column width (as is almost always the case). A resource demand common to all these meetings is effectively a full-width demand with the properties just outlined. For example, if student group 7C attends English for one part of a tile and Geography for the rest, and both classes require an ordinary classroom, then that requirement might as well lie in a single full-width meeting.

When such requirements are subtracted out it is often possible to recognise that the timetabling problem for some forms is independent of other forms. For example, if Year 7 is attending English and Geography while Year 10 is attending Science and Music, then the two forms may be timetabled independently. We frequently find that the number of forms that must be timetabled together is reduced to three or four using this analysis.

7 Resource Allocation

After times are assigned to meetings, the last major phase of our algorithm is to assign resources. The method to be used here could be applied to any situation in which resources are to be assigned after the times of meetings have been fixed, although, as we will see, it relies to some extent on the coherence provided by tiling.

At first sight resource allocation may seem trivial, since time assignment is supposed to guarantee that for each time, resources are sufficient to cover all the slots of all the meetings running at that time. However, merely using the resources provided by that guarantee would produce large numbers of split assignments.

A tree search algorithm was tried for this problem. The teachers were taken one faculty at a time and assigned in all possible ways. Despite full propagation of constraints and grouping of equivalent resource slots to avoid searching symmetrical situations, this method was never able to search the full tree, and often the best solution it could find in a reasonable time (several minutes) was easily improved by a small chain of exchanges.

This experience suggested a switch to an alternating path algorithm, as used in bipartite matching. Choose a currently unfilled slot of maximum width. See if there is a teacher able to fill this slot (the teacher must be free at its times, and adding the slot to the teacher's current load must not overload the teacher). If so, assign that teacher and move on to the next widest unfilled slot. If not, see if there is a teacher who would be able to fill this slot if only one of the slots that teacher is currently teaching was taken away and given to some other teacher able to fill it. If so, make the indicated chain of two assignments and one deassignment, and move on. If not, look for a longer chain of three assignments and two deassignments, and so on. In searching for these chains, possible assignments and deassignments are marked *visited* when they are first considered. No already visited assignment or deassignment may be revisited in the course of one search. This prevents loops, and ensures that only a limited amount of time is spent assigning any one slot. If the search fails, then the slot is left unassigned for the time being.

The chain of assignments and deassignments is actually carried out by the algorithm as the search proceeds. At any moment where the state is *feasible* (where no teacher is overloaded), the total weight of the assignments in that state is compared with the total weight of the best solution found so far, and the best solution is replaced with the current one if the current weight is greater. The weight of an assignment is the number of times in the slot being assigned (so that large slots are favoured over small ones). If quota overflows are allowed the amount of any overflow is subtracted from the weight, to discourage overflowing even though it is permitted. Split assignments (see below) are also discouraged by being given smaller weights.

A few failed slots can be retrieved by a second pass, attempting again to assign slots that failed the first time around. This achieves nothing in the traditional applications of the alternating path method, where the first pass produces an optimal result, but in our more complex situation it does produce an occasional extra assignment. After that, a third pass is made in which split assignments and partial assignments are permitted, as we now describe.

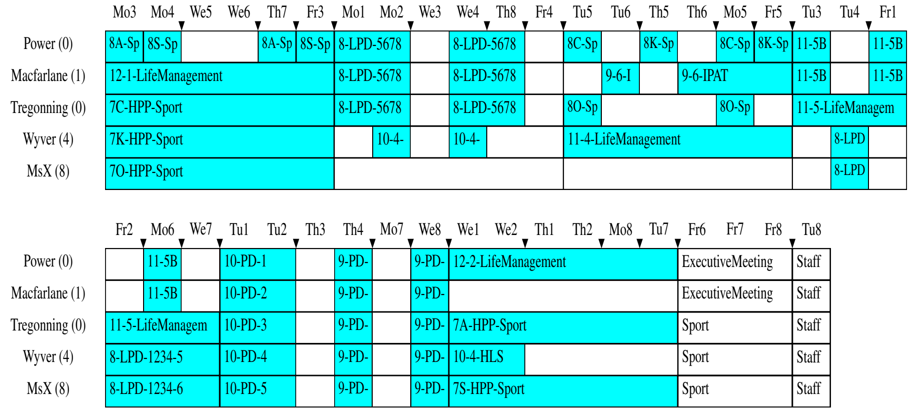


Fig. 4. Part of the teacher allocation for instance *bghs98*, showing the teachers and classes for Physical Education. This allocation is perfect: all classes are covered, there are no split assignments, and as many teachers as possible are teaching Sport outside the faculty. The number in parentheses after each teacher is the remaining unused portion of the teacher's quota. This faculty is lightly loaded.

Enhancing the basic algorithm to include split assignments is quite straightforward: a split assignment merely affects the quotas of two teachers rather than one. The problem with split assignments is not one of definition, it is their large number. Consider a slot occupying T times, and for which there are R qualified resources. Then there are R possible ordinary assignments (one for each qualified resource), but $R(R-1)(2^{T-1}-1)$ split assignments: choose an arbitrary nontrivial subset of the times, assign an arbitrary qualified resource to that subset, assign some other qualified resource to the remaining times, and divide by two to correct for counting every assignment twice. For typical values such as $R = 10$ and $T = 6$ this is already in the thousands and growing rapidly. Introducing thousands of objects in order to model something that we would rather not use anyway does not seem cost-effective; we have not tried it.

As mentioned earlier, we choose not to introduce split assignments at all initially, to give the algorithm a chance to show what it can do without them. When they are eventually introduced, they are in the form of *assignment factories* rather than the assignments themselves. These factories are lists of qualified resources and subsets of times, from which assignments can be generated. When a factory for some meeting is present, it will be asked to produce a single assignment which would terminate the search at that meeting. It then searches its lists for a pair of resources that can split the current slot between them while *both* remaining not overloaded. Any such assignment is added to the pool of assignments and competes with them. If not used it is removed again. Partial assignments are handled in the same way.

This algorithm has proved to be fast and almost perfect (Section 8). It does not need to assign the teachers faculty by faculty as the tree search method did. Some detailed examples of its results appear in Figures 4 and 5. We offer the

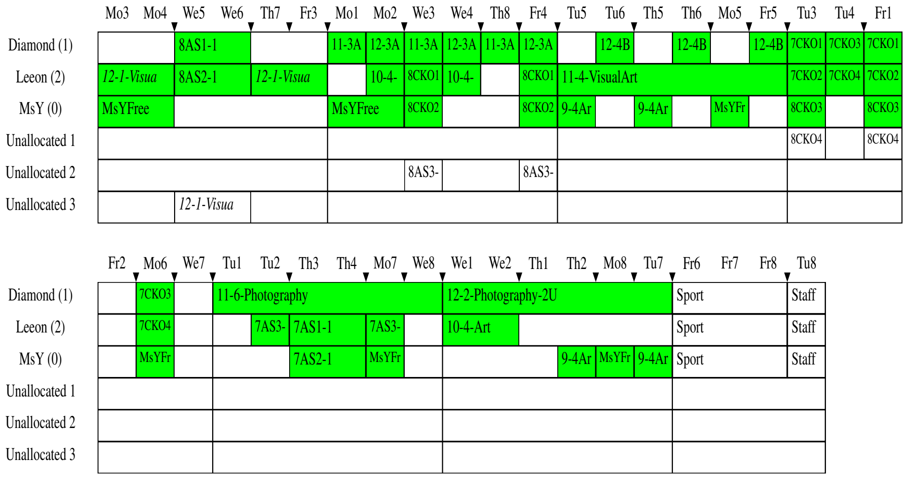


Fig. 5. Another part of the teacher allocation for instance *bghs98*, showing the teachers and classes for Art. This allocation is much less perfect: there is one split assignment, shown in italics, partly unallocated, for which a correcting alternating path exists that the algorithm has failed to find; plus two other unallocated classes, caused by assigning four simultaneous Art classes when there are only three Art teachers. If these meetings' times were moved, the Art teachers would need to be assigned less Sport in order to take them, since the total remaining unused teachers' quota is only 3 times. This suggests that it would have been better if some Art classes had been timetabled over Sport, rather than classes from some other subject in which the teachers are more lightly loaded.

following explanation for its success in a context where the guarantees it usually operates under are absent.

The literature contains occasional references to an exact network flow algorithm for resource allocation (for example, [1] cites [4] on this point). It is easy to find such a network for our problem in the special case where all meetings require the same number of times, each pair of meetings either clashes completely or not at all, and no split assignments are allowed. Then limits on the number of *times* a teacher may teach may be converted into limits on the number of *classes* a teacher may teach, by dividing by the common meeting length and rounding down. Each path in the network begins with an edge from the source to a node representing one teacher, of capacity equal to the number of classes that teacher may teach, then proceeds via an edge of capacity 1 to a node representing that teacher's availability at a particular set of times, thence to each node representing a class at that set of times that the teacher is qualified for, and from there to the sink with capacity 1.

It is easy to verify that the usual max-flow algorithm on this network is equivalent to our algorithm above. There is also a matroid intersection formulation whose equivalence is even more intuitive. Thus, in this special case, our algorithm is optimal.

When we move to the general problem, two sources of NP-completeness appear: when meetings vary in length, fitting them into teachers' quotas is a bin packing problem; and when they clash in arbitrary ways, the clash graph (in which nodes are meetings and edges join pairs of clashing meetings) changes from a set of disjoint cliques to an arbitrary graph, producing a node colouring problem [3]. Thus the alternating path method cannot be optimal in the general case, but we argue now that it stands an excellent chance of doing well nevertheless.

The algorithm sorts the meetings by decreasing number of times, assigning the meetings with the most times first. In bin packing terms this is the 'first fit decreasing' heuristic, for which there are quite good performance guarantees [5]. In the common situation where the largest meeting size is equal to the column width, and no meetings of that size are split across two columns, the algorithm will be optimal while assigning these large meetings.

The node colouring intractability is mitigated by the use of tiles. Most meetings occupy a single tile, so the clash graph is close to the set of disjoint cliques of the tractable special case. Meetings split across two tiles spoil the disconnectedness, and meetings that occupy less than the full column width may not clash with other short meetings in their column; but these cases are in the minority.

8 Results

This section analyses the performance of our algorithm on three instances taken from a high school in Sydney, Australia. A statistical description of these instances appears in Table 1. These instances contain staff meetings, which are not yet included in our solutions except when their times are preassigned. Staff meetings do not affect teacher quotas, they merely make the teachers involved unavailable when they are running.

Run times for the four phases and in total are given in Table 2. Total times were checked against wristwatch time. Both the tile layout and main timetabling phases include laying out meetings within the tiles created by those phases, and this operation in its current defective state (Section 6) dominates the cost of both these phases, so speed improvements can be expected here in future. The times given for resource allocation include a stopgap attempt at room allocation using the teacher allocation algorithm. When a dedicated room allocation algorithm is installed, resource allocation time should decrease by two to three seconds.

The quality of the solutions found for the three instances is summarized in Table 3. The algorithm always assigns the correct number of times to each meeting, never introduces student group clashes, and prefers to leave teacher and room slots unassigned rather than introducing teacher and room clashes. So the possible defects are time layout problems (wrong number of double periods, meeting spread over too few days, etc.), missing teacher and room assignments, and split teacher assignments.

The number of meetings with some kind of time layout problem is quite high at present (over 40% in two instances), but this is less serious that it may seem.

Table 1. Statistical description of the three instances tested, showing the number of times in the week, meetings, teachers, rooms, and student groups. The last three lines show, for each of the three resource groups, the number of tixels of demand for that resource group as an absolute number and as a percentage of the number of tixels of supply for that resource group. (A tixel is one resource at one time.) For students and rooms the tixel supply is just the number of resources times the number of times in the week, but for teachers it is less owing to teachers' quota limits. The demand for student groups is less than 100% because a few final year students attend marginally less than full time.

Instance description	*bghs93*	*bghs95*	*bghs98*
Times in the week	40	40	40
Meetings	148	146	152
Teachers	53	52	56
Rooms	46	48	45
Student groups	23	27	30
Teacher demand (tixels)	1489 (95.3%)	1378 (95.4%)	1408 (96.6%)
Room demand (tixels)	1295 (70.4%)	1306 (68.0%)	1357 (75.4%)
Student group demand (tixels)	872 (98.0%)	1041 (99.3%)	1197 (99.8%)

Table 2. Run times in seconds for the three instances tested. The tests used a 1.2GHz Pentium IV running Redhat Linux 5.1. Run times are as reported by the Linux *time* command, which is accurate to one second.

	bghs93	*bghs95*	*bghs98*
Column layout	0.0	0.0	0.0
Tile layout	2.0	7.0	1.0
Main timetabling	2.0	6.0	5.0
Resource allocation	6.0	6.0	6.0
Total time	10.0	19.0	12.0

Table 3. Solution quality for the three instances tested. The table shows both the absolute number of each possible kind of defect, and the number as a percentage of the number of meetings, room tixels, teacher slots, or teacher tixels as appropriate.

	bghs93	*bghs95*	*bghs98*
Meetings with at least one time layout problem	19 (13.8%)	63 (43.2%)	63 (41.4%)
Room tixels unassigned during time assignment	33 (2.5%)	12 (0.9%)	26 (1.9%)
Teacher slots split by resource assignment	13 (2.9%)	31 (6.7%)	21 (4.8%)
Teacher tixels unassigned during time assignment	4 (0.3%)	12 (0.9%)	10 (0.7%)
Teacher tixels unassigned during resource assignment	11 (0.7%)	25 (1.8%)	20 (1.4%)
Total teacher tixels unassigned	15 (1.0%)	37 (2.7%)	30 (2.1%)

Most of the problems concern assigning one more or less double period than was requested, and often there would not be a strong preference about this in any case. We hope to capture better data concerning time layout preferences in future, including optional alternatives and priorities, and this plus the planned new algorithm for distributing meetings to tiles (Section 5) should reduce the number of time layout problems to an acceptable level.

We have not yet written a dedicated room assignment algorithm, so the table only reports the number of tixels for which rooms are not available after the main timetabling phase. Since room constancy is not required this is probably the exact number of unassigned rooms that will occur in the end (the only possible problem being the need for room constancy in double periods). These unassignable room demands are for specialised laboratories whose demand is very tight. This problem is quite common in high schools and is not of major concern, since, given its low relative frequency, it is not difficult to ensure that no class meets in an inappropriate room for more than one of its times, and the teacher would organise the classroom material accordingly. Our remarks below about reducing unallocated teacher tixels also apply to rooms.

The number of split teacher assignments seems to be close to optimal now. In other experiments, not reported in detail here, in which teachers were allowed to take just one more period than their quota, but with a penalty if this occurred, the number of teachers who exceeded their quota was quite modest (between 10 and 20), and the number of split assignments typically halved. This, along with hand analysis, provides good evidence that split assignments are mainly needed to pack classes into teachers' quotas, and thus are inevitable.

This leaves the problem of assigning qualified teachers to classes. Although as a last resort an unqualified teacher may be assigned, this is considered much worse than assigning an inappropriate room, and our absolute numbers of unassigned slots are at present too high for our timetables to be usable.

Unassigned teacher tixels are created in two ways. First, during the main timetabling, a decision may be made to run tiles simultaneously that results in the demand for teachers with a certain capability at some time exceeding the number of teachers qualified for that capability available at that time. The number of tixels affected by defective main timetabling in our instances is quite small (4, 12, and 10), and close examination shows that some of them are caused by defective layout of meetings within large tiles. Our new algorithm (Section 6) should correct that problem. Beyond that, it will be necessary to break open the tile structure to swap fragments of classes containing unassignable teachers to other times where teachers are available. Hand analysis of our current solutions indicates that this will succeed most of the time. Any of the meetings contributing to the excessive demand may be moved, and we would naturally choose to try to move small classes rather than large electives.

The second chance to create unassigned teacher tixels comes during resource assignment, when the resource allocation algorithm is unable to assign a teacher or split teacher to a slot, even if there are teachers free. Examination of the data suggests that many of these problems arise from various imbalances in the supply of teachers.

For example, some faculties are lightly loaded, so their teachers should take some classes from outside the faculty. But the number of such outside classes is very limited (in our instances, essentially only Sport), so care must be taken to ensure that the faculty's own classes are not scheduled at the same time as these other classes. Our algorithm is currently quite blind to the need for this, although we can diagnose the situation well by comparing supply with demand for each faculty. On occasions during development we have observed solutions in which the right decisions were made fortuitously, and these contained significantly fewer unassigned tixels than reported in our formal results.

9 Conclusions

The work reported in this paper is ongoing, and our results at the time of writing are not perfect. Nevertheless, they show that it is possible to construct high school timetables of high quality in about ten seconds. Our key innovations are tiling and an effective resource allocation algorithm based on alternating paths. The alternating path algorithm might find application in other resource allocation tasks, although it does rely on tiling to ensure that the graph colouring problems it faces are not too severe.

Things to do immediately include getting staff meetings into tiles, replacing the tile layout algorithm, replacing the algorithm for timetabling the meetings within a single tile, writing a room allocator, and writing code for breaking open the tile structure and swapping small parts of meetings that failed to receive their needed resources to better times. Ideas for detecting and correcting resource supply imbalances could be developed further.

When all this is done our timetables should be good enough to show to high schools. Australian high schools are just now receiving broadband Internet connections, so exciting prospects are opening up for delivering timetabling across the Internet. Fast response time will be important.

References

1. Carter, M. W., Laporte, G.: Recent Developments in Practical Course Timetabling. In: Burke, E., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 3–19
2. Cooper, T. B., Kingston, J. H.: The Solution of Real Instances of the Timetabling Problem. Comput. J. **36** (1993) 645–653
3. Cooper, T. B., Kingston, J. H.: The Complexity of Timetable Construction Problems. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 283–295
4. Dyer, J. S., Mulvey, J. M.: Computerized Scheduling and Planning. New Directions Inst. Res. **13** (1977) 67–86
5. Garey, M. R., Johnson, D. S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
6. de Werra, D.: An Introduction to Timetabling. Eur. J. Oper. Res. **19** (1985) 151–162

Project Scheduling

Lower Bounds for the Multi-skill Project Scheduling Problem with Hierarchical Levels of Skills

Odile Bellenguez and Emmanuel Néron

Laboratoire d'Informatique de l'Université de Tours,
64 avenue Jean Portalis, 37200 Tours, France
odile.bellenguez@etu.univ-tours.fr
emmanuel.neron@univ-tours.fr

Abstract. In this paper, we introduce an extension of the classical Resource-Constrained Project Scheduling Problem: the Multi-skill Project Scheduling Problem. We consider a project made up of activities that must be implemented by a staff: every member of this staff masters one or more skill(s). An activity needs a given amount of each skill with a fixed minimum level of mastering. For each unit of a skill needed, we have to assign an employee who masters the required level of this skill during the whole processing time of the activity. The objective is to minimize the duration of the project, i.e. the makespan. We introduce here two lower bounds used to evaluate the minimum duration.

1 Introduction

This paper deals with the Multi-skill Project Scheduling Problem (MSPSP) with hierarchical levels of skill. In this problem, there is a project to schedule, in which activities need to be done by staff members who master specific skills at specific level of ability. It can be seen as an extension of the classical Resource-Constrained Project Scheduling Problem with multiple modes of execution (MM-RCPSP), but in our problem the number of modes allowed for each activity corresponds to the number of subsets of staff members that are able to satisfy needs, and this number can be very large. This is the reason why methods used for the MM-RCPSP, for example in [7], [13], [16], [12], [18], [23], cannot be used directly for the MSPSP. Here the goal is to find some efficient lower bounds that are not too time-consuming, in order both to evaluate heuristic methods (as for example in [6]) and to be used in a branch-and-bound method. Section 2 presents the problem; Section 3 is devoted to lower bounds, then we show experimental results (Section 4), before concluding in Section 5.

2 Multi-skill Project Scheduling

The MSPSP is a model that can be applied to different cases of project management. We focus on a particular case of multi-skill project scheduling that appears when we have to take into account different levels of skill abilities.

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 229–243, 2005.

2.1 General Case

As in the classical project scheduling problem (RCPSP, MM-RCPSP), we want to schedule a project in a minimum elapsed time, respecting resource and precedence constraints. A project is made up of a set of activities A_i, $i \in \{0, \ldots, n\}$, that represent all the steps of the project. Each activity has to be processed without preemption. Between the activities of the project, there exist precedence relations $(A_i,\ A_j)$ that can be modelled using an activity-on-node graph $G = (A, E, d)$. In this precedence graph, we include the dummy source A_0 and the dummy sink A_n that represent the beginning and the end of the project, respectively. Each activity has given needs of resources. In our case, the resources are staff members that have to be assigned to activities.

Moreover, the resources, which are staff members P_m, $m \in \{0, \ldots, M\}$, cannot be chosen arbitrarily. To satisfy each need of an activity A_i, we have to assign a person according to his/her skill that must match the required skill during the whole processing time p_i of activity A_i. Actually, a staff member can master or not each skill needed in the project. Thus, to schedule an activity we have to choose among all the staff members who possess the required skill for each need. Besides that, the employees may not be available all along the time horizon; this means we are allowed to assign a person for the need of an activity A_i starting at time t_i only if he/she is available during the total duration of the activity, i.e. during $[t_i, t_i + p_i[$.

2.2 Hierarchical Levels of Skills

There has been considerable research on workforce planning and project scheduling. For example, the nurse rostering problem [8] is well studied: it assigns employees to satisfy all the needs on all the shifts, trying to find a fair solution for everybody. The course timetabling problem has also been studied (for example in [1]). It consists in finding a timetable for each lesson, respecting teachers availabilities and the rooming constraints. The authors in [14] present a problem related to course scheduling but to minimize the total cost. The paper [2] treats simultaneously the problem of minimizing the project duration and the associated manpower cost. Although the notion of skill has been studied with respect to some workforce planning problems (as in [25]), to the best of our knowledge, very few papers deal with hierarchical levels of skills in the field of project scheduling [15].

For the MSPSP with hierarchical levels of skills, we evaluate for each employee the level l of quality he/she guarantees regarding skill S_k according to his/her experience. In the same way, we evaluate for each activity its level of difficulty. Then, we can deduce S_k^l, the level l of mastering skill S_k that will be required, and the number $b_{i,k}^l$ of employees that will be required for level l of this skill S_k to process A_i. So, we have to assign to each need somebody that masters this skill at least at the required level. We also know the total number of persons required, for each level of each skill, allowing this number to be equal to zero.

The notation used in this paper is defined in Table 1.

Table 1. Input data and auxiliary notation

Activity data	
n	the index of the last activity,
A_i, $i \in \{0, \dots, n\}$	the set of activities of the project: A_0 is dummy start node and A_n the dummy end of the project,
p_i, $i \in \{0, \dots, n\}$	the processing time of activity A_i,
$(A_i, A_j) \in E$	if there exists a precedence relation between A_i and A_j,
$G = (A, E, d)$	the precedence graph.
Resource data	
K	number of skills,
L	number of levels per skill,
M	number of staff members,
S_k, $k \in \{0, \dots, K\}$	set of skills: k is the number of the skill,
S_k^l, $k \in \{0, \dots, K\}$, $l \in \{0, \dots, L\}$	set of levels of skills; k is number of the skill and l the level (1 is first level of mastering skill and L is the highest level),
P_m, $m \in \{0, \dots, M\}$	staff members,
$S_{m,k} = l$, $m \in \{0, \dots, M\}$, $k \in \{0, \dots, K\}$	the maximum level l at which P_m can do S_k,
to simplify: $S_{m,k}^{l'} = 1$	$\forall l' \leq l$, if P_m can do S_k at level l, 0 otherwise,
$A(P_m, t)$, $m \in \{0, \dots, M\} = 1$ $t \in \{0, \dots, T_{\max}\}$	if P_m is available at time t, 0 otherwise,
$b_{i,k}^l$, $i \in \{0, \dots, n\}$ $k \in \{0, \dots, K\}$, $l \in \{0, \dots, L\}$	number of persons able to do S_k at level l, required to execute A_i.
Auxiliary notation	
r_i	release date of activity A_i,
$\tilde{d}_i$	deadline of activity A_i,
t_i	starting time of activity A_i,
$A(P_m, t_1, t_2) = \sum_{t=t_1}^{t_2-1} A(P_m, t)$, $m \in \{0, \dots, M\}$	total time P_m is available between t_1 and t_2.

2.3 Example and Integer Linear Program

The MSPSP model can be applied to some project scheduling problems that arise in the software development industry, where employees are programmers, analysts, designers, etc [15]. In this section we give an example of such a problem. The project to be implemented is made up of four activities linked by the precedence relationship presented in Figure 1.

Needs of the activities are summarized in Table 2 and skills of employees are presented in Table 3. There are two levels per skill. We are interested in schedules that minimize the makespan of the project. Finding a solution consists in fixing starting times of all the activities of the project and assigning a subset of staff

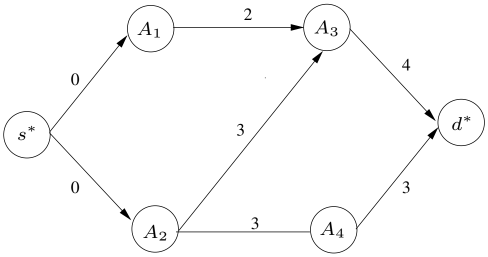


Fig. 1. Precedence graph of the project of the example

Table 2. Needs of activities of the example

	Skill S_1		Skill S_2		Skill S_3		Skill S_4	
$b^l_{i,k}$	S_1^1	S_1^2	S_2^1	S_2^2	S_3^1	S_3^2	S_4^1	S_4^2
A_1	1	1	0	0	1	0	1	0
A_2	0	0	0	1	0	0	1	0
A_3	0	0	1	0	2	0	0	0
A_4	1	0	0	0	0	1	0	0

Table 3. Skills of employees of the example

	Skill S_1		Skill S_2		Skill S_3		Skill S_4		Unavailability period(s)	
	S_1^1	S_1^2	S_2^1	S_2^2	S_3^1	S_3^2	S_4^1	S_4^2	no.	(start, end)
P_0	1	1	1	1	0	0	0	0	1	$(2,4)$
P_1	0	0	1	1	1	1	0	0	0	–
P_2	0	0	1	0	1	1	1	1	0	–
P_3	1	0	1	0	1	0	0	0	2	$(2,3)$; $(6,8)$
P_4	1	0	0	0	0	0	1	1	0	–
P_5	1	0	0	0	1	1	0	0	1	$(7,10)$

members to each activity, according to their needs. We present a feasible solution for this problem in Figure 2.

The MSPSP can be modelled by the integer LP formulation below, where

- $x_{i,m,t} = 1$ if P_m begins to work for A_i at time t, 0 otherwise
- $\delta^l_{i,m,k} = 1$ if P_m does S_k at level l for A_i, 0 otherwise:

$$\forall (i,j) \in E, \frac{\sum_{m=0}^{M} \sum_{t=0}^{T_{\max}} x_{i,m,t} \cdot t}{\sum_{k=0}^{K} \sum_{l=0}^{L} b^l_{i,k}} + p_i \leq \frac{\sum_{m=0}^{M} \sum_{t=0}^{T_{\max}} x_{j,m,t} \cdot t}{\sum_{k=0}^{K} \sum_{l=0}^{L} b^l_{j,k}}, \qquad (1)$$

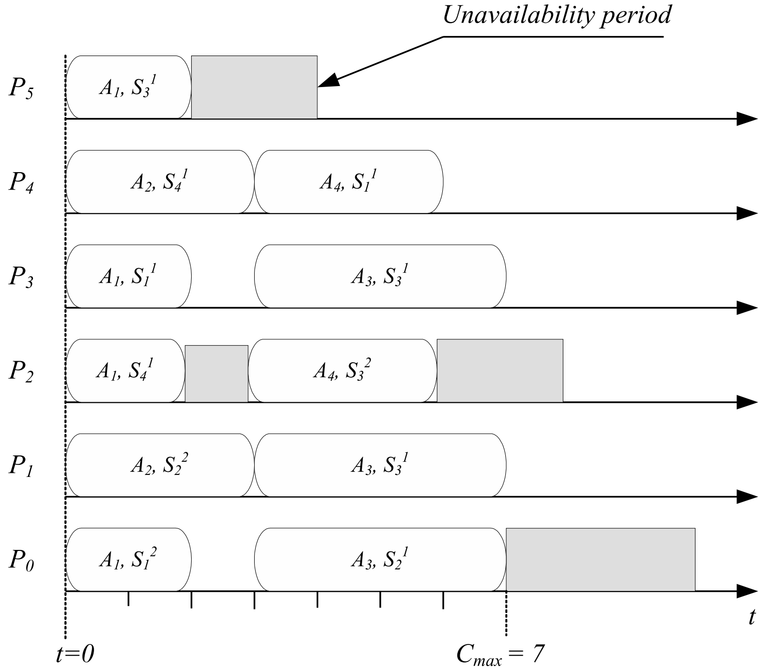


Fig. 2. One feasible solution for the example

$$\forall i \in \{1, \ldots, n\}, \forall m \in \{1, \ldots, M\}, \sum_{t=0}^{T_{\max}} x_{i,m,t} \leq 1 \,, \tag{2}$$

$$\forall i \in \{1, \ldots, n\}, \forall m \in \{1, \ldots, M\}, \sum_{t=0}^{T_{\max}} x_{i,m,t} \cdot t \leq \frac{\sum_{h=0}^{M} \sum_{t=0}^{T_{\max}} x_{i,h,t} \cdot t}{\sum_{k=0}^{K} \sum_{l=0}^{L} b_{i,k}^{l}} \,, \tag{3}$$

$$\forall i \in \{1, \ldots, n\}, \forall m \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, K\},$$
$$\forall l \in \{1, \ldots, L\}, \delta_{i,m,k}^{l} \leq S_{m,k}^{l} \,, \tag{4}$$

$$\forall i \in \{1, \ldots, n\}, \sum_{m=0}^{M} \sum_{t=0}^{T_{\max}} x_{i,m,t} \cdot t = \sum_{k=0}^{K} \sum_{l=0}^{L} b_{i,k}^{l} \,, \tag{5}$$

$$\forall i \in \{1, \ldots, n\}, \sum_{m=0}^{M} \delta_{i,m,k}^{l} = b_{i,k}^{l} \,, \tag{6}$$

$$\forall m \in \{1, \ldots, M\}, \sum_{i=0}^{n} \sum_{d=t-p_i+1}^{t} x_{i,m,d} \leq 1 \,, \tag{7}$$

$$\forall i \in \{1, \ldots, n\}, \forall m \in \{1, \ldots, M\}, \sum_{t=0}^{T_{\max}} x_{i,m,t} = \sum_{k=0}^{K} \sum_{l=0}^{L} \delta_{i,m,k}^{l} \,, \tag{8}$$

$$\min C_{\max} = \frac{\sum_{t=0}^{T_{\max}} \sum_{m=0}^{M} x_{n,m,t} \cdot t}{\sum_{k=0}^{K} \sum_{l=0}^{L} b_{n,k}^{l} + p_n} .$$

Equation (1) ensures that two activities that respect the precedence relation do not overlap. Equation (2) allows staff members to be assigned to an activity only once. Equation (3) obliges all the staff members assigned to a common activity to start at the same time. According to Equation (4), a person cannot be assigned to a need if he/she does not master the level of the skill needed. Equation (5) ensures that the number of persons that do an activity is equal to the sum of the need of this activity. Equation (6) obliges the number of persons that are assigned to a level of a skill to be equal to the needs for this level of this skill, for each activity. Equation (7) ensures that a person will not start an activity during the whole processing time of an activity he/she is already assigned to. Finally, Equation (8) ensures that a person participates in all the activities he/she starts.

This model is a time-indexed one, and in the general case a simple relaxation of this kind of model does not provide good lower bounds as it has been demonstrated for RCPSP. It is necessary to use constraint programming and /or a cutting plane technique [4], [10] to have good bounds. But these methods are too time-consuming: our goal is to develop an efficient lower bound that may be used both to compute a global lower bound and that may be used at each node of a branch-and-bound method. Thus, we focus on other types of lower bound formulation.

2.4 Literature Review

The MSPSP is a kind of problem very close to the classical MM-RCPSP [24], [12], [13], [7] and can be considered as a particular case of it. Actually, in the multi-mode RCPSP, every activity has different possible modes of execution. One mode is defined by a given quantity of each resource and an associated processing time, and for each activity there exists a finite number of modes (less than ten in the classical instances). If we apply this model to our problem, a mode corresponds to a feasible subset of staff members that can be assigned to an activity according to the required skills, while the processing time remains the same for all modes.

To the best of our knowledge, most of the methods that exist for the MM-RCPSP are based on explicit enumeration of all possible modes (even for computing lower bounds the authors of [7] use variables $\xi_{i,m}$ that are equal to 1 if and only if A_i is processed in mode m). For the MSPSP the number of different modes, i.e. the number of different subsets of staff members that can be assigned to an activity, may become huge. For example if we want to solve an instance with three skills with only one level for each skill, and 10 employees, some activities may have more than 1,000 modes each. Then, according to the LP formualtion proposed by [7], the number of variables grows quickly and the problem becomes intractable even for small size instances.

The problem we want to solve can also be seen as a particular case of the multi-resource shop [9], where every activity has specific fixed needs for each type of resource, a type of resource being defined by a set of machines. The difficulty is that the sets of resources corresponding to each different type are not disjoint. In our problem the resources are always the employees and the set of resources is those who master the required level of the skill needed.

3 Lower Bounds

As mentioned previously, the lower bounds we are looking for must provide a good trade-off between their efficiency and their time-consumption because they will be used in a branch-and-bound method at each node of the search tree. We have adapted two such bounds that have been proposed for RCPSP. Notice that the two bounds that we have adapted, namely from [22] and [3], are two of the most efficient ones to be used in branch-and-bound methods, considering respectively instances with high ratio of disjunction between activities and instances with few disjunctions between activities (see [11] and [3] for further details).

Here we present two classical lower bounds for the RCPSP that we adapt for the MSPSP. Both of them are destructive, which means that we fix a value D we want to test and either we detect a contradiction, i.e. we prove that the project cannot end before D and then $D+1$ is a valid lower bound, or we cannot detect an infeasibility and then D is decreased. Practically, we use binary search between the value given by the critical path and a valid upper bound [6].

For these two methods it is necessary that each activity has a time window. So, first of all, we use the precedence graph to compute the release date r_i of each activity A_i, according to the release dates and the durations of all its predecessors. Setting $r_i = L(0, A_i)$ as the longest path from the source to the activity A_i, then the value D, which is a lower bound for the project duration we are testing, is propagated from the last activity to the first in order to get deadlines $\tilde{d}_i(D)$ for the activities. Here $\tilde{d}_i(D)$ is equal to $L(A_i, A_n) - p_i$, the longest path from an activity to the sink of the project. Notice that these two lower bounds are mainly based on $[r_i, \tilde{d}_i(D)]$, thus they can be applied even if generalized precedence relationships are considered.

Notice that we have adapted these two lower bounds to MSPSP because their efficiency on a large range of RCPSP instances has been proved.

3.1 Graph of Compatibility

The first lower bound that we have adapted from RCPSP was introduced by [22] and is based on the notion of a block. A block is a feasible subset of activities that can be processed simultaneously, i.e. violating neither the resource constraints nor the precedence constraints. In the MSPSP framework, the precedence constraints can be simply verified, and checking whether a subset J of activities does not violate the resource constraints if its activities are in progress at the same time can be done using a max-flow formulation (see Figure 5). Thus, the bound of Mingozzi can be adapted to the MSPSP.

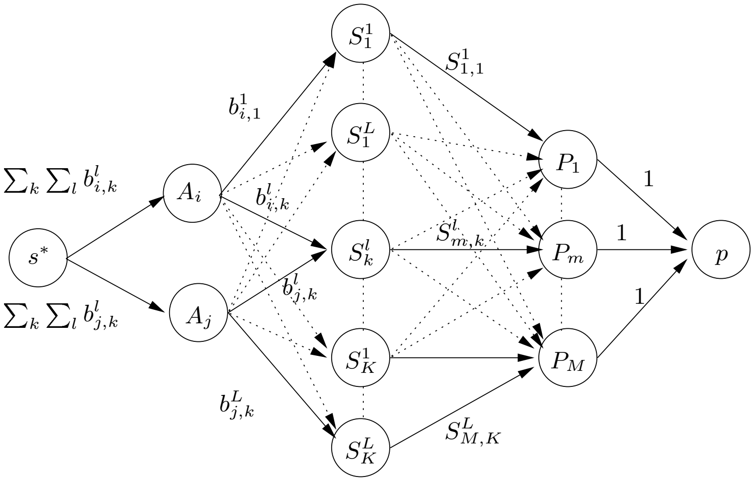


Fig. 3. Graph G_1 used for testing the needs of A_i and A_j

More precisely, we have adapted the third bound (LB_3) of [22] that is based on the use of disjunction between a couple of activities due to the resource constraint. For each couple of activities A_i and A_j (except the dummy activities), we test if it is possible for them to be in progress at the same time. First of all, we have to check that there is no precedence relationship between the two activities. Then we have to test if their time windows overlap, i.e. if there exists an intersection between $[r_i,\tilde{d}_i(D)]$ and $[r_j,\tilde{d}_j(D)]$, and finally we have to check the resource constraints. To find out if there are enough resources to schedule the two activities A_i and A_j during a common period, we solve the corresponding assignment problem using a max-flow formulation. The graph $G_1 = (X_1, F, c)$, $X_1 = \{A_i, A_j\} \bigcup \{S_k^l | \forall k \in \{0, \dots, K\}, \forall l \in \{0, \dots, L\}\} \bigcup \{P_m | \forall m \in \{0, \dots, M\}\}$ in which we search for a maximum flow is presented in Figure 3.

In this graph, the two nodes of the first column symbolize the activities A_i and A_j we are testing. The nodes of the second column represent each level of each skill S_k^l needed by the activities, and the third column of nodes corresponds to the staff members P_m that can be assigned to one of those activities. Edges have maximum capacity: for those from activity A_i to a level of a skill S_k^l this capacity is equal to the number of required staff members: $b_{i,k}^l$. From a staff member to the sink (p) this capacity is equal to 1 in order to limit a person to do one thing at a time. The edge between a level of a skill S_k^l and a person P_m exists if the person is able to do this level of this skill, i.e. if $S_{m,k}^l = 1$. We compute the maximum flow in order to compare it to the sum of the needs of the two activities ($\sum_{k=0}^K \sum_{l=0}^L (b_{i,k}^l + b_{j,k}^l)$) and conclude if the two activities can be in progress at the same time.

Property 1. If the maximum flow found in graph G_1 is strictly less than the sum of the needs of the two activities ($\sum_{k=0}^K \sum_{l=0}^L (b_{i,k}^l + b_{j,k}^l)$), there is a contradiction for the activities A_i and A_j to be in progress at the same time, and we can conclude that those two activities have to be scheduled in two time-intervals that do not overlap.

Proof. The proof is obvious due to the graph construction.

Once all those verifications have been done, we know exactly which couple of activities can be in progress at the same time, i.e. $(A_i,\, A_j) \in E'$. We can then build the graph of compatibility. Based on the precedence graph $G = (A, E, d)$ we define a graph $G' = (A, E')$, in which there is an edge between two activities (A_i, A_j) if no infeasibility has been detected and they can be scheduled in a common interval. We associate a weight with each node, equal to the duration p_i of the corresponding activity A_i. Then, we search for a maximum-weighted stable set in this graph, in order to determine the longest set of activities that cannot be in progress at the same time, which is a lower bound of the project duration.

But finding a maximum-weighted stable set in a graph is $\mathcal{NP}$-hard in the strong sense. This is the reason why we first try to solve it through a heuristic method as in [22]. This method is a greedy algorithm. First, we take all the nodes corresponding to the activities of the critical path, and then we add possible nodes one by one, in order of decreasing weights, until we cannot add any other node. A second way to compute the stable set is based on a MIP formulation solved with Cplex (1). Practical results show that this problem is well-solved. The model used is the following, where $x_i \in \{0, 1\}$, $x_i = 1$ if A_i is in the stable set:

$$\max \sum_{i=0}^{n} p_i \cdot x_i \qquad \text{s.t. } \forall (A_i, A_j) \in E' \; x_i + x_j \leq 1\,. \tag{9}$$

3.2 Energetic Reasoning Based Lower Bound

In [3] and [21] satisfiability tests and time-bound adjustments were introduced for the classical RCPSP. Satisfiability tests can notably be used to find if a given schedule can complete before a global deadline D and then can be used to compute a destructive lower bound. Energetic reasoning is based on the fact that in a given time interval $[t_1, t_2]$, we are able to detect if all the mandatory parts of the activities that have to be processed in this time interval can be done or not. The mandatory part of activity A_i that has to be scheduled between t_1 and t_2 is computed either by left-shifting or right-shifting the activity in its time window $[r_i, \tilde{d}_i(D)]$. Then we are sure that if there exists at least one time interval where those mandatory parts cannot be satisfied, i.e. there are not enough resources, then the lower bound can be increased to $D + 1$.

In order to use energetic reasoning, we compute all time intervals $[t_1, t_2]$, where $t_1 \in \{r_i, r_i + p_i, \tilde{d}_i(D) - p_i, \forall i \in \{1, \dots, n\}\}$ and $t_2 \in \{r_i + p_i, \tilde{d}_i(D) - p_i, \tilde{d}_i(D), \forall i \in \{1, \dots, n\}\}$, $t_1 < t_2$. All these time intervals are to be tested. This set of time points is a subset of those that have been proved to be relevant for bounding RCPSP [3]. They have not been proved to be relevant for the MSPSP, but the two problems are very close, so we assume that at least those time points

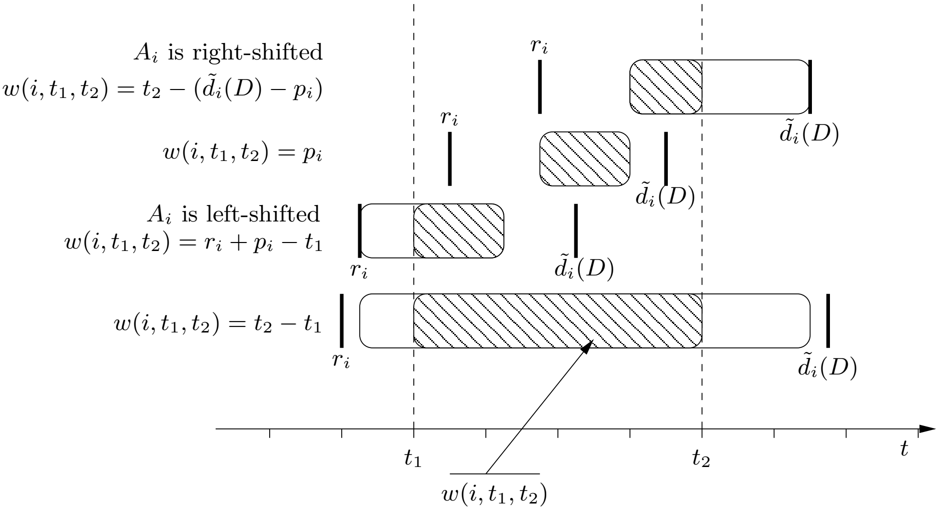


Fig. 4. Computation of the mandatory part of an activity

are interesting for our problem, and we cannot add all the possible time points because this lower bound is already as slow as acceptable, as is proved in Section 4.2.

To test the interval $[t_1, t_2]$, we have to compute the mandatory part of each activity in this interval. The mandatory part of an activity is the minimum part of this activity we have to schedule in this interval if we do not want to violate the time window of the activity. The mandatory part of A_i between t_1 and t_2, $w(i, t_1, t_2)$, is (see Figure 4)

$$w(i,t_1,t_2) = \min(\max(0, r_i + p_i - t_1), \max(0, t_2 - (\tilde{d}_i(D) - p_i)), p_i, t_2 - t_1) .$$

Once all the mandatory parts are computed we check if there are enough available resources in this interval to execute at least all these mandatory parts. As above, this problem can be modelled as an assignment problem that can be solved using a max-flow formulation. To do this, we use the graph $G_2(t_1, t_2, D)$ presented in Figure 5, where we search for a maximum flow in order to verify property 2. This graph is made of the first column of nodes that represent each level of each skill S_k^l and of the second column of nodes that corresponds to the staff members. Each edge from s to a level of a skill S_k^l has a maximum capacity equal to the mandatory parts times the number of persons needed for this level l of this skill $(\sum_{i=0}^{n} \sum_{k=0}^{K} \sum_{l=0}^{L} (w(i, t_1, t_2) \cdot b_{i,k}^l))$. Edges between levels of skills S_k^l and staff members P_m exist if the staff member masters this level of the skill, i.e. $S_{m,k}^l = 1$. These edges have the maximum capacity equal to the length of the time interval $[t_1, t_2]$. Finally, the edge between person P_m and p has the maximum capacity equal to the total time this person is available between t_1 and t_2, which is equal to $A(P_m, t_1, t_2)$.

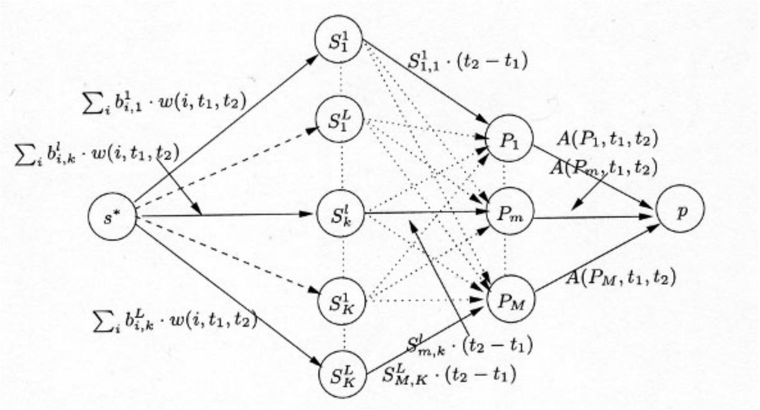

S_1^1
S_1^L
S_k^l
S_K^1
S_K^L
s^*
P_1
P_m
P_M
p
$\sum_i b_{i,1}^1 \cdot w(i,t_1,t_2)$
$\sum_i b_{i,k}^l \cdot w(i,t_1,t_2)$
$\sum_i b_{i,k}^L \cdot w(i,t_1,t_2)$
$S_{1,1}^1 \cdot (t_2 - t_1)$
$S_{m,k}^l \cdot (t_2 - t_1)$
$S_{M,K}^L \cdot (t_2 - t_1)$
$A(P_1,t_1,t_2)$
$A(P_m,t_1,t_2)$
$A(P_M,t_1,t_2)$

Fig. 5. Solving the assignment problem for energetic lower bound

Property 2. If there is at least a time-interval $[t_1, t_2]$ where the maximum flow in $G(t_1, t_2, D)$ is strictly less than the sum of the mandatory parts times the number of persons needed $(\sum_{i=0}^{n} \sum_{k=0}^{K} \sum_{l=0}^{L} (w(i, t_1, t_2) \cdot b_{i,k}^l))$, then D+1 is a valid lower bound.

Proof. The proof is obvious due to the graph construction.

4 Experimental Results

4.1 Instances Generation

Since there are no standard benchmark instances for the problem this paper deals with, we have generated some instances to test our methods. In fact, the problem is really close to other project management problems like the MM-RCPSP and multi-resource shop, but to validate our lower bounds we have to take instances where the number of "modes" or possible subsets of persons are much more numerous. Thus, we decided to keep the precedence graph from the PSPlib instances [19], [20], and build the data sets on those graphs. We have taken 180 instances from the single mode RCPSP, with 30, 60 and 90 activities, and a network complexity equal to 1.5, 1.8 or 2.1. Then for each instance, we have randomly generated between three and six different skills with three levels of ability. Then we have generated a number of persons between five and 30, and defined their skills in order to have at least one feasible solution. The instances we have generated in this way represent a wide range of instances, according to the *disjunctive ratio on precedence* and *disjunctive ratio on resource* defined in [3]. Some of them are strongly constrained so there are very few blocks of activities that can be scheduled in parallel, and other have many activities that can be in progress at the same time.

Table 4. Deviation between the two lower bounds and the upper bound given by the tabu search

No. of	No. of	LB with energetic reasoning (%)		LB with graph of compatibility (%)		Best of the two LB (%)	
activities	instances	Av.	Max.	Av.	Max.	Av.	Max.
30	60	3.06	16.66	2.95	13.63	2.53	13.636
60	60	8.03	20.58	11.34	28.2	7.35	18.6
90	60	11.47	26.47	22.34	47.17	11.17	25
Total	180	7.52	26.47	12.21	47.17	7.02	25

Table 5. Computational time of the lower bounds

No. of activities	No. of instances	LB with energetic reasoning (s)	LB with graph of compatibility (s)
30	60	0.938	0.012
60	60	3.013	0.04
90	60	5.99	0.15
Total	180	3.31	0.07

4.2 Results

Using 180 generated data sets, we have obtained the results presented in Table 4. In this table, the lower bounds are compared to the best known upper bound for each instance. These upper bounds have been obtained by a tabu search we have applied to the specific problem with hierarchical levels of skills, inspired by [6]. This tabu search has been adapted from the one for RCPSP [17]. Thus, in the first step, activities are sorted in a list, according to a classical priority rule (Minimum Slack Time, Latest Starting Time,. . .), and we apply a dispatching rule adapted from the Serial Schedule Scheme to define a solution. Then a neighbour is defined by swapping two activities in the priority list, respecting precedence constraints. Moves that do not modify the current solution in terms of starting times of activities are not allowed. All generated solutions are stored using a hashtable, with the index of the iteration in which the solution has been explored. These solutions are considered to be tabu during a given number of iterations, in order to avoid cycling.

The deviation presented in Tables 4 and 5 is equal to: $\frac{upper\ bound - lower\ bound}{upper\ bound}$. In these tables we do not show the results given by the lower bound based on the graph of compatibility when the stable set is computed by the greedy algorithm because this method is not efficient: it is clearly dominated by the two other lower bounds. Notice that the minimum deviation between LB and UB is not reported because it is always equal to zero.

These results clearly show that the lower bound given by energetic reasoning is better than the lower bound given by the graph of compatibility for every instance. However, Table 5 shows that this lower bound is much more time consuming. Although the average deviation is more important for the method based on the graph of compatibility, there exist some instances where this lower bound is more efficient than the other one. This is why the third column has been included in Table 4 in order to take the best one from the two lower bounds. It appears that the deviation we get considering the best from the two lower bounds is better than the one obtained by the energetic reasoning-based lower bound. So, the two lower bounds appear to be complementary. This is due to the fact that they do not consider the same aspect of the problem: the graph of compatibility first considers the activities two by two, and checks if there is a resource conflict between them in the whole time horizon, whereas the energetic lower bound checks a set of activities but only for restricted time intervals.

Notice that even if the lower bounds seem very time consuming, they will be used in this way, i.e. included into binary search on D, only at the root node of the search tree. In every other node of a branch-and-bound method, it will not be useful to search for lower bound by binary search, it is only necessary to check if the current value of the best upper bound minus one is feasible or not. If this value cannot be respected, that means that at least one of the two lower bounds detects infeasibility, so the node is pruned. Thus, the time needed at each node will be drastically reduced.

In order to find out if the lower bound limitations are particularly linked to one feature (or more) of the instances, as it appears for the RCPSP [3], we have tried to classify our instances and find a link between the average deviation and the *ratio of resource disjunction* between activities, the *ratio of precedence disjunction* between activities or the average number of couple of activities that have no contradiction to be assigned in parallel. No conclusion can be drawn because none of these features appears to be directly influential.

Finally, the average deviation increases dramatically when the size of the instances grows, but this gap may be due to the quality of the upper bound used. Actually, the tabu search we used is not guaranteed to be really efficient for the kind of instances with hierarchical levels of skill we consider, and the gap with the optimum is unknown. On some instances we have generated for the MSPSP without hierarchical levels of skills, i.e. certainly less difficult, the average deviation between the best lower bounds and the upper bound is around 4%.

5 Conclusion

This paper deals with the MSPSP, which is an extension of the RCPSP. In this particular scheduling problem, the resources are staff members with specific skills and levels of skills that allow them to be assigned to different kind of activities of the project. Each activity has a specific need for each level of each skill.

We have proposed a model for this problem, and introduced two lower bounds, adapted from lower bounds known for the RCPSP. The results show that the lower bounds are complementary and efficient.

The research direction we focus on now is to design an exact method in order to compute optimal solutions for small and medium size instances. This method will be used to determine whether the gap between lower bounds and upper bound is due to the upper bound or not. Our two lower bounds will be used in the branch-and-bound method. This exact method is already in progress [5].

References

1. Alvarez-Valdes, R., Martin, G., Tamarit, J. M.: Constructing Good Solutions for a School Timetabling Problem. J. Oper. Res. Soc. **47** (1996) 1203–1215
2. Bailey, J.: Optimization and Heuristic Models to Integrate Project Task and Manpower scheduling. Comput. Indust. Eng. **29** (1995) 473–476
3. Baptiste, P., Le Pape, C., Nuijten, W.: Satisfiability Tests and Time Bound Adjustments for Cumulative Scheduling Problems. Ann. Oper. Res. **92** (1999) 305–333
4. Baptiste, P., Demassey, S.: Tight LP Bounds for Resource Constrained Project Scheduling. OR Spectrum **26** (2004) 251–262
5. Bellenguez, O., Néron, E.: An Exact Method for Solving the Multi-skill Project Scheduling Problem. Operations Research Conference, Tilburg (2004) 97–98
6. Bellenguez, O., Néron, E.: Methods for Solving the Multi-skill Project Scheduling Problem. 9th Int. Workshop on Project Management and Scheduling, Nancy (2004) 66–69
7. Brucker, P., Knust, S.: Lower Bounds for Resource-Constrained Project Scheduling Problems. Eur. J. Oper. Res. **149** (2003) 302–313
8. Burke, E. K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The State of the Art of Nurse Rostering. J. Scheduling **7** (2004) 441–499
9. Dauzère-Pérès, S., Roux, J., Lasserre, J. B.: Multi-resource Shop Scheduling with Resource Flexibility. Eur. J. Oper. Res. **107** (1998) 289–305
10. Demassey, S., Artigues, C., Michelon, P.: Constraint-Propagation-Based Cutting Planes: An Application to the Resource-Constrained Project-Scheduling Problem. INFORMS J. Comput. (2003) In press
11. Demeulemeester, E., Herroelen, W.: New Benchmark Results for the Resource-Constrained Project Scheduling Problem. Manage. Sci. **43** (1997) 1485–1492
12. De Reyck, B., Herroelen, W.: The Multi-mode Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations. Eur. J. Oper. Res. **119** (1999) 538–556
13. Hartmann, S., Drexl, A.: Project Scheduling with Multiple Modes: A Comparison of Exact Algorithms. Networks **32** (1998) 283–297
14. Haase, K., Latteier, J., Schirmer, A.: The Course Scheduling Problem at the Lufthansa Technical Training. Eur. J. Oper. Res. **110** (1998) 441–456
15. Hanne, T., Nickel, S.: A Multiobjective Evolutionary Algorithm for Scheduling and Inspection Planning in Software Development Projects. Eur. J. Oper. Res. **167** (2005) 663–678
16. Josefowska, J., Mika, M., Rozycki, R., Waligora, G., Weglarz, J.: Simulated Annealing for Multi-mode Resource-Constrained Project Scheduling Problem. Ann. Oper. Res. **102** (2001) 137–155

17. Klein, R.: Project Scheduling with Time-Varying Resource Constraints. Int. J. Prod. Res. **38** (2000) 3937–3952
18. Kolisch, R., Drexl, A.: Local Search for Non-preemptive Multi-mode Resource-Constrained Project Scheduling Problem. IIE Trans. **29** (1997) 987–999
19. Kolisch, R., Sprecher, A.: PSPLIB—A Project Scheduling Problem Library. Eur. J. Oper. Res. 96 (1997) 205–216
20. Kolisch, R., Sprecher, A.: PSPLIB—A Project Scheduling Problem Library. ftp://ftp.bwl.uni-kiel.de/pub/operations-research/psplib/html/indes.html (2000)
21. Lopez, P., Erschler, J., Esquirol, P.: Ordonnancement de Tâches sous Contraintes: Une Approche Énergétique. RAIRO-APII **26** (1992) 453–481
22. Mingozzi, A., Maniezzo, V., Riciardelli, S., Bianco, L.: An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. Manage. Sci. **44** (1998) 714–729
23. Özdamar L.: A Genetic Algorithm Approach to a General Category Project Scheduling Problem. IEEE Trans. on Systems, Man, and Cybernetics, forthcoming
24. Sprecher, A., Drexl, A.: Multi-mode Resource Constrained Project Scheduling Problem by a Simple, General and Powerful Sequencing Algorithm. Eur. J. Oper. Res. **107** (1998) 431–450
25. Toroslu, I.: Personnel Assignment Problem with Hierarchical Ordering Constraints. Comput. Indust. Eng. (2003) 493–510

Examination Timetabling

A Novel Similarity Measure for Heuristic Selection in Examination Timetabling

Yong Yang and Sanja Petrovic

School of Computer Science and Information Technology,
The University of Nottingham, NG8 1BB, UK
{yxy, sxp}@cs.nott.ac.uk

Abstract. Metaheuristic approaches to examination timetabling problems are usually split into two phases: an initialisation phase in which a sequential graph colouring heuristic is employed to construct an initial solution and an improvement phase in which the initial solution is gradually improved. Different hybridisations of metaheuristics with sequential heuristics are known to lead to solutions of different quality. A Case Based Reasoning (CBR) methodology has been developed for selecting an appropriate sequential construction heuristic for hybridisation with the Great Deluge metaheuristic. In this paper we propose a new similarity measure between two timetabling problems that is based on fuzzy sets. The experiments were performed on a number of real-world benchmark problems and the results were also compared with other state-of-the-art methods. The results obtained show the effectiveness of the developed CBR system.

1 Introduction

Examination timetabling is an important and difficult task for educational institutions since it requires expensive human and computer resources and has to be solved several times every year. Timetabling can be defined to be the problem of allocating a set of examinations over a limited number of time periods subject to constraints in such a way as to generate no conflicts between any two examinations. For example, no student should be required to attend two examinations at the same time and no student should have two examinations on the same day.

The timetabling problem can be represented as an undirected weighted graph where vertices represent examinations, while edges represent conflicts between examinations (i.e. an edge connects examinations with common students) [24]. To both vertices and edges weights are assigned that correspond to the number of students enrolled in the examinations and the number of students enrolled in two examinations that are in conflict, respectively. For illustration purposes, a simple timetabling problem (with four examinations) is shown in Figure 1. For example, the weight of Math is 30 because 30 students are enrolled in this course. The edge connecting AI and PA1 is assigned weight 9 because there are nine students who are enrolled in both examinations. The timetabling problem is closely linked to the graph colouring problem [24], which is concerned with

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 247–269, 2005.

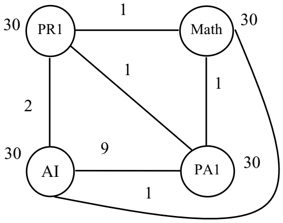


Fig. 1. A simple example of examination timetabling problem

the colouring of the vertices in such a way that no two adjacent vertices are coloured by the same colour. In the context of examination timetabling, colours correspond to time periods. In Figure 1, it can be seen that at least four different time periods are required to solve the problem since no two examinations which are in conflict with each other should be scheduled in the same time period.

Both the examination timetabling problem and the graph colouring problem are known to be NP-complete [29]. However, the examination timetabling problem has an additional wide variety of hard and soft constraints [24]. Hard constraints are those that must be completely satisfied. Solutions which do not violate hard constraints are called feasible solutions. Soft constraints are not essential to the feasibility of a timetable, but their satisfaction is highly desirable. In practice, the quality of an examination timetable is evaluated by some measure of satisfaction of soft constraints since it is usually impossible to fully satisfy all of them. A thorough review of the variety of constraints imposed on examination timetabling can be seen in [4].

1.1 Heuristics for Examination Timetabling

The complexity and the large size of the real-life university examination timetabling problems required development of different heuristics which were employed with reasonable success for their solving over the last 40 years [12], [18], [38]. Early research was focused on sequential heuristics for solving graph colouring problems [7], [20], [21], [46]. The main idea of these heuristics is to schedule examinations one by one, starting from the examination which is evaluated as the most "difficult" for scheduling. Different heuristics measure the "difficulty" of each examination in different ways. The drawback of these heuristics is that they have different performance on varied problem instances [23].

In recent years, there has been an increased interest in application of metaheuristics to examination timetabling problem solving because these techniques can take into consideration soft constraints and are usually able to generate more satisfactory solutions than sequential heuristics alone can do. In practice, timetabling problems are usually solved by a two-phase approach that consists of initialisation and improvement phases. In the first phase, an initial solution is iteratively constructed by using an appropriate sequential heuristic. The improvement phase gradually improves the initial solution by using a metaheuristic such as Simulated Annealing [31], [35], [45], Memetic Algorithm [8], [13], [15], GRASP [25] and Tabu Search [27], [47]. However, the performance of some metaheuristics is known to be highly dependent on the parameter values [6]. For example, it is well known that the settings of the cooling parameters have a great importance to the successful application of Simulated Annealing [45]. Furthermore, the performance of many approaches may vary from one problem instance to another, because they were developed specifically for solving one particular class of real-world problems [2].

In practice, a timetable administrator needs to make a great effort to select an appropriate (successful) sequential heuristic for hybridisation with a metaheuristic, and then needs to "tailor" the chosen heuristics by utilising the domain-specific knowledge to obtain a preferred solution for a given problem. Recently, the development of more general timetabling approaches that are capable of solving a variety of problems with different characteristics equally well, has attracted the attention of the timetabling community. In particular, the research into hyper-heuristics for examination timetabling gave promising results. Hyper-heuristics is defined as "the process of using (meta-)heuristics to choose (meta-)heuristics to solve the problem in hand" [5]. Terashima-Marín et al. [44] introduced an evolutionary approach as a hyper-heuristic for solving examination timetabling problems. In their approach, a list of different sequential heuristics, parameter value settings, and the conditions for swapping sequential heuristics are encoded as chromosomes. The timetable is built by using the best chromosome found by a genetic algorithm. Burke et al. [6] proposed a hyper-heuristic for timetabling problems in which the selection of heuristics is controlled by a tabu search algorithm. Tabu search approaches were employed within the hyper-heuristic framework that searched for different permutations of graph heuristics for solving both exam and course timetabling problems [3], [10].

1.2 Case-Based Heuristic Selection

Case-based reasoning (CBR) [32] is an artificial intelligence methodology which is an effective alternative to traditional rule-based systems. It is in particular useful for generating intelligent decisions in weak-theory application domains [26]. CBR stems from the observation that similar problems will have similar solutions [34]. Rather than defining a set of "IF THEN" rules or general guidelines, a CBR system solves a new problem by reusing previous problem solving experience, stored as cases in the case base. In CBR, a new input problem is usually solved by four steps: retrieve a case that is the most similar to the new problem, reuse and revise the solution of the retrieved case to generate a solution for the new problem, and retain the new input problem and its solution as a new case in the case base.

In the domain of scheduling, there have been some attempts to resort to CBR for achieving the intelligent heuristic selection so that the flexibility and robustness of scheduling is enhanced. Current CBR systems for heuristic selection fall into two categories: algorithm reuse and operator reuse. The basic underlying assumption of the CBR systems in the first category is that it is likely that an approach proved to be effective for solving a specific problem will be also effective for solving a similar problem. In these CBR systems, a case contains a problem representation and an algorithm known to be effective for its solving. Schmidt [43] proposed a CBR framework to choose an appropriate algorithm for a given production scheduling problem. Schirmer [42] designed a similar CBR system for solving project scheduling problems and showed that the CBR system outperformed a number of metaheuristics. A case-based reasoning system was developed by Burke et al. for solving university course timetabling problems [9], [11].

The CBR scheduling systems in the second category iteratively reuse the operators for solving a new input problem. A case in these systems describes a context in which a previously used scheduling operator proved to be successful. Miyashita and Sycara [36] built a CBR system called CABINS for solving job scheduling problems in which sub-optimal solutions were improved by iteratively employing a number of move operators, selected by CBR. Petrovic, Beddoe, and Berghe [37] developed a CBR system for nurse rostering problems in which the constraint satisfaction procedure was driven by iterative application of the scheduling repair operators employed in previously encountered similar situations. Burke, Petrovic, and Qu [19] proposed a novel case based hyper-heuristic for solving timetabling problems. A timetable was iteratively constructed by using a number of heuristics, which were selected by a CBR controller.

In general, the CBR systems' effectiveness depends on the proper definition of the similarity measure, because it determines which case will be used for solving a new input problem. In the current CBR scheduling systems for heuristic selection, cases are usually represented by the sets of attribute-value pairs, while the similarity between two cases is calculated as the distance between their attribute sets. The attributes and their weights can be set either empirically [36], [37], [42] or by employing knowledge discovery methods [19].

The objective of our research is to raise the level of generality of metaheuristic approaches to examination timetabling problems. A CBR system [40], [41] based on algorithm reuse was developed which produced high quality solutions for a range of different examination timetabling problems. The CBR system selected an appropriate sequential heuristic for the initialisation of the Great Deluge algorithm (GDA) [28]. GDA has been chosen due to its simplicity of use in terms of required parameters and high-quality results that it produced for examination timetabling problems. It has been shown that sequential heuristic selected for the initialisation phase had a great impact on the quality of the final solution [41]. In addition, a sequential heuristic which provided a "good" starting point for the GDA search in solving a particular timetabling problem, was proved to be good for the GDA initialisation in solving a similar timetabling problem.

Our research is focused on the application of sequential heuristics in the initialisation phase of the GDA. In the CBR system developed, a case consists of a description of an examination timetabling problem and the sequential heuristic that was used to construct a good initial solution for the GDA applied to the problem. The selection of the sequential heuristic for a new input problem comprises the following steps. The similarity between the new input problem and each problem stored in the case base is calculated. A case which is the most similar to the new input problem is retrieved, and the associated sequential heuristic of the retrieved case is used for the GDA initialisation for a new input problem.

In this paper, we discuss different representations of timetabling problems and corresponding similarity measures. The first representation takes into consideration the number of students involved in examinations and uses weighted graph representation of the timetabling problem [40]. The second representation does not consider number of students and uses unweighted graph representation [41]. We propose a new similarity measure based on weighted graph representation, which instead of using crisp number of students involved in the conflicts uses linguistic terms (*Low*, *Medium*, *High*) to evaluate the importance of conflicts between two examinations. Fuzzy sets are used to model these linguistic terms.

The paper is organised as follows. Section 2 provides a brief introduction to GDA and different sequential heuristics that are used for the initialisation phase. Section 3 describes briefly two different similarity measures based on the weighted and unweighted graph representation of timetabling problems, and introduces a new fuzzy similarity measure. Section 4 briefly introduces the retrieval process in our CBR system. Section 5 presents a series of experiments on benchmark problems that were carried out to evaluate the performance of the new CBR system. The final conclusions are presented in Section 6.

2 Great Deluge Algorithm and Sequential Heuristics

Great Deluge Algorithm (GDA) is a local search method proposed by Dueck [28]. Compared to the well known Simulated Annealing approach, GDA uses a simpler acceptance rule for dealing with the move that leads to a decrease in the solution quality. Such a worse intermediate solution can be accepted if the value of the objective function of the solution is smaller than a given upper boundary value, referred to as “water-level”. Water-level is initially set to be the penalty of the initial solution multiplied by a predefined factor. After each move, the value of the water-level is decreased by a fixed rate, which is computed based on the time that is allocated for the search (expressed as the total number of moves). One important characteristic of the GDA is that better solutions could be obtained with the prolongation of the search time of the algorithm [1]. This may not be valid in other local search algorithms in which the search time cannot be controlled. Burke et al. developed a GDA algorithm for examination timetabling [1]. The authors proposed to use the total number of moves, which expresses the computational time that the user is willing to spend, in the calculation of water level.

A variety of sequential heuristics can be used to construct initial solutions for the GDA. Five different heuristics are used in this research:

1. Largest Degree, which schedules examinations with the largest number of conflicts first,
2. Largest Enrolment, which priorities for scheduling examinations with the largest student enrolment,
3. Largest Colour Degree, which prioritises examinations with the largest number of conflicts that they have with already scheduled examinations,
4. Largest Weighted Degree, which estimates the difficulty of scheduling of each examination by the weighted conflicts, where each conflict is weighted by the number of students who are enrolled in both examinations, and
5. Least Saturation Degree, which schedules examinations with the least number of available periods for placement first.

They can be further hybridised with Maximum Clique Detection [30], Backtracking [33], and Adding Random Elements [17]. In total, 40 different sequential heuristics are investigated. The details of these heuristics are given in [41].

3 Similarity Measures for Examination Timetabling Problems

A properly defined similarity measure has a great impact on the CBR system. On the other hand, similarity measure is tightly connected with the representation of the cases. In this section we will briefly introduce two different similarity measures between examination timetabling problems based on different graph representations, which we investigated in our previous research work. A new similarity measure will be introduced next, which addresses some deficiencies of the previous ones.

3.1 Similarity Measure Based on Weighted Graph Representation

A timetabling problem is represented by a undirected weighted graph $G = (V, E, \alpha, \beta)$, where V is the set of vertices that represent examinations, $E \subseteq V \times V$ is the finite set of edges that represent conflicts between examinations, $\alpha : V \mapsto \mathsf{N}^+$ assigns a positive integer weight to each vertex that corresponds to the number of students enrolled in the examination, $\beta : E \mapsto \mathsf{N}^+$ is an assignment of weight to each edge which corresponds to the number of students enrolled in two examinations that are in conflict. The similarity measure between a new input problem $G_q = (V_q, E_q, \alpha_q, \beta_q)$ and a problem stored in the case base $G_s = (V_s, E_s, \alpha_s, \beta_s)$ is based on the graph isomorphism, which is known to be a NP-complete problem. An isomorphism is presented by a vertex-to-vertex correspondence $f : V_q \rightarrow V_s$ which associates vertices in V_q with those in V_s. In our notation, vertices and edges of graph G_q are denoted by Latin letters, while those of graph G_s are denoted by Greek letters.

The similarity degree between two vertices, $a \in V_q$ and $\chi \in V_s$, determined by correspondence f is denoted by $DS_f(a, \chi)$ and calculated in the following way:

$$DS_f(a,\chi) = \begin{cases} \min(\alpha_q(a), \alpha_s(\chi)) & \text{if } f(a) = \chi \\ 0 & \text{otherwise}\,. \end{cases} \tag{1}$$

Similarly, $DS_f(x,\gamma)$ represents the similarity degree between two edges determined by correspondence f, where $x = (a,b) \in E_q$ and $\gamma = (\chi,\delta) \in E_s$ and is calculated as follows:

$$DS_f(x,\gamma) = \begin{cases} \min(\beta_q(x), \beta_s(\gamma)) & \text{if } f(a) = \chi \text{ and } f(b) = \delta \\ 0 & \text{otherwise}\,. \end{cases} \tag{2}$$

The label ϕ is used to denote an extraneous vertex or edge in a graph, which is not mapped by correspondence f. $DS_f(a,\phi)$, $DS_f(\phi,\chi)$, $DS_f((a,b),\phi)$ and $DS_f(\phi,(\chi,\delta))$ are set to be equal to 0.

Finally, the similarity degree $\mathrm{SIM1}_f(G_q,G_s)$ between the graphs G_q and G_s determined by correspondence f is calculated in the following way:

$$\mathrm{SIM1}_f(G_q,G_s) = \frac{F_v + F_e}{M_v + M_e} \tag{3}$$

where

$$F_v = \sum_{a\in V_q} \sum_{\chi \in V_s} DS_f(a,\chi) \tag{4}$$

$$F_e = \sum_{x\in E_q} \sum_{\gamma \in E_s} DS_f(x,\gamma) \tag{5}$$

$$M_v = \min\left(\sum_{a\in V_q} \alpha_q(a), \sum_{\chi\in V_s} \alpha_s(\chi) \right) \tag{6}$$

$$M_e = \min\left(\sum_{x\in E_q} \beta q(x), \sum_{\gamma\in E_s} \beta_s(\gamma) \right)\,. \tag{7}$$

Note that the value of $DS_f(G_q,G_s) \in [0,1]$ is subject to correspondence f. The task is to find a correspondence f that yields as high value of $DS_f(G_q,G_s)$ as possible.

The results obtained using the weighted graph representation and described similarity measure are given in [40] (the normalisation of $\mathrm{SIM1}_f(G_q,G_s)$, performed by dividing with $(M_v + M_e)$ is calculated here differently than in [40] due to the changes in the retrieval process which will be described in Section 4).

3.2 Similarity Measure Based on Unweighted Graph Representation

A timetabling problem is represented by a graph $G = (V,E)$. The numbers of students who are sitting examinations and are involved in examination conflicts are not taken into consideration.

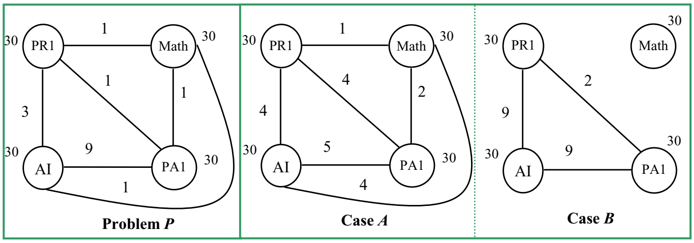


Fig. 2. New problem P and the case base with cases A and B

The similarity degree $DS_f(a,\chi)$ between two vertices in G_q and G_s determined by correspondence f is calculated in the following way:

$$DS_f(a,\chi) = \begin{cases} 1 & \text{if } f(a) = \chi \\ 0 & \text{otherwise}. \end{cases} \tag{8}$$

Similarly, the calculation of the similarity degree $DS_f(x,\gamma)$ between two edges determined by correspondence f, where $x = (a,b) \in E_q$ and $\gamma = (\chi,\delta) \in E_s$, is given by

$$DS_f(x,\gamma) = \begin{cases} 1 & \text{if } f(a) = \chi \text{ and } f(b) = \delta \\ 0 & \text{otherwise}. \end{cases} \tag{9}$$

In such a definition of similarity between two timetabling problems a mapped pair of vertices/edges in two graphs contributes to the similarity by a constant value 1 (independently from a number of students involved in the mapped vertices/edges). Finally, the similarity degree SIM2$_f(G_q, G_s)$ between G_q and G_s determined by correspondence f is calculated in the following way:

$$SIM2_f(G_q, G_s) = \frac{F_v + F_e}{M_v + M_e} \tag{10}$$

where

$$F_v = \sum_{a \in V_q} \sum_{\chi \in V_s} DS_f(a,\chi) \tag{11}$$

$$F_e = \sum_{x \in E_q} \sum_{\gamma \in E_s} DS_f(x,\gamma) \tag{12}$$

$$M_v = \min(|V_q|, |V_s|) \tag{13}$$

$$M_e = \min(|E_q|, |E_s|) \tag{14}$$

where $|V|$ and $|E|$ denote the cardinality of the sets V and E, respectively. Experimental results show that the similarity measure SIM2 on average outperforms SIM1 on benchmark problems established within university timetabling community [41].

Table 1. Similarity between timetabling problems P and A, B, by similarity measure SIM1

	P and A	P and B
F_v	$30 + 30 + 30 + 30 = 120$	$30 + 30 + 30 + 30 = 120$
F_e	$3 + 5 + 1 + 1 + 1 + 1 = 12$	$3 + 1 + 9 = 13$
M_v	$\min(120, 120) = 120$	$\min(120, 120) = 120$
M_e	$\min(16, 20) = 16$	$\min(16, 20) = 16$
SIM1$(P,^{*})$	$(120 + 12)/(120 + 16) = 0.97$	$(120 + 13)/(120 + 16) = 0.978$

3.3 Fuzzy Similarity Measure Based on Weighted Graph Representation

The similarity measure SIM1 is investigated further. In order to find a case in the case base that is similar to the new timetabling problem, i.e. to establish a "good" isomorphism between two graphs, two issues are considered. Firstly, it is necessary to find a "good" correspondence between vertices/edges of the new timetabling problem and the one stored in the case base. Secondly, weights of the vertices/edges should have equal or similar values. However, it was noticed that in some situations the similarity measure SIM1 will give priority to a graph with less similar structure to the new input problem but with the same (high) weights of the corresponding vertices/edges over a graph with more similar structure but different weights of the corresponding vertices/edges.

To illustrate this observation let us consider three timetabling problems whose structures are given in Figure 2: a new input problem P and problems A and B which are stored in the case base. Let us suppose that the established graph isomorphism(s) associates vertices in P and those in $A(B)$ that have the same examination names. The similarities between P and A and B are given in Table 1.

Similarity measure SIM1 evaluates case B to be more similar (although slightly) to new problem P than case A. Obviously, following the definition of similarity SIM1 the weights of the corresponding edges of P and B that are equal contribute more to the similarity than the corresponding edges of P and A which do not have the same weights. However, graph P has the same structure as graph A, but is structurally very different to graph B. These observations motivated the definition of the new similarity measure SIM3 to improve the effectiveness of the previously developed CBR system [40], [41]. This similarity measure does not consider vertex weights but only edge weights because they indicate the size of the conflict between the examinations. The corresponding edges will still contribute to the similarity between two graphs, but their contribution needs to be smaller than their weights. The procedure for calculation of the contribution of the edge weights to the similarity measure consists of two steps:

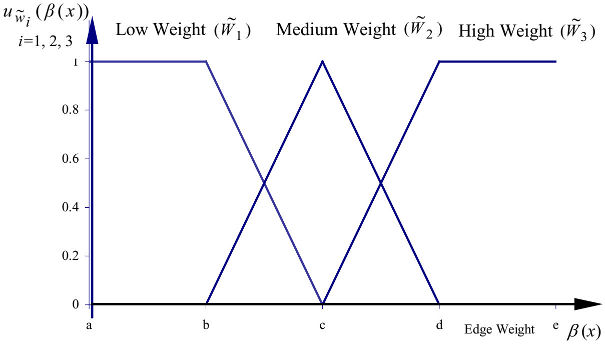


Fig. 3. Membership functions defined for fuzzy sets *Low Weight*, *Medium Weight*, *High Weight*

Step I The corresponding edges of the two graphs are classified to sets: *Low Weight*, *Medium Weight* and *High Weight*. In order to avoid a rigid definition of strict boundaries of these sets, fuzzy sets [48, 49] are used for their modelling. Unlike classical sets in which each object is either a member or not a member of a given set, a fuzzy set $\tilde{A}$ defined on a universe of discourse U is characterised by a membership function $u_{\tilde{A}}(x) \in [0,1]$ that assigns to each object $x \in U$ a degree of membership of x in $\tilde{A}$. The membership functions for three fuzzy sets *Low Weight* ($\tilde{W}_1$), *Medium Weight* ($\tilde{W}_2$) and *High Weight* ($\tilde{W}_3$) are given in Figure 3.

Parameters a, b, c, d, e are defined in the following way. Parameter a defines the lower bound of the set *Low Weight* and is set to be 1 (weight of edges are positive integers). Parameter b is calculated as the mean value of all edge weights in the graph:

$$b = \frac{\sum\limits_{x \in E} \beta(x)}{|E|} . \tag{15}$$

The assumption is that the edges whose weight is smaller than the mean weight have high degree of membership to *Low Weight*. Parameter e is set to be the maximum edge weight in the graph:

$$e = \max_{x \in E} \beta(x) . \tag{16}$$

Parameters c and d are set to divide the $[b, e]$ interval into equal sizes:

$$c = b + \frac{e-b}{3} = \frac{2b+e}{3} \tag{17}$$

$$d = b + 2\frac{e-b}{3} = \frac{b+2e}{3} . \tag{18}$$

The result of step I is the classification of the corresponding edge weights in the established graph isomorphism given by a triplet

$$\left(u_{\widetilde{low\_wt}}(\beta(x)), u_{\widetilde{med\_wt}}(\beta(x)), u_{\widetilde{high\_wt}}(\beta(x))\right)$$

which denotes a membership degree of edge x to three fuzzy sets: *Low Weight*, *Medium Weight* and *High Weight*.

Step II Based on the classification obtained in step I, the weight of the edge is assigned a real number W_x which determines its contribution to the similarity measure between two graphs. Experiments indicated that real number should not be greater than the average edge weight in the graph. It is calculated using the formula

$$W_x = \frac{\sum_{i=1}^{3} h_i u_{\tilde{w}_i}(\beta(x))}{\sum_{i=1}^{3} u_{\tilde{w}_i}(\beta(x))} \tag{19}$$

where h_1 is set to be 1; h_2 is set as mean of h_1 and h_3; h_3 is set as mean weight of all edges' weights of the graph of the new input timetabling problem.

The similarity degree between two vertices a and χ on correspondence f is defined as follows:

$$DS_f(a,\chi) = \begin{cases} 1 & \text{if } f(a) = \chi \\ 0 & \text{otherwise}\,. \end{cases} \tag{20}$$

The similarity degree between two edges x and γ, where $x = (a,\, b) \in E_q$ and $\gamma = (\chi,\, \delta) \in E_s$, on correspondence f is denoted by $DS_f(x,\, \gamma)$:

$$DS_f(x,\gamma) = \begin{cases} \min(W_x, W_\gamma) & \text{if } f(a) = \chi \text{ and } f(b) = \delta \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

where W_x and W_γ are the new edge weights for edges x and γ, respectively.

Similarity degree $\mathrm{SIM3}_f(G_q, G_s)$ between two undirected weighted graphs G_q and G_s on correspondence f is calculated as

$$\mathrm{SIM3}_f(G_q, G_s) = \frac{F_v + F_e}{M_v + M_e} \tag{22}$$

where

$$F_v = \sum_{a \in V_q} \sum_{\chi \in V_s} DS_f(a,\chi) \tag{23}$$

$$F_e = \sum_{x \in E_q} \sum_{\gamma \in E_s} DS_f(x,\gamma) \tag{24}$$

$$M_v = (|V_q|, |V_s|) \tag{25}$$

$$M_e = \min\left(\sum_{x \in E_q} W(x), \sum_{\gamma \in E_s} W_\gamma\right) \tag{26}$$

where M_v and M_e are the maximum values that F_v and F_e can take, respectively.

The procedure for calculation of the similarity between case P and cases A and B from the case base is illustrated by an example given in Figure 2. The calculation of "new weights" of edges with which they will contribute to the similarity measure are given in Table 2, while Table 3 presents the calculation of new similarities between cases P and A and B. According to this new similarity measure, case A is more similar to case P than case B.

4 Retrieval Process

A case base may contain a large number of cases. The retrieval process of the CBR system has to establish a graph isomorphism between a new problem and all cases in the case base. In order to enable the faster retrieval a filtering phase is introduced which retrieves the subset of cases from the case base using a set of features, that we refer to as *shallow properties*. They reflect the size and the complexity of the problem: f_1, the number of examinations; f_2, the number of enrolments; f_3, the number of time periods available; f_4, the density of the conflict matrix (calculated as the ratio of the number of examinations in conflict to the square of the total number of examinations).

The nearest neighbour is used to calculate the similarity degree of two cases based on the shallow properties, represented by feature sets F_q and F_s:

$$\mathrm{SIM}_{\mathrm{shallow}}(F_q, F_s) = 1 - \sqrt{\frac{1}{n}\sum_{i=1}^{n} \mathrm{distance}\,(f_{q_i}, f_{s_i})^2} \tag{27}$$

where n is the number of features, f_{q_i} and f_{s_i} are the values of the ith feature in F_q and F_s, respectively, and the *distance* between two feature values f_{q_i} and f_{s_i} is computed as

$$\mathrm{distance}(f_{q_i}, f_{s_i}) = \left| \frac{f_{q_i} - f_{s_i}}{f_{\mathrm{max}_i} - f_{\mathrm{min}_i}} \right| \tag{28}$$

where f_{max_i} and f_{min_i} are the maximum and minimum values of the ith feature recorded in the case base.

The cases whose similarity with the new problem is greater than the predefined threshold (empirically set to be 0.6) are passed to the Tabu Search algorithm [39] which searches for the best graph isomorphism SIM1 in terms of defined similarity measures (SIM1, SIM2 or SIM3) between the new problem and the retrieved subset of cases. Finally, the general similarity measure is calculated between the new problem C_q and a case C_s from the subset of cases, using the formula

$$\mathrm{SIM}(C_q, C_s) = \mathrm{SIM}_{\mathrm{shallow}}(F_q, F_s) \cdot \mathrm{SIM}_f(G_q, G_s)\,. \tag{29}$$

Table 2. Calculation of “new weights” in graphs P, A and B

	Problem P			Case A			Case B			$DS_f(x,\gamma)$	$DS_f(x,\gamma)$
Edge	Weight	Step I	Step II	Weight	Step I	Step II	Weight	Step I	Step II	$x \in E_P$, $\gamma \in E_A$	$x \in E_P$, $\gamma \in E_B$
(PR1, AI)	3	(0.72, 0.28, 0)	1.23	4	(0, 0.8, 0.2)	2	9	(1, 0, 0)	2.67	1.23	1.23
(PR1, Math)	1	(1, 0, 0)	1	1	(1, 0, 0)	1	–	–	–	1	0
(PR1, PA1)	1	(1, 0, 0)	1	4	(1, 0, 0)	2	2	(0, 0, 1)	1	1	1
(PA1, Math)	1	(1, 0, 0)	1	2	(1, 0, 0)	1	–	–	–	1	0
(PA1, AI)	9	(0, 0, 1)	2.67	5	(0, 0, 1)	2.67	9	(0, 0, 1)	2.67	2.67	2.67
(AI, Math)	1	(1, 0, 0)	1	4	(1, 0, 0)	2	–	–	–	1	0
Sum	16		7.9	20		10.67	20		6.34	$F_e = 7.9$	$F_e = 4.9$

Table 3. Similarity between timetabling problem P and A, B, by the new similarity measure SIM3

Similarity	F_v	M_v	F_e	M_e	SIM3 (P, *)
P and A	4	4	7.9	$\min(7.9, 10.67) = 7.9$	$(4+7.9)/(4+7.9) = 1.0$
P and B	4	4	4.9	$\min(7.9, 6.34) = 6.34$	$(4+4.9)/(4+6.34) = 0.86$

5 Experimental Results

The experiments were performed on a number of real-world examination problems from different universities that has been collected and used as benchmark problems. The objectives of the experiments are

- to compare different similarity measures;
- to investigate whether the new similarity measure can enable retrieval of the most effective sequential heuristics for the benchmark problems;
- to evaluate the new CBR system performance by comparing it with the other state-of-the-art approaches to examination timetabling.

ftp://ftp.mie.utoronto.ca/pub/carter/testprob/ gives the benchmark problems. Their characteristics are shown in Table 4.

The cost function of these problems takes into consideration the spread of student's examinations. The cost function was adopted in the research on university examination timetabling and enables comparison between different timetabling approaches. It can be described by the following formula [23]:

$$w_s = \frac{32}{2^s}, \quad s \in \{1, 2, 3, 4, 5\} \tag{30}$$

where w_s is the cost given to the solution whenever a student has to take in two examinations scheduled s periods apart from each other. Experiments were run on a PC with a 1400 MHz Athlon processor and 256 MB RAM.

5.1 Case Base Initialisation

In our experiments, the initial case base was seeded with a number of examination timetabling problems that were randomly generated (more details are given in [41]). Seeding problems differ in three parameters: the number of examinations (n), the number of students (s), and the density of the conflict matrix (d). Three seeding problems were created for each combination of these parameters, which are random variables with a normal distribution where mean of $n \in \{100, 200, 300, 400\}$, mean of $s \in \{10 \times n,\ 20 \times n\}$, and mean of $d \in \{0.07, 0.15, 0.23\}$. For each n, s and d, the proportion of the standard deviation and the mean was set as 0.05. Thus, 72 ($3 \times 4 \times 2 \times 3$) different seeding problems were generated for the case base.

Table 4. Examination timetabling benchmark problems

Data	Institution	Periods	No. of Exams	No. of Students	No. of Enrolments	Density of Conflict Matrix
car-f-92	Carleton University, Ottawa	32	543	18,419	55,522	0.14
car-s-91	Carleton University, Ottawa	35	682	16,925	56,877	0.13
ear-f-83	Earl Haig Collegiate Institute, Toronto	24	190	1,125	8,109	0.29
hec-s-92	Ecole des Hautes Etudes Commercials, Montreal	18	81	2,823	10,632	0.20
kfu-s-93	King Fahd University Dharan	20	461	5,349	25,113	0.06
lse-f-91	London School of Economics	18	381	2,726	10,918	0.06
rye-s-93	Ryerson University, Toronto	23	486	11,483	45,051	0.07
sta-f-83	St Andrew's Junior High School, Toronto	13	139	611	5,751	0.14
tre-s-92	Trent University, Peterborough, Ontario	23	261	4,360	14,901	0.18
uta-s-92	Science, University of Toronto	35	622	21,276	11,793	0.08
ute-s-92	Faculty of Engineering, University of Toronto	10	184	2,750	11,793	0.08
yor-f-83	York Mills Collegiate Institute, Toronto	21	181	941	6,034	0.27

In order to find the best initialisation heuristic for each seeding problem, the GDA initialised by each sequential heuristic was run for 5 times using 20×10^6 iterations (this value was set empirically), while the water-level was set to 1.3 (this value is taken from [14]). These values for the number of iterations and for the water-level will be employed in most of the experiments presented in this paper. Finally, three case bases were established: a small, a middle and a large case base with 24, 48 and 72 cases, respectively.

5.2 Evaluation of Similarity Measures

The purpose of this set of experiments is to evaluate the effectiveness of the proposed similarity measure SIM3. This new similarity measure is also compared with the similarity measures SIM1 and SIM2.

Having established three case bases and defined three different similarity measures, each combination of a case base and a similarity measure was employed to choose a sequential heuristic for each of the 12 benchmark problems. We adopted the method described in [41] to evaluate whether the retrieved sequential heuristic is effective for the benchmark problem. For each benchmark problem, the GDA was run five times initialised by each sequential heuristic. After that sequential heuristics were sorted in ascending order by the average final solution cost obtained. The rank of the sequential heuristic H for the problem P is denoted by $R(H, P)$.

The System Effectiveness Degree $SED(P)$ indicates the distance between the sequential heuristic used in the case retrieved from the case base denoted by H^{CB} and the heuristic H^{best} which is the best for the GDA initialisation for the benchmark problem P $(R(H^{\text{best}}, P) = 1)$. It is calculated as

$$SED(P) = 1 - \frac{R(H^{\text{CB}}, P) - 1}{N - 1} \tag{31}$$

where N is the total number of heuristics used for the GDA initialisation. A high value of SED indicates the high effectiveness of the retrieved sequential heuristic. For each combination of the case base and the similarity measure, the average $SED(P)$ values were computed for all benchmark problems and are shown in Figure 4.

It is evident that the SED values of SIM3 are higher than those of SIM1 and SIM2 for all three case bases. This result justifies the new fuzzy similarity measure. The experimental results also show that the growth of the size of the case base leads to the retrieval of more effective sequential heuristics.

5.3 System Performance on Benchmark Problems

The following set of experiments aims to investigate the effectiveness of our CBR system by comparing the obtained results with those of other approaches. The CBR system with the similarity measure SIM3 and the large case base were used to solve benchmark problems. In each experiment, our CBR system selected

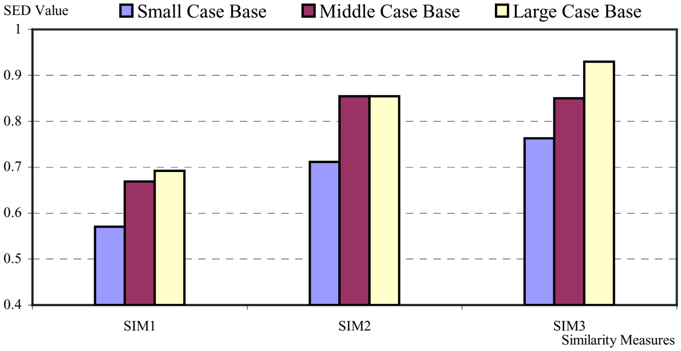


Fig. 4. Performance of different similarity measures

a sequential heuristic for a benchmark problem. The problem was solved by running the retrieved sequential heuristic and the GDA successively for 200×10^6 iterations, five times with varying random number seeds. System Effectiveness Degree *SED* is calculated for each retrieved sequential heuristic. Table 5 shows our results and the best results achieved by the exhaustive search across all heuristics.

It can be seen that CBR succeeded in suggesting the appropriate heuristics for the GDA initialisation and thus resulted in high-quality solutions. The new CBR initialisation was successful in finding the best heuristics for the benchmark problem lse-f-91, sta-f-83 and uta-s-92. For seven problem instances car-f-92, car-s-91, ear-f-83, hec-s-92, kfu-s-93, rye-f-92 and yor-f-83, the retrieved heuristics are among the four best ($0.923 \leq SED \leq 0.974$). It is important to note that the developed CBR initialisation took in average less than 10 minutes for each timetabling problem, while an exhaustive test needed more than six hours.

Table 6 shows the comparison of the average results generated by three other state-of-the-art approaches: the GDA where the initial solution was constructed by saturation degree (SD) [1], the GDA initialised by the adaptive heuristic [14], [16], the GDA where the saturation degree heuristic was applied with the maximum clique detection (MCD) and backtracking (BT) in the initialisation phase (this heuristic was suggested by Carter et al. [22] to be the best constructive heuristic). Each problem instance was solved five times. The time (in seconds) shown is the average time spent on the search. The GDA was also allocated the same number of iterations $200{*}10^6$ for each approach. In this experiment, we employed the higher number of iterations than in the previous ones in order to compare our results with the published ones. The times shown are different due to the use of computers of different characteristics.

For nine benchmark problems, our CBR system obtained best average results (highlighted by bold characters). For two problems, second best average results were obtained. Even more, for eleven benchmark problems the best value of the cost function was obtained as a result of appropriate GDA initialisation. The obtained results prove the significance of the appropriate initialisation of GDA.

Table 5. Comparison of results for benchmark problems obtained by the exhaustive search and CBR initialisation of the GDA

	Exhaustive test				CBR ($\|CB\| = 72$, SIM3)			
	Retrieval		Run GDA		Retrieval		Run GDA	
Data	*SED*	Time (s)	Cost	Time (s)	*SED*	Time (s)	Cost	Time (s)
car-f-92	1.00	35700	**3.97**	1080	0.923	491	3.99	1027
car-s-91	1.00	42739	**4.52**	1310	0.948	1733	4.53	1040
ear-f-83	1.00	15245	**34.78**	690	0.949	445	34.87	690
hec-s-92	1.00	20874	**11.32**	1490	0.923	73	11.36	1021
kfu-s-93	1.00	19643	**14.11**	689	0.974	1402	14.35	751
lse-f-91	1.00	15095	**10.78**	595	**1.00**	1170	**10.78**	559
rye-f-92	1.00	20123	**8.74**	862	0.974	683	8.79	699
sta-f-83	1.00	12368	**158.02**	676	**1.00**	91	**158.02**	649
tre-s-92	1.00	16495	**8.03**	730	0.744	972	8.10	844
uta-s-92	1.00	32094	**3.20**	1051	**1.00**	839	**3.20**	1051
ute-s-92	1.00	10755	**25.70**	557	0.769	172	26.10	574
yor-f-83	1.00	26723	**36.85**	1200	0.949	348	36.88	1243

Finally, we also compare our results with those produced by the state-of-the-art timetabling metaheuristics: Simulated Annealing (SA) [35], Tabu search [47], and GRASP [25]. The average of the scores for the twelve problem instances is shown in Table 7.

We can see that our CBR system outperformed other metaheuristics. Our CBR system obtained the best average results for seven benchmark problems and the second best average results for two benchmark problems. In addition, it is clear that the additional time on the case retrieval is required by our CBR system. However, the time spent on the selection of an appropriate sequential heuristic is justified by the quality of the results.

6 Conclusions

Different graph representation of examination timetabling problems and the corresponding similarity measures between two problems have been discussed. They are used in the CBR system for heuristic initialisation of GDA. The experimental results on a range of real world examination timetabling problems prove that the new fuzzy similarity measure based on weighted graph representation leads to the good selection of sequential heuristic for the GDA initialisation. By assigning linguistic terms to the edge weights of the timetabling graphs, the new similarity measure enables the retrieval of the timetabling problem from the case base which is structurally similar to the new problem.

We have also demonstrated that the CBR system with the new similarity measure can efficiently select a good heuristic for the GDA initialisation for

Table 6. Comparison of results for benchmark problems obtained by different initialisation of the GDA

	SD			Adaptive			SD & MCD & BT			CBR ($\|CB\| = 72$)			
Data	GDA Time	Best Cost	Avg. Cost	GDA Time	Best Cost	Avg. Cost	GDA Time	Best Cost	Avg. Cost	Retrieval Time	GDA Time	Best Cost	Avg. Cost
car-f-92	1120	4.03	4.07	416	–	4.10	1220	3.97	4.04	491	1027	**3.93**	**3.99**
car-s-91	1400	4.57	4.62	681	–	4.65	1441	4.62	4.66	1733	1040	**4.50**	**4.53**
ear-f-83	806	34.85	36.04	377	–	37.05	767	33.82	36.26	445	690	**33.71**	**34.87**
hec-s-92	1471	11.27	12.43	516	–	11.54	1411	11.08	11.48	73	1021	**10.83**	**11.36**
kfu-s-93	843	14.33	14.64	449	–	**13.90**	996	14.35	14.62	1402	751	**13.82**	14.35
lse-f-91	646	11.61	11.65	341	–	10.82	675	11.57	11.94	1170	559	**10.35**	**10.78**
rye-f-92	845	9.19	9.66	–	–	–	881	9.32	9.50	683	699	**8.53**	**8.79**
sta-f-83	675	165.12	169.7	418	–	168.73	674	166.07	166.31	91	649	**151.52**	**158.02**
tre-s-92	907	8.13	8.29	304	–	8.35	751	8.19	8.27	972	844	**7.92**	**8.10**
uta-s-92	1070	3.25	3.30	517	–	**3.20**	1101	3.24	3.31	839	1051	**3.14**	**3.20**
ute-s-92	716	25.88	26.05	324	–	**25.83**	653	25.53	26.02	172	574	**25.39**	26.10
yor-f-83	1381	36.17	**36.59**	695	–	37.28	1261	**36.31**	37.27	348	1243	36.53	36.88

Table 7. Comparison of results for benchmark problems obtained by different metaheuristics

	SA [35]			Tabu [47]			GRASP [25]			CBR ($\lvert CB\rvert = 72$)			
Data	Time	Best Cost	Avg. Cost	Time	Best Cost	Avg. Cost	Time	Best Cost	Avg. Cost	Retrieval Time	GDA Time	Best Cost	Avg. Cost
car-f-92	233	4.3	4.4	–	4.63	4.69	–	4.4	4.7	491	1027	**3.93**	**3.99**
car-s-91	296	5.1	5.2	–	5.73	5.82	–	5.4	5.6	1733	1040	**4.50**	**4.53**
ear-f-83	26	35.1	35.4	–	45.8	45.6	–	34.8	35.0	445	690	**33.71**	**34.87**
hec-s-92	5.4	**10.6**	**10.7**	–	12.9	13.4	–	10.8	10.9	73	1021	10.83	11.36
kfu-s-93	40	**13.5**	**14.0**	–	17.1	17.8	–	14.1	14.3	1402	751	13.82	14.35
lse-f-91	35	10.5	11.0	–	14.7	14.8	–	14.7	15.0	1170	559	**10.35**	**10.78**
rye-f-92	70	**8.4**	**8.7**	–	11.6	11.7	–	–	–	683	699	8.53	8.79
sta-f-83	5	157.3	157.4	–	158	158	–	**134.9**	**135.1**	91	649	151.52	158.02
tre-s-92	39	8.4	8.6	–	8.94	9.16	–	8.7	8.8	972	844	**7.92**	**8.10**
uta-s-92	233	3.5	3.6	–	4.44	4.49	–	–	–	839	1051	**3.14**	**3.20**
ute-s-92	9	**25.1**	**25.2**	–	29.0	29.1	–	25.4	25.5	172	574	25.39	26.10
yor-f-83	30	37.4	37.9	–	42.3	42.5	–	37.5	38.1	348	1243	**36.53**	**36.88**

most of the benchmark problems, and even more it outperforms the other state-of-the-art solution approaches based on GDA. This research makes a further contribution to the attempt of development of a general metaheuristic framework for timetabling, which is not tailored for a particular timetabling problem, i.e. works well on a range of different timetabling problems. We believe that this new similarity measure along with the proposed CBR methodology are also applicable to other domains such as personnel scheduling, job shop scheduling, and project scheduling.

References

1. Burke, E. K., Bykov, Y., Newall, J. P., Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Trans. Oper. Eng. **36** (2004) 509–528
2. Burke, E. K., Eckersley, A. J., McCollum, B., Petrovic, S., Qu, R.: Similarity Measures For Exam Timetabling Problems. In: Proc. 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (2003) 120–136
3. Burke, E. K., Dror, M., Petrovic, S., Qu, R.: Hybrid Graph Heuristics in a Hyper-Heuristic Approach to Exam Timetabling. In: Golden, B.L., Raghavan, S., Wasil, E. A. (eds.): The Next Wave in Computing, Optimization and Decision Technologies. Springer, Berlin (2005) 79–92
4. Burke, E. K., Elliman, D. G., Ford, P. H., Weare, R. F.: Examination Timetabling in British Universities—A Survey. In: Burke, E., Ross, P.: The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 76–92
5. Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-Heuristics: An Emerging Direction in Modern Search Technology. Chapter 16 in: Glover, F., Kochenberger, G. (eds.): Handbook of Meta-heuristics, Kluwer, Dordrecht (2003) 457–474
6. Burke, E. K., Kendall, G., Soubeiga, E.: A Tabu Search Hyper-heuristic for Timetabling and Rostering. J. Heuristics **9** (2003) 451–470
7. Burke, E., Kingston, J., De Werra, D.: Applications to Timetabling. Section 5.6 in: Gross, J., Yellen, J. (eds.): Handbook of Graph Theory, Chapman and Hall/CRC Press, London (2004) 445–474
8. Burke, E. K., Landa, J. D.: Design of Memetic Algorithms for Scheduling and Timetabling Problems. In: Krasnogor, N., Hart, W., Smith, J. (eds.): Recent Advances in Memetic Algorithms and Related Search Technologies. Springer, Berlin (2004) 289–312
9. Burke, E. K., MacCarthy, B., Petrovic, S., Qu, R.: Structured Cases in CBR—Re-using and Adapting Cases for Time-Tabling Problems. Knowledge-Based Syst. **13** (2000) 159–165
10. Burke, E., MacCarthy, B., Petrovic, S., Qu, R.: Knowledge Discovery in a Hyper-heuristic Using Case-Based Reasoning for Course Timetabling. In: Burke, E., De Causmaecker, P.: Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 276–287
11. Burke, E. K., MacCarthy, B., Petrovic, S., Qu, R.: Multiple-Retrieval Case Based Reasoning for Course Timetabling Problems. J. Oper. Res. Soc. (2005) accepted for publication

12. Burke, E. K., Meisels, A., Petrovic, S., Qu, R.: A Graph-Based Hyper Heuristic for Timetabling Problems. Eur. J. Oper. Res. (2005) accepted for publication
13. Burke, E. K., Newall, J. P., Weare, R. F.: A Memetic Algorithm for University Exam Timetabling. In: Burke, E., Ross, P.: The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 241–250
14. Burke, E. K., Newall, J. P.: Enhancing Timetable Solutions with Local Search Methods. In: Burke, E., De Causmaecker, P.: Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 195–206
15. Burke, E. K., Newall, J. P., Weare, R. F.: Initialisation Strategies and Diversity in Evolutionary Timetabling. Evol. Comput. **6** (1998) 81–103
16. Burke, E. K., Newell, J. P.: Solving Examination Timetabling Problems Through Adaptation of Heuristic Orderings. Ann. Oper. Res. **129** (2004) 107–134
17. Burke, E. K., Newell, J. P., Weare, R. F.: A Simple Heuristically Guided Search for the Timetable Problem. In: Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems (University of La Laguna). Academic, New York (1998) 574–579
18. Burke, E. K., Petrovic, S.: Recent Research Directions in Automated Timetabling. Eur. J. Oper. Res. **140** (2002) 266–280
19. Burke, E. K., Petrovic, S., Qu, R.: Case Based Heuristic Selection for Timetabling Problems. J. Scheduling (2006) accepted for publication
20. Carter, M. W.: A Survey of Practical Applications on Examination Timetabling. Oper. Res. **34** (1986) 193–202
21. Carter, M. W., Laporte, G.: Recent Developments in Practical Course Timetabling. In: Burke, E., Ross, P.: The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 3–21
22. Carter, M. W., Laporte, G., Chinneck, J. W.: A General Examination Scheduling System. Interfaces **24** (1994) 109–120
23. Carter, M. W., Laporte, G., Lee, S. Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
24. Carter, M. W., Johnson, D. G.: Extended Clique Initialisation in Examination Timetabling. J. Oper. Res. Soc. **52** (2001) 538–544
25. Casey, S., Thompson, J.: GRASPing the Examination Scheduling Problem. In: Burke, E., De Causmaecker, P.: Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 232–246
26. Deng, P. S.: Using Case-Based Reasoning Approach to the Support of Ill-structured Decisions. Eur. J. Oper. Res. **93** (1996) 511–521
27. Di Gaspero, L., Schaerf, A.: Tabu Search Techniques for Examination Timetabling. In: Burke, E., Erben, W.: The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 104–117
28. Dueck, G.: New Optimization Heuristics. J. Comput. Phys. **104** (1993) 86–92
29. Garey, M. R., Johnson, D. S.: Computers and Intractability a Guide to the Theory of NP-completeness. Freeman, San Francisco (1977)
30. Gendreau, M., Soriano, P., Salvail, L.: Solving the Maximum Clique Problem Using a Tabu Search Approach. Ann. Oper. Res. **41** (1993) 385–403
31. Johnson, D.: Timetabling University Examinations. J. Oper. Res. Soc. **41** (1990) 39–47

32. Kolodner, J.: Case-Based Reasoning. Morgan Kaufmann, San Mateo, CA (1993)
33. Laporte, G., Desroches S.: Examination Timetabling by Computer. Comput. Oper. Res. **11** (1984) 351–360
34. Leake, D. B.: CBR in Context: the Present and Future, Case-Based Reasoning: Experiences, Lessons, and Future Directions. AAAI Press/MIT Press, Menlo Park, CA (1996)
35. Merlot, L. T. G., Boland, N., Hughes, B. D., Stuckey, P. J.: A Hybrid Algorithm for the Examination Timetabling Problem. In: Burke, E., De Causmaecker, P.: Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 205–232
36. Miyashita, K., Sycara, K.: CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair. Artif. Intell. **76** (1995) 377–426
37. Petrovic, S., Beddoe, G. R., Berghe, G. V.: Storing and Adapting Repair Experiences in Employee Rostering. In: Burke, E., De Causmaecker, P.: Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 149–166
38. Petrovic, S., Burke, E.: Educational Timetabling. Chapter 45 in: Leung, J. (ed.): Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC Press, London (2004), 45.1–45.23
39. Petrovic, S., Kendall, G., Yang, Y.: A Tabu Search Approach for Graph-Structured Case Retrieval. In: Proc. STarting Artificial Intelligence Researchers Symposium (France). IOS Press (2002) 55–64
40. Petrovic, S., Yang Y., Dror, M.: Case-based Initialisation of Metaheuristics for Examination Timetabling. In: Kendall, G., Burke, E., Petrovic, S., Gendreau, M. (eds.): Multidisciplinary Scheduling Theory and Applications. Springer, Berlin (2005) 289–308
41. Petrovic, S, Yang, Y., Dror, M.: Use of Case Based Reasoning In Solving Examination Timetabling Problems. Technical Report NOTTCS-TR-2004-6, University of Nottingham, UK (2004)
42. Schirmer, A.: Case-Based Reasoning and Improved Adaptive Search for Project Scheduling. Naval Res. Log. **47** (2000) 201–222
43. Schmidt, G.: Case-Based Reasoning for Production Scheduling. Int. J. Product. Econ. **56/7** (1998) 537–546
44. Terashima-Marín, H., Ross, P., Valenzuela-Rendón, M.: Evolution of Constraint Satisfaction Strategies in Examination Timetabling. In: Proceedings of the Genetic and Evolutionary Conference (1999) 635–642
45. Thompson, J. M., Dowsland, K. A.: Variants of Simulated Annealing for the Examination Timetabling Problem. Ann. Oper. Res. **63** (1996) 105–128
46. Welsh, D. J. A., Powell, M. B.: An Upper Bound on the Chromatic Number of a Graph and its Application to Timetabling Problems. Comput. J. **10** (1967) 85–86
47. White, G. M., Xie, B. S., Zonjic, X.: Using Tabu Search With Longer-Term Memory and Relaxation to Create Examination Timetables. Eur. J. Oper. Res. **153** (2004) 80–91
48. Zadeh, L. A.: Fuzzy Sets. Inform. Control **8** (1965) 338–353
49. Zadeh, L. A.: The Concept of a Linguistic Variable and its Application to Approximate Reasoning. Inform. Sci. **8, 9** (1975) 199–249, 43–80, respectively

A Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at the MARA University of Technology

Graham Kendall[1] and Naimah Mohd Hussin[2]

[1] Automated Scheduling, Optimisation and Planning (ASAP) Research Group,
School of Computer Science and Information Technology, University of Nottingham,
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
gxk@cs.nott.ac.uk

[2] Faculty of Information Technology and Quantitative Sciences,
MARA University of Technology, 40450 Shah Alam,
Selangor Darul Ehsan, Malaysia
naimah@tmsk.uitm.edu.my

Abstract. In this paper we introduce an examination timetabling problem from the MARA University of Technology (UiTM). UiTM is the largest university in Malaysia. It has 13 branch campuses and offers 144 programmes, delivered by 18 faculties. This dataset differs from the others reported in the literature due to weekend constraints that have to be observed. We present their examination timetabling problem with respect to its size, complexity and constraints. We analyse their real-world data, and produce solutions utilising a tabu-search-based hyper-heuristic. Since this is a new dataset, and no solutions have been published in the literature, we can only compare our results with an existing manual solution. We find that our solution is at least 80% better with respect to proximity cost. We also compare our approach against a benchmark dataset and show that our method is able to produce good quality results.

1 Introduction

Work on timetabling problems has been published since the 1960s [17]. Since then, numerous researchers have been working on problems ranging from sports timetabling [35], railway timetabling [14], [25] and educational timetabling [2], [3], [6], [15], [17], [30].

Most academic institutions face the problem of scheduling both courses and examinations in every semester or term. As the difficulty of the problem increases, due to a large number of students, courses, examinations, rooms and invigilator constraints, an automated timetabling system that can produce feasible, and high quality timetables, is often required. The timetabling procedure at universities and schools varies from manual timetabling, semi-automated timetabling to fully automated timetabling. A survey conducted by Burke et al. [10] received feedback from 56 registrars from British universities with regard

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 270–293, 2005.

to the nature of their examination timetabling problem, how they solved it (manual or automated) and what qualities were considered for a good examination timetable. They discovered that only 58% (32 universities) of their respondents used a computer at some stage in producing their examination timetable and 21% (11 universities) of these had a scheduling system. Only two universities used commercial software whilst the other systems were developed in house. Thus, while commercialised software is available, such as, EXAMINE [16], Syllabus-Plus [23], ConBaTT [24], OPTIME [28], and CelCAT [32], many universities are yet to be convinced that an automated system will provide a satisfactory solution. Universities may need to develop their own system or customise a commercial system to fulfil their specific needs. Once a customised system is developed, it will also require full support with frequent updates and maintenance due to changes in academic policy or educational structures.

An early survey by Comm and Mathaisel [19] in 1988 involving 1494 U.S. college registrars concluded that there was a large market for a computerised timetabling system and that most registrars were unhappy with their current systems. The survey showed that a computerised system must produce good quality timetables allowing some user intervention, it must be easy to use and be comprehensive and compatible with any previous systems. JISC (Joint Information System Committee) Technology Applications Programme [26] published findings from a questionnaire from which they received replies from 16 universities in the UK. The universities were asked whether a central computerised system was in use and for their views on its effectiveness. The report concluded that centralising the whole process of room bookings, examination timetables and lecture timetabling was carried out in phases using a wide variety of software packages and there was a need for full and complete management support for such systems.

The above surveys show that much computerisation has taken place over the years, and there is ongoing research on new techniques and methods to solve timetabling problems so as to produce better quality solutions. In this paper, we focus on the examination timetabling problem faced by universities. Carter and Laporte [17] defined the basic problem as “the assigning of examinations to a limited number of available time periods in such a way that there are no conflicts or clashes”. A timetable is feasible if all examinations can be scheduled and all other hard constraints are not violated. Student conflict exists if at least one student is scheduled to sit for more than one examination at the same time. This conflict is categorised as a hard constraint and should be eliminated. A proximity constraint is an example of a soft constraint that can be violated, if necessary. A weighted proximity cost is assigned whenever a student has to sit for two examinations scheduled at most five slots apart. A lower proximity cost relates to a better quality solution and thus our objective is to minimise the proximity cost.

Carter et al. [18] provide a set of benchmark examination timetabling instances from real examination problem datasets from universities across the world. They applied a variety of constructive algorithms, with backtracking, based on graph colouring heuristics and solve the problem with and without capacity constraints (capacitated and un-capacitated problem). The solution quality is measured by an objective function based on proximity cost. A lower proximity cost indicates that, on average, a student will have his/her examinations better spread over the length of his/her examination period and, therefore, will have more time to concentrate on each examination.

Burke et al. [12] introduced another dataset from the University of Nottingham that also includes room requirements and capacities. They used a memetic algorithm and a weighted objective function for adjacent, overnight and unscheduled examinations. Burke et al. [11] apply some sequential heuristics and various ordering techniques to allocate examinations to slots whilst not violating the clash and capacity constraints.

Merlot et al. [29] introduced two new datasets from the University of Melbourne that includes two additional hard constraints: examination availability (examinations preassigned to specific slots) and large examinations (large examinations scheduled in the first n slots). They used a hybrid algorithm (a constraint method, simulated annealing and hill climbing) to find a feasible schedule and found that they have to relax the constraints (by adding additional slots to the large examinations constraints) so as to produce a clash free timetable.

All of the datasets described above are available via

– ftp://ftp.mie.utoronto.ca/pub/carter/testprob
– ftp://ftp.cs.nott.ac.uk/ttp/Data and
– http://www.or.ms.unimelb.edu.au/timetabling

We presented the examination timetabling problem from the MARA University of Technology in [20]. MARA University of Technology (UiTM) is the largest university in Malaysia with a total number of students approaching 100,000. The university has 13 branch campuses, one in every state in Malaysia with 144 programmes offered by 18 faculties. A common examination timetable is shared amongst all campuses and programmes since students sitting for the same examination paper must take it at the same time, irrespective of their geographical location.

UiTM uses an examination scheduling program [36] (developed in-house, using the COBOL language, about 30 years ago) to produce a first draft of the examination timetable. The timetable then goes through a manual update process by scheduling new courses and removing old ones.

Apart from the constraints that are common for examination timetabling [8], [15], [17], [33], UiTM has to consider the following additional constraint: If an examination falls on a state public holiday and there are students from that state sitting for the examination then the examination must be moved to another slot. Malaysia has a number of public holidays that are not shared between states. Data on the available space for examinations are not considered when moving examinations and therefore every faculty, centre and branch campus needs to provide prompt feedback on space availability for examinations already scheduled. Once the examination timetable is in its final draft, it is sent to all faculties and branch campuses for the assignment of rooms and invigilators.

In this paper, we apply a tabu-search-based hyper-heuristic approach to the UiTM examination timetabling problem. This method has been used to solve the Carter benchmark datasets and has produced competitive results compared to other methods [27]. Hyper-heuristic [7], [9] methods operate at a higher level of abstraction than other search methods and are able to intelligently choose a (meta-)heuristic to be applied at any given time. We refer to Burke et al. [7] for further motivation and discussion on the emergence of hyper-heuristics in solving optimisation problems. An example of a hyper-heuristic approach for solving a large-scale university examination timetable can be seen in Terashima-Marin et al. [34]. The problem is solved in phases. Each phase uses a different set of heuristics and a switch condition determines when to move from one phase to the other. A genetic algorithm, using an indirect chromosome representation, was used to evolve the choice of heuristics and the switch condition. Another recent application of hyper-heuristic on timetabling problems is presented Burke et al. [5]. The hyper-heuristic uses tabu search to search for different permutations of graph heuristics that solve both examination and course timetabling problems. A further use of a hyper-heuristic is in solving multi-objective [4] space allocation and timetabling problems. The idea is to choose, at each iteration, a heuristic that is suitable for the optimisation of a given, individual objective.

Tabu search is a meta-heuristic that has been applied successfully on examination timetabling problem [21], [37]. In this paper, we incorporate the tabu list mechanism in our hyper-heuristic framework to help guide the selection of heuristics.

In the next section, we present a detailed description of the examination timetabling problem at the MARA University of Technology. Section 3 provides a description of our tabu-search-based hyper-heuristic and Section 4 presents our algorithm with this new large dataset. We show how it compares with the only other known solution to this problem. Section 5 concludes with a summary and presents potential future research directions.

2 UiTM Examination Timetable Problem Formulation

2.1 Problem Description

MARA University of Technology (UiTM) is the largest university in Malaysia with 13 branch campuses. The Centre for Integrated Information System (CIIS), UiTM has responsibility for planning and managing the overall Information Technology strategy in UiTM and fulfilling administrative and academic needs. One of its information system modules, Integrated Student Information Systems (iSIS), was designed and developed as a collaborative project. This system offers six main modules encompassing the complete Student Life Cycle process, from Intake to Convocation and Alumni. The four principal modules comprise the Student Intake System, Academic Affairs Systems, Student Affairs System and Student Accounting System. Thirteen personnel headed by a senior information system officer provide support for the system and two information system officers are specifically assigned to the examination unit.

We have been fortunate to acquire most of our examination data from the CIIS. The data are classified by faculty and branch campuses. The total number

Table 1. Characteristics of UiTM dataset

No.	Branch campuses	No. of students	No. of student exams	Avg exam per student
1	Shah Alam	40,275	222,559	5.52
2	Melaka	4,447	25,334	5.70
3	Negeri Sembilan	315	1,913	6.07
4	Johor	4,057	22,109	5.45
5	Perak	5,366	31,518	5.87
6	Perlis	5,824	32,807	5.63
7	Kelantan	3,515	19,627	5.58
8	Terengganu	4,842	27,444	5.67
9	Pahang	4,607	27,221	5.91
10	Sarawak	4,800	25,618	5.34
11	Sabah	2,423	11,470	4.73
12	Penang	1,453	9,296	6.40
13	Kedah	2,751	15,285	5.56
	UiTM-03 (Total)	84,675	472,201	5.56

of students enrolled in Semester 2, 2002/03 was 84,675, course enrolment was 472,201 and the total number of courses to be scheduled was 2,650 (see Table 1).

The data supplied from CIIS were in the form of: a list of examinations that must be scheduled, a list of examinations that must be scheduled at the same time (concurrent), a list of students and their course selection (split by campuses) and an examination timetable that was used in the May 2003 semester. We do not have any information regarding capacity or other hard constraints that were imposed via feedback by faculties or campuses. The original list of examinations has 2,650 examinations to be scheduled, but after computing the enrolment for each examination using the student file, examinations with zero enrolments were removed even though these examinations were present in the examination timetable. Examinations with zero enrolments will have no effect in the way we compute solution quality and therefore can be removed from the timetable. There are 491 examinations with zero enrolments and so we only need to schedule 2,159 examinations. Data accuracy is one of the difficulties in processing data from a large university that involves many faculties and campuses. We believe that these examinations have zero enrolments because officers in faculties and campuses may not have up-to-date information on the exact course enrolments and, also, students add or drop courses at the last moment.

2.2 Problem Formulation

We can represent the examination timetabling problem as follows:

1. E: a set of m examinations E_1, $E_2, \ldots, E_m$.
2. S: a set of n slots S_1, $\mathrm{S}_2, \ldots, S_n$.

3. U: a set of u campuses U_1, $\mathrm{U}_2, \ldots, U_u$.
4. A final examination timetable T_{mn} such that: $T_{ik} = 1$ if examination i is scheduled in slot k, 0 otherwise.
5. CampusType = {A, B} where campus type A has half-day Saturday and full-day Sunday weekend and campus type B has half-day Thursday and full-day Friday weekend.
6. A conflict matrix C_{mm} such that C_{ij} = total number of students sitting for both examinations i and j categorised by campus type.
7. A co-schedule matrix R_{mm} such that $R_{ik} = 1$ if examination i and examination k must be scheduled in the same time slot, 0 otherwise.

The examination timetabling problem is to assign examinations to n number of slots subject to various constraints, so as to minimise various costs. The total number of slots is already fixed and the examination scheduler must schedule examinations into at most n slots. There are 2,159 examinations with some examinations being held concurrently. Even though each examination may have a different duration (120, 150 or 180 minutes), we will treat each examination as occupying a complete slot of 180 minutes. The total number of examination days is 20 and each day will have two slots (morning and afternoon), i.e. we will have 40 slots in which to assign examinations. We categorise each campus (A or B) to indicate which days or slots must not be used in assigning examinations taken by students in that particular campus. Campuses in category A have weekend: half-day on Saturday and full-day on Sunday while campuses in category B have weekend: half-day on Thursday and full-day on Friday. The examination dates were from 20th April 2003 to 10th May 2003 with 1st May as a public holiday across all campuses, so no examinations can be scheduled on this day. For this dataset, we are not concerned with other public holidays since none occur on a different day for different states. On other occasions there might be a situation where a campus has a public holiday and others do not.

Constraints. Hard constraints are those which cannot be violated. A timetable that violates a hard constraint will render it infeasible. Such infeasibilities may be unavoidable in certain cases and universities have to take drastic measures to resolve the problem.

The following hard constraints are present in the UiTM dataset:

a Conflict: no student should sit for more than one examination in the same slot:

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n} T_{ik}T_{jk}C_{ij} = 0\,.$$

If examination i and examination j are scheduled in slot k, the number of students sitting for both examination i and j (C_{ij}) must be equal to zero, and this should be true for all examinations already allocated.

b Co-schedule: examinations that must be scheduled together must be assigned in the same time slot.

For all examinations i:

$$\sum_{j=1}^{m}\sum_{k=1}^{n} T_{ik}T_{jk}R_{ij} = \sum_{j=1}^{m} R_{ij} .$$

All examinations that must be scheduled with examination i should be assigned to the same slot k.

c All examinations must be scheduled:

$$\sum_{k=1}^{n} T_{ik} = 1 \qquad i = 1, \ldots, m .$$

Each examination, $(E_1,\, E_2, \ldots, E_m)$ should be scheduled only once.

Costs. Other additional soft constraints that are specific to university requirements can be added to this problem. We determine the cost of an examination timetable solution based on the penalty given if certain soft constraints are violated. The soft constraints that we consider are as follows:

a *Proximity cost.* A proximity cost x_s is given whenever a student has to sit for two examinations scheduled s periods apart: these weights are $x_1 = 16$, $x_2 = 8$, $x_3 = 4$, $x_4 = 2$ and $x_5 = 1$. P_{ik} is the proximity cost if examination i is scheduled in slot k.

The total proximity cost of a timetable is as follows:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} T_{ik}P_{ik} .$$

b *Weekend cost.* The current examination timetable scheduler schedules examinations during the weekend. We can try to produce a better quality timetable by penalising examinations that are scheduled during the weekend. Weekends for category A campuses are half-day on Saturday and full-day on Sunday and weekends for category B campuses are half-day on Thursday and full-day on Friday. A penalty cost of 16 is given whenever a student has to sit for a weekend examination. W_{ik} is the proximity cost if examination i is scheduled in slot k.

The total weekend cost is as follows:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} T_{ik}W_{ik} .$$

Finally, our objective is to optimise the following:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} T_{ik}P_{ik} + \sum_{i=1}^{m}\sum_{k=1}^{n} T_{ik}W_{ik}$$

i.e. minimise the total proximity cost and the weekend cost.

3 Tabu Search Based Hyper-heuristic

A hyper-heuristic [7], [9] works at a higher level of abstraction than a meta-heuristic and does not necessarily require domain knowledge. A hyper-heuristic only has access to non-domain-specific information that it receives from the heuristics that it operates upon. A hyper-heuristic can be implemented as a generic module that has a common interface to the various low-level heuristics and other domain specific knowledge (typically the evaluation function) of the problem being solved. Initially, the hyper-heuristic needs to know the number of n heuristics provided by the low-level heuristic module. It will guide the search for good quality solutions by setting up its own strategy for calling and evaluating the performance of each heuristic known by their generic names H_1, H_2, ..., H_n. The hyper-heuristic does not need to know the name, purpose or implementation detail of each low-level heuristic. It just needs to call a specific heuristic, H_i, and the heuristic may modify the solution state and return the result via an evaluation function. The low-level-heuristic module can be viewed as a “black box” that hides the implementation details and only returns a new solution and a revised value for the evaluation function.

3.1 Hyper-heuristic Module

The hyper-heuristic module is the focus of this research, where we need to design and test strategies that can intelligently select the best heuristic that will help guide the search to either intensify or diversify the exploration of the search region.

The general framework for our hyper-heuristic is as follows:

```
Step 1   Construct initial solution
Step 2   Do
           Consider heuristics that are not tabu
           Choose the best heuristic (with the best improvement)
           Apply chosen heuristic and make the heuristic tabu
           Update solution
           Update the tabu status of heuristics in the tabu list
         Until terminating condition
```

The initial solution is produced using a constructive heuristic (largest degree or saturation degree [17]. Next, a randomisation (randomly move examinations to other valid slots) is carried out in order to start different runs with different solutions. In Step 2 we explore the neighbourhood to search for a better solution or local optima (and possibly global optima). The framework is similar to a local search except that in Step 2, we explore the neighbourhood by selecting which heuristic to use to update the current solution.

The core part of the algorithm is Step 2, where we need to decide which heuristics are candidates to be applied and which heuristic we actually apply. Each heuristic differs in how it decides to move, thus creating its own search region (heuristic search space) in the solution search space. In the search for good quality solutions, the hyper-heuristic exhibits a type of reinforcement learning that will assist in an intelligent action at each decision point. At this point, the hyper-heuristic can actually choose intelligently when to intensify or diversify the search because we believe that allowing the low-level heuristics to compete at each iteration and selecting the heuristic with the best performance will help to balance the diversification and intensification of the solution search space. Heuristics that have been applied become tabu so that in the next iteration we can explore the solution space of other low-level heuristics that may perform well but, perhaps, not as well as the previous heuristics that are now tabu.

The hyper-heuristic monitors the behaviour of each low-level heuristic by storing information about their performance using an adaptive memory. Our hyper-heuristic uses a tabu list that is of a fixed length n, where n is the number of low-level heuristics. Instead of storing moves, each tabu entry stores (non-domain) information about each heuristic i.e. heuristic number, recent change in evaluation function, CPU time taken to run the heuristic, and tabu status (or tabu duration, which is the term we use here). Tabu duration indicates how long a heuristic should remain tabu and, will therefore, not be applied in the current iteration. If the tabu duration is zero, the heuristic is said to be tabu inactive and can be applied to update the solution. If the tabu duration is non-zero, the heuristic is said to be tabu active and may not be used to update the solution. The tabu duration is set for a heuristic whenever a tabu restriction is satisfied. After each iteration, the tabu duration is decremented until it reaches zero and the heuristic is now tabu inactive. We do not use any aspiration criteria since a tabu active heuristic will have its tabu duration decremented in each iteration, and will eventually be tabu inactive. If all heuristics are tabu active in any iteration, no heuristics will be evaluated and obviously none will be applied. Therefore, a heuristic changes its status from tabu active to tabu inactive only when the tabu duration is zero. In using a tabu list, we need to decide what tabu duration value works best for a given problem instance.

Our first implementation used a deterministic tabu duration where, in each run, we used a fixed range of tabu duration and compared the final best solution. In our previous work [27], we showed empirically, using deterministic tabu durations, that an effective tabu duration is dependent upon the conflict matrix density of a given examination timetabling problem instance. In general, having a low tabu duration allows the exploration to move within the same heuristic search space and a high tabu duration allows exploration into other regions. If a tabu duration is too high, we found that the quality of the solution deteriorates since good heuristics are made tabu for too long. If a tabu duration is too low (or equal to zero), we have limited ourselves to search within a small region in the solution space. In this paper, we will consider both deterministic tabu durations and random dynamic tabu durations. Random dynamic tabu duration uses a tabu duration range (t_{min} and t_{max}), and at each decision point, when making a heuristic tabu, a tabu duration value is selected at random from the given range.

The next issue we have to address is the mechanism for updating a solution. At each iteration, we compare the solution from each heuristic and take the best solution. We use three different strategies for accepting the best solution at each decision point. First, is to accept the solution from the best performing heuristic. This best solution may not improve the current or previous best solution. Second, is to accept the best solution and if this solution improves the previous solution, the same heuristic will be applied until it cannot improve the solution anymore (steepest ascent hill climbing). Third, is to accept the best solution only if the solution is less than a boundary penalty. The boundary penalty was first introduced by Dueck [22] in his great deluge algorithm (GDA). Burke et al. [13] have also applied it to examination timetabling. The algorithm deals with feasible solutions and will accept worse solutions if the cost evaluation is less than or equal to a boundary, called level. The level is reduced at each iteration using a decay rate that is dependent upon the initial cost evaluation, desired cost evaluation and the time for the algorithm to achieve this desired cost.

We have implemented and tested three different hyper-heuristics, which incorporate the various strategies mentioned above.

1. The simplest form, i.e. the Basic Tabu Search Hyper-heuristic (TSHH-B), considers all tabu inactive heuristics and applies the heuristic that has the best improvement only. The algorithm iterates for a fixed time (4 hours).
2. The second hyper-heuristic is Tabu Search Hyper-heuristic with Hill Climbing (TSHH-HC), which adds to TSHH-B a successive call to the best performing heuristic until no further improvement is made.
3. The third hyper-heuristic is Tabu Search Hyper-heuristic with Great Deluge (TSHH-GD), which updates a solution within a certain boundary only. The execution time is limited by a number of steps (i.e. 10,000,000 iterations) and the algorithm will terminate if there is no improvement in the last 10,000 iterations. The time taken by a TSHH-GD iteration is longer compared to GDA because GDA generates only one new solution by doing a random move while TSHH-GD generates several solutions as it calls every heuristic which is not tabu. Each low-level heuristic may also take a longer time than just a random move. The decay rate is calculated as the difference between the desired cost and the initial cost divided by the number of iterations. In each iteration, the boundary is lowered by the decay rate. A solution from the best performing low-level heuristic is accepted to modify the current solution if it is better than the previous solution or it is lower than the boundary.

For each of these hyper-heuristics, we apply both deterministic and random dynamic tabu durations.

3.2 Low-Level Heuristics Module

Low-level heuristics are heuristics that allow movement through a solution space and that require domain knowledge and are problem dependent. Each heuristic creates its own heuristic search space that is part of the solution search space.

The idea is to build a collection of (possibly) simple moves or choices since we would like to provide a library of heuristics that can be selected intelligently by a hyper-heuristic tool.

The heuristics change the current state of a problem into a new state by accepting a current solution and returning a new solution. Each low-level heuristic can be considered as improvement heuristics that returns a move, a change in the penalty function and the amount of time taken to execute the heuristic. The best performing heuristic should cause a maximum decrease in penalty (the lowest value). Each move from an individual heuristic may cause the search to probe into the current neighbourhood or to explore a different neighbourhood. A change in the penalty value means changing the penalty value for each of the soft constraints that were violated (first-order conflict, second-order conflict, etc) or moving an examination into an unscheduled list (examination becomes unscheduled and violates hard constraints).

We use the same low-level heuristics as in [27]:

1. Five graph colouring heuristics that select an examination from an unscheduled list and schedule it into the best available slot that maximises the reduction in penalty. The heuristics are: largest enrolment, largest examination conflict, largest total student conflict, largest examination conflict already scheduled, and examination with least valid slots.
2. Five move heuristics that select an examination, either at random, with maximum penalty, with highest second order conflict or highest first order conflict. This examination is rescheduled into a new random slot or a new slot which maximises the reduction in either the total penalty, total first order conflict or second order conflict.
3. Two swap heuristics that select an examination, either at random, with maximum penalty or with minimum penalty. The two examinations selected will swap slots subject to no hard constraint violations.
4. A heuristic that removes a randomly selected examination from the examinations already scheduled. This is the only heuristic which will move the search into an infeasible region because any examination may be unscheduled. We make sure that the search can move back into its feasible region by un-scheduling examinations that have other valid slots to move to in the next iteration.

All of the above low-level heuristics are either 1-opt (one move) or 2-opt (two moves) and there is also a mixture of some randomness and some deterministic selection of examinations and slots. We purposely use low-level heuristics that are simple moves rather than complex low-level heuristics because we want to make sure that the hyper-heuristic can recognise good moves and make an intelligent decision based on these simple moves. Furthermore, we want to make the problem-domain knowledge heuristics easy to implement and the hyper-heuristic more generalised.

Table 2. Comparison of results between Uitm-03 and Car-s-91 datasets

Tabu	Uitm-03			Car-s-91		
Duration	TSHH-B	TSHH-HC	TSHH-GD	TSHH-B	TSHH-HC	TSHH-GD
0	2.16	2.08	2.14	6.88	6.78	6.85
1	1.55	1.55	1.44	6.83	6.88	7.01
2	1.93	1.94	1.37	5.37	5.14	5.15
3	3.95	3.84	**1.35**	6.31	6.04	**4.93**
4	6.37	6.33	1.44	6.91	7.10	5.01
Random	1.65	1.63	1.40	5.43	5.31	5.6

4 Experimental Results

We have implemented and tested our tabu search based hyper-heuristic framework on a PC with an AMD Athlon 1 GHz processor, 128 Mb RAM and Windows 2000. The program was coded in C++ using an object-oriented approach. We defined and implemented the hyper-heuristic and heuristics as objects that have a common interface and can interact with each other. In our previous paper [27], we tested the simplest form of our hyper-heuristic module TSHH-B with deterministic tabu duration on Carter's examination timetabling benchmark data. We compared the results (using proximity cost per student) and found that the method is able to find good solutions on all datasets. Our tabu-search-based hyper-heuristic method has also added a significant improvement to tabu search because our results are better in all cases compared to the tabu search approach of Di Gaspero and Schaerf [21]. Without changing the low-level heuristics, we tested two more hyper-heuristics with two different objective functions and produced results with respect to minimising proximity cost and minimising both proximity and weekend cost

4.1 Proximity Cost Evaluation Method

We use our previous method (TSHH-B) and proximity cost evaluation function on our new dataset. We also run our new and improved method: tabu-search-based hyper-heuristic with hill climbing (TSHH-HC) and tabu-search-based hyper-heuristic with great deluge (TSHH-GD), with Carter's examination timetabling benchmark data and the Uitm-03 dataset. Here, we discuss the algorithm performance when applied to the Uitm-03 dataset and compare its behaviour with one other dataset, i.e. Car-s-91 (one of the benchmark datasets). Car-s-91 (Carleton University, Ottawa) is one of the largest datasets with 682 examinations to be scheduled in 35 slots, 16,925 students and 56,877 student examinations.

The first column in Table 2 shows the tabu duration, i.e. how long a heuristic is placed in the tabu list. We tested with both deterministic and random dynamic tabu durations. The best result for Car-s-91 is 4.50 published by Yong and

Table 3. Compare results for UiTM dataset with other solution methods

	TSHH-B	TSHH-HC	TSHH-GD	Manual	GD	SA
UiTM-03	1.55	1.55	1.35	12.83	1.40	1.68

Petrovic [38]. They used a combination of case base reasoning using a fuzzy similarity measure to choose sequential heuristics during the construction phase and then the great deluge algorithm. The next best result of 4.65, published by Burke and Newall [1], uses a combination of adaptive initialisation strategies and the great deluge algorithm. Both methods are based upon the idea that good initial solutions fed to the great deluge algorithm will produce good solutions. Referring to Table 2, our best result for Car-s-91 is 4.93 using the tabu-search-based hyper-heuristic and the great deluge with a tabu duration of 3. Thus, the hyper-heuristic is able to produce a good-quality solution for the Car-s-91 dataset, even though the initial solution is not as good as the initial solutions in [1], [38]. Our method shows similar behaviour between both datasets, where the best result is obtained using tabu search hyper-heuristic and great deluge with a tabu duration of 3. Table 3 compares our best result on the UiTM dataset with the existing manual solution, the great deluge algorithm and simulated annealing.

Figures 1 and 2 show the best proximity cost, with various tabu durations obtained for the Uitm-03 and Car-s-91 datasets. In our tabu-based hyper-heuristic strategy, we apply the concept of heuristics cooperating with each other rather than penalising a non-performing heuristic. When the tabu duration is greater than zero, we apply a tabu restriction where a heuristic will be tabu active if its solution value has been accepted to update the current solution. The heuristic will remain tabu active for a number of steps equal to tabu duration. We make a heuristic tabu because we want to direct the search to other possible heuristic search spaces. Eventually we may return to a heuristic search space once it is no longer tabu active and can give the best solution amongst all tabu inactive heuristics. Similar to the tabu search meta-heuristic, we need to decide which tabu duration (or list size in tabu search) works best for a given problem instance. For the UiTM dataset (Figure 1), as the tabu duration increases, solution quality improves, and once it reaches its best tabu duration, the solution quality begins to deteriorate as we increase the tabu duration further. Car-s-91 (Figure 2) shows only a slight difference when the tabu duration equals zero and tabu duration equals one for both hyper-heuristics that either incorporate hill climbing or great deluge. In that instance, solution quality is better when tabu duration is zero compared to when the tabu duration is one and improves again after that.

Figures 3–5 show the hyper-heuristic performance with different tabu durations. The graphs show how the hyper-heuristic explores the search space. Both TSHH-B and TSHH-HC show more dispersed points in the graph compared to TSHH-GD because the boundary penalty used in great deluge directs the search

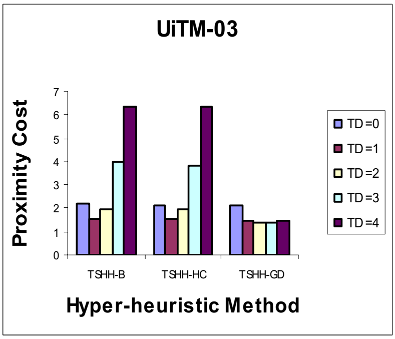


Fig. 1. Hyper-heuristic methods on UiTM-03

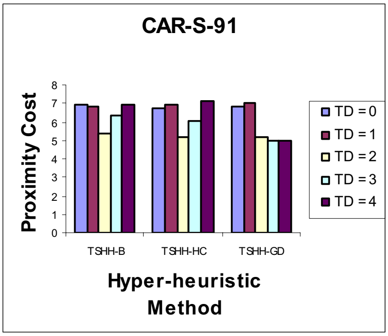


Fig. 2. Hyper-heuristic methods on Car-s-91

to a much better solution. The lower tabu duration does not show much movement in the search space and it might be trapped in local optima. As we increase the tabu duration, it shows more movement and exploration of search space and is therefore able to find a better quality solution. As the tabu duration increases it becomes more difficult to get better solutions since many heuristics stay tabu too long and the hyper-heuristic has no option but to take the best solution from the worst heuristics. TSHH-GD is better at not moving to a worse solution since it is also directed by the boundary penalty and thus will not make bad moves in such a way that it cannot get back to good solution space. Figure 6 shows a comparison of the TSHH-GD with great deluge algorithm (see the algorithm in [11]). We ran both algorithms for 4 hours and traced the penalty evaluation at every 5,000 iterations (steps). The TSHH-GD converges faster in terms of steps but it actually takes longer because each iteration explores various heuristic solution space and may take 10 times longer compared to the great deluge algorithm.

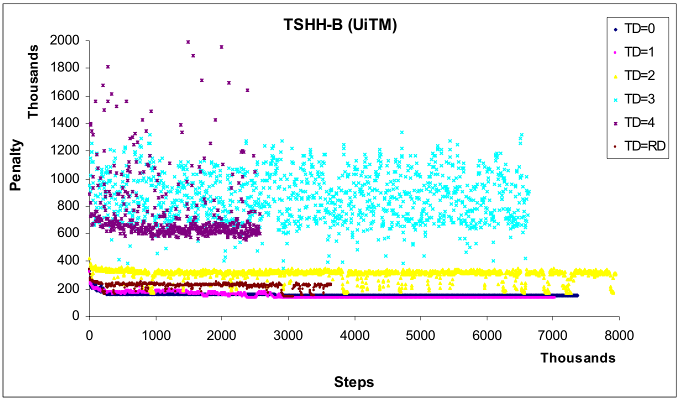


Fig. 3. Basic tabu-search-based hyper-heuristic for UiTM dataset

4.2 Proximity and Weekend Cost Evaluation Method

The UiTM dataset has an additional constraint or criterion, i.e. a weekend cost that needs to be minimised. The proximity cost and weekend cost are combined into one objective function with the same weights for proximity cost and a weight of 16 for every student who has to sit for a weekend examination.

The high-level part of the hyper-heuristic does not need to be changed. The low-level heuristics, which require more knowledge about the problem, need to be adjusted. The cost evaluation function is also modified to reflect a change in the objective function. The unscheduled examination weight needs adjustment (new weight of 10,000) and a new weekend weight added. In the previous implementation, two heuristics minimise the second- and third-order conflict (a conflict by a student having to sit for examinations, 1 or 2 slots apart). In the current implementation, a new heuristic, reduces a conflict by a student who has to sit for a weekend examination.

In general, we can further improve our low-level heuristics by generalising the number of conflicts and the type of conflicts we want to minimise. A separate low-level heuristic can be used to minimise a specific criteria and the hyper-heuristic can choose which criteria to optimise in each decision step. In this way, the tabu search hyper-heuristic is also a general method that can solve an examination timetabling problem with multiple objectives. Petrovic and Bykov [31] solve the examination timetabling problem with multiple objectives by dynamically changing the weights of the criterion during the search process. The hyper-heuristic approach does this implicitly because it has the intelligence to choose between the low-level heuristic that minimises the first or the second criterion. The only parameter that we need to adjust would be the most suitable weights associated with the conflicts or criteria.

In the UiTM problem, it is important to associate appropriate weights with the weekend conflict violation and the unscheduled examination. Currently the

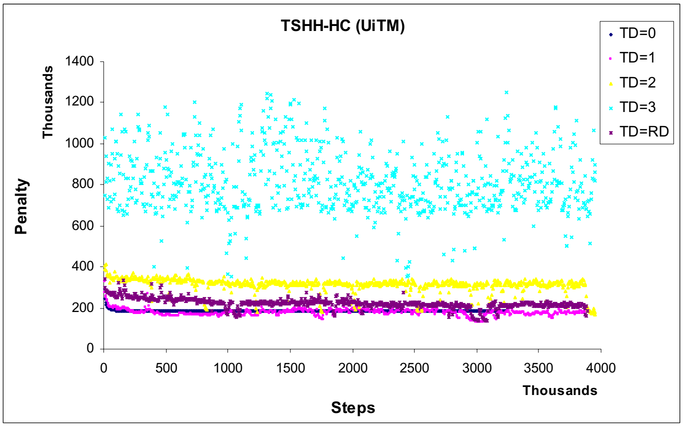


Fig. 4. Tabu-search-based hyper-heuristic with Hill Climbing for UiTM dataset

low-level heuristics are able to optimise a bi-criteria problem and can easily be extended to optimise a number of criteria (e.g. minimise usage of smaller rooms). In deciding what is an appropriate weight, we need to consider the importance of a criterion and how it will influence the search direction. A high weekend weight will push the search into a higher proximity cost since the number of examination slots will be reduced. Experiments were conducted with weights of 32 and 16 associated with a weekend conflict. Finally, a weight of 16 is chosen because a weight of 32 was too high and caused the search to become trapped and not able to improve the solution. The weekend weight is also the same (and therefore as important) as the weight for students who have examinations in adjacent periods.

The unscheduled examination weight of 5,000 that is being used with benchmark datasets is also not suitable for the UiTM dataset. Normally the unscheduled weight determines whether an examination is better being unscheduled, or scheduled and causing high proximity cost. A weight of 5,000 for an unscheduled examination is not suitable because the total number of students is too large and this will direct the search into an infeasible region since the remove examination heuristic will outperform other heuristics by un-scheduling an examination. In the UiTM problem, the unscheduled examination weight is fixed at 10,000.

UiTM Proximity and Weekend Cost Analysis. Table 4 shows the same hyper-heuristics being applied but with the different objective function. The results should be higher than in Table 2 since we are also taking into account the penalty for weekend examinations. On average, we could say that students may not have to sit for weekend examinations or sit for an adjacent examination since the best solution (cost of 7.12 by TSHH-GD with tabu duration = 3) is less than the weight for a weekend examination or adjacent examinations. However, this is not true since we do need to look at the overall timetable and count the number of students who have to sit for adjacent or weekend examinations.

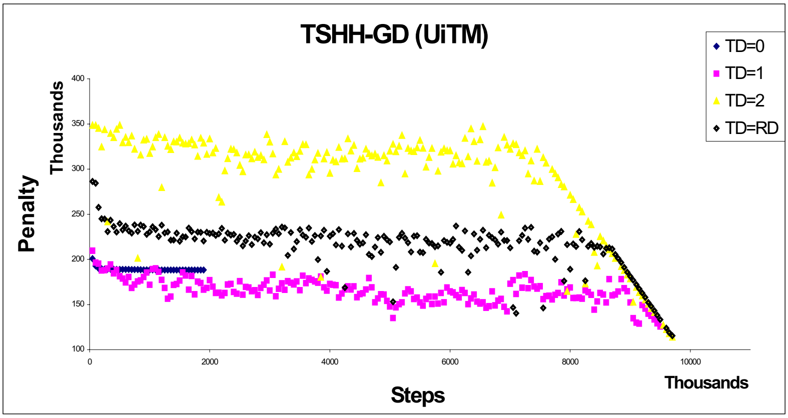


Fig. 5. Tabu-search-based hyper-heuristic with Great Deluge for UiTM dataset

Table 4. UiTM-03 dataset that minimised proximity and weekend cost

Tabu	UiTM-03		
Duration	TSHH-S	TSHH-HC	TSHH-GD
0	10.10	9.29	12.12
1	9.27	11.20	11.46
2	9.87	9.70	10.04
3	17.36	17.31	**7.12**
4	16.42	17.16	7.92
Random	9.51	9.65	8.04

Tables 5 and 6 show the number of students and percentage of students involved in a conflict that occur in two of the solutions produced. Table 5 is an analysis of the best result by the tabu search based hyper-heuristic with great deluge (TD = 3). Table 6 is an analysis of the best result by tabu search based hyper-heuristic with hill climbing (TD = 1). The columns show each of the constraints that we are minimising. The gap number indicates a gap between two examinations that a student is sitting. The results seem consistent with the objective of minimising proximity and weekend cost. The best solution has a lower cost in all of the individual constraints that we wish to minimise.

The weekend cost contributes a high percentage to the total cost in both solutions (last row in Tables 5 and 6). Therefore, it is beneficial to look at how the initial solution and the unscheduled examination weight affect the performance of the hyper-heuristic.

New Initial Solution and Dynamic Unscheduled Examination Weight. Previously, the constructive heuristic used to generate the initial solution for the

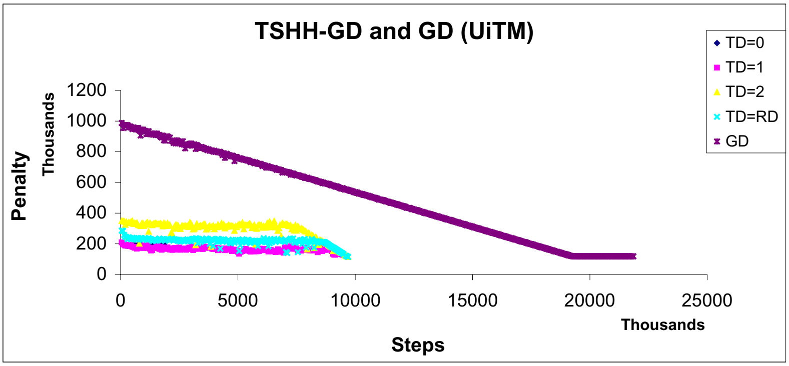


Fig. 6. Comparing TSHH-GD and Great Deluge algorithm performance

Table 5. Data characteristic of solution with 7.12 proximity and weekend cost (by TSHH-GD with TD = 3)

	Gap = 1	Gap = 2	Gap = 3	Gap = 4	Gap = 5	Weekend
Total cost	88,464	86,536	59,440	45,104	29,488	294,208
Avg cost per student	1.04	1.02	0.70	0.53	0.35	3.47
Number of students	5,529	10,817	14,860	22,552	29,488	18,388
% of total students	6.53%	12.77%	17.55%	26.63%	34.82%	21.72%
% of cost	14.66%	14.35%	9.85%	7.48%	4.89%	48.77%

Uitm-03 dataset was the same as the constructive heuristic for the benchmark datasets. With that initial solution, the hyper-heuristic is able to produce a feasible solution that is much better when compared to the existing manual solution. The constructive heuristic for benchmark datasets does not need to differentiate between weekend and non-weekend slots, and therefore does not prioritise which slots to assign an examination to. The results for Uitm-03 dataset in Tables 5 and 6 show that the weekend cost contributes almost half of the total cost. Therefore, it is worth considering prioritising non-weekend slots in the constructive heuristic, i.e. we need to inject more knowledge about the problem domain so as to produce an initial solution that starts from a reasonable point in the search space that does not violate one of the important constraints that we want to minimise. The only modification in the constructive heuristic is in determining which is the first available slot for an examination. The new constructive algorithm will only consider weekend slots if non-weekend slots are unavailable. During the search, the algorithm will also use the same criteria when moving an examination. This method should direct the search along a trajectory that avoids weekend slots.

Another issue that needs to be addressed is whether the unscheduled weight of 10,000 is suitable for the Uitm-03 dataset. In deciding which low-level heuristic

Table 6. Data characteristic of solution with 9.27 proximity and weekend cost (by TSHH-B with TD = 1)

	Gap = 1	Gap = 2	Gap = 3	Gap = 4	Gap = 5	Weekend
Total cost	172,688	115,512	69,344	53,930	30,579	343,296
Avg cost per student	2.04	1.36	0.82	0.64	0.36	4.05
Number of students	10,793	14,439	17,336	26,965	30,579	21,456
% of total students	12.75%	17.05%	20.47%	31.85%	36.11%	25.34%
% of cost	21.99%	14.71%	8.83%	6.87%	3.89%	43.71%

to select, the hyper-heuristic uses two important factors, i.e. cost of the objective function and non-tabu low-level heuristics. If the *remove examination* heuristic un-schedules an examination with a proximity cost higher than the unscheduled weight, it would seem advantageous to un-schedule the examination. The heuristics that reschedule examinations might find it difficult to improve the objective function since the new proximity cost may be greater than the un-schedule cost. This will eventually cause some examinations not being scheduled at the end of the search. The unscheduled weight does play an important role in deciding whether we need to un-schedule some examinations so that other conflicting examinations can be scheduled into the vacated slot. Therefore, instead of using a fixed unscheduled weight, we can use a dynamic un-scheduled weight that is dependent upon the number of students. Thus, in this implementation, the cost of un-scheduling an examination differs for each examination, i.e. the cost of un-scheduling an examination e_i is calculated as 60 multiplied by the number of students enrolled in examination e_i or 5,000, whichever is the maximum.

Table 7 shows a comparison of the results obtained using two different initial solutions and fixed and dynamic unscheduled weights. The old solution indicates an initial solution used in the previous experiment (Table 4). The new solution is produced using the new constructive heuristic. The last four columns in Table 7 show the results produced by TSHH-B and TSHH-HC with fixed deterministic tabu duration between 0 and 4. Each run takes one hour. Since we already have results from previous experiment (Table 4) of a four-hour run with fixed unscheduled weight and old initial solution, we did not run it again for one hour. Moreover, the one-hour run with the new parameters has already improved the results compared to the four-hour run. The TSHH-GD is not tested with this parameter because we want to observe how the dynamic unscheduled weight and new initial solution affect the final results and it should be sufficient to test the first two methods only.

From Table 7, we can conclude that with the new solution, both methods are able to produce better solutions compared to the best method (TSHH-GD) that was initialised with the old solution (Table 4). A dynamic unscheduled weight also assists the hyper-heuristic in deciding which low-level heuristic is in favour. Since the best solution is when TD equals zero, this implies that the hyper-heuristic does not even need a tabu list to help manage the low-level heuristics.

Table 7. Results with two different initial solutions and two unscheduled weight mode

		Dynamic weight		Fixed weight	
	TD	Old solution (1 h run)	New solution (1 h run)	Old solution (4 h run)*	New solution (1 h run)
Basic tabu search based hyper-heuristic (TSHH-B)	0	**6.98**	**4.40**	10.10	18.43
	1	7.67	4.91	**9.27**	20.60
	2	10.23	6.64	9.88	**7.16**
	3	16.99	8.73	17.36	8.67
	4	16.75	15.11	16.42	15.43
Tabu search based hyper-heuristic with hill climbing (TSHH-HC)	0	**6.77**	**4.25**	**9.29**	16.64
	1	7.70	6.01	11.20	16.88
	2	10.12	5.93	9.70	**6.94**
	3	16.91	8.95	17.31	9.94
	4	16.30	14.74	17.16	16.09

* As Table 4.

Table 8. Data characteristic of solution with 4.40 proximity and weekend cost. Best solution for TSHH-B with new initial solution and dynamic weight.

	Gap = 1	Gap = 2	Gap = 3	Gap = 4	Gap = 5	Weekend
Total cost	80,672	73,984	58,820	50,476	30,881	78,064
Avg cost per student	0.95	0.87	0.69	0.60	0.36	0.92
Number of students	5,042	9,248	14,705	25,238	30,881	4,879
% of total students	5.95%	10.92%	17.37%	29.81%	36.47%	5.76%
% of cost	21.63%	19.84%	15.77%	13.54%	8.28%	20.93%

Table 9. Data characteristic of solution with 6.98 proximity and weekend cost. Best solution for TSHH-B with old initial solution and dynamic weight.

	Gap = 1	Gap = 2	Gap = 3	Gap = 4	Gap = 5	Weekend
Total cost	91,360	89,440	55,808	47,600	29,472	277,440
Avg cost per student	1.08	1.06	0.66	0.56	0.35	3.28
Number of students	5,710	11,180	13,952	23,800	29,472	17,340
% of total students	6.74%	13.20%	16.48%	28.11%	34.81%	20.48%
% of cost	15.46%	15.13%	9.44%	8.05%	4.99%	46.93%

Tables 8–10 present a detailed analysis of the best solution with different parameters. With the new initial solution that avoids weekend slots, the hyper-heuristic is able to improve the weekend conflicts, implying that with more knowledge about the problem domain injected into the initial solution construction phase and the low-level heuristics, the hyper-heuristic can perform better.

Table 10. Data characteristic of solution with 6.94 proximity and weekend cost. Best solution for TSHH-HC with new initial solution and fixed weight.

	Gap = 1	Gap = 2	Gap = 3	Gap = 4	Gap = 5	Weekend
Total cost	135,360	92,664	89,080	58,090	32,723	89,768
Avg cost per student	1.60	1.09	1.05	0.69	0.39	1.06
Number of students	8,460	11,583	22,270	29,045	32,723	5,611
% of total students	9.99%	13.68%	26.30%	34.30%	38.65%	6.63%
% of cost	27.20%	18.62%	17.90%	11.67%	6.58%	18.04%

With this good initial solution and a fixed unscheduled weight, we can also get a reasonable result compared to solutions with a dynamic unscheduled weight. As a general method and without the extra knowledge, the hyper-heuristic can also produce competitive results (compared to a manual solution) for the UiTM dataset specification (as shown in Table 3).

5 Conclusions

Collecting data from a large university is a difficult task, but with the help of a distributed network and a frequent update of student information, we can facilitate the automated process of producing examination schedules that are feasible and may satisfy everybody involved. The Uitm-03 dataset is a large real-world examination timetabling problem. Its data may not exactly reflect all the required information such as validated student data and implicit constraints, however, other crucial data such as the actual timetable used and the list of examinations to be scheduled are consistent with the real problem. All data inconsistencies were removed prior to processing so that the dataset is as close as possible to the actual problem. The important constraints are considered in the search engine with a flexibility of adding additional constraints if necessary.

Tabu-search-based hyper-heuristics have been shown to produce feasible and good quality solutions on other benchmark datasets [27]. The same hyper-heuristic framework was tested with the Uitm-03 dataset. An additional criterion of minimising the weekend cost is added to the objective function and with no modification to the hyper-heuristic framework it is able to produce good quality solutions. It is obvious that the nature of the Uitm-03 dataset is different from the benchmark datasets because of an additional objective criteria, a larger number of students and the maximum number of examinations in conflict is high compared to the average number of examinations in conflict. Thus, by adding such knowledge in the form of a new constructive heuristic in the initialisation phase, prioritising non-weekend slots and dynamic unscheduled examination weights, the hyper-heuristic framework is able to produce a much better solution. It is such knowledge that differentiates problem instances and as such the hyper-heuristic is a general method that was shown to be effective on benchmark datasets and also on the UiTM problem.

Currently, the tabu search hyper-heuristic method is dependent upon the tabu duration value. Future work will investigate an adaptive tabu strategy of selecting the best tabu duration at each decision point. This method should free the hyper-heuristic from the problems of parameter tuning.

Acknowledgements

This research was carried out at the University of Nottingham and supported by the Malaysian Public Services Department and MARA University of Technology (UiTM), Malaysia.

References

1. Burke, E. K., Newall, J. P.: Solving Examination Timetabling Problems Through Adaption of Heuristic Orderings. Ann. Oper. Res. **129** (2004) 107–134
2. Burke, E. K., Newall, J. P.: Enhancing Timetable Solutions with Local Search Methods. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 195–206
3. Burke, E. K., Petrovic, S.: Recent Research Directions in Automated Timetabling. Eur. J. Oper. Res. **140** (2002) 266–280
4. Burke E. K., Landa Silva, J. D., Soubeiga, E.: Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling. In: Ibaraki T., Nonobe K., Yagiura M. (eds.): Meta-heuristics: Progress as Real Problem Solvers. Springer, Berlin (2005)
5. Burke, E. K., Meisels, A., Petrovic, S., Qu, R.,A: Graph-Based Hyper-heuristic for Timetabling Problems. Eur. J. Oper. Res. (2005) accepted for publication
6. Burke, E. K., Petrovic, S. and Qu, R.: Case Based Heuristic Selection for Timetabling Problems. J. Scheduling (2005) accepted for publication
7. Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-Heuristics: An Emerging Direction in Modern Search Technology. Chapter 16 in: Glover, F., Kochenberger, G. (eds.): Handbook of Meta-Heuristics. Kluwer, Dordrecht (2003) 457–474
8. Burke, E. K., Jackson, K. S., Kingston, J. H., Weare, R. F.: Automated Timetabling: The State of the Art. Comput. J. **. 40** (1997) 565–571
9. Burke, E. K., Kendall, G., Soubeiga, E. (2003b) A Tabu-Search Hyperheuristic for Timetabling and Rostering. J. Heuristics **9** (2003) 451–470
10. Burke, E. K., Elliman, D. G., Ford, P. H., Weare, R. F.: Examination Timetabling in British Universities—A Survey. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 76–92
11. Burke, E. K., Newall, J. P., Weare, R.: Initialisation Strategies and Diversity in Evolutionary Timetabling. IEEE Trans. on Evol. Comput. **6** (1998) 81–103
12. Burke, E. K., Newall, J. P., Weare, R. F.: A Memetic Algorithm for University Exam Timetabling. In: Burke, E., Ross, P. (eds.): PATAT 1995—Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling 496–503

13. Burke, E. K., Bykov, Y., Newall, J., Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Trans. on Oper. Eng. **36** (2004) 509–528
14. Caprara, A., Fischetti, M., Guida, P. L., Monaci, M., Sacco, G., Toth, P.: Solution of Real-World Train Timetabling Problems. Proceedings of the 34th Annual Hawaii International Conference on System Sciences (2001) 1057–1066
15. Carter, M. W.: A Survey of Practical Applications of Examination Timetabling Algorithms. Oper. Res. Soc. Am. **34** (1986) 193–202
16. Carter, M. W.: EXAMINE: A General Examination Timetabling System. In: Burke, E. K., Carter, M. (eds.): PATAT 1997—Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling 363
17. Carter, M. W., Laporte, G.: Recent Developments in Practical Examination Timetabling. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 3–21
18. Carter, M. W., Laporte, G., Lee, S. Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
19. Comm, C. L., Mathaisel, D. F. X.: College Course Scheduling. A Market for Computer Software Support. J. Res. Comput. Educ. **21** (1988) 187–195
20. Cowling, P., Kendall, G., Mohd Hussin, N.: A Survey and Case Study of Practical Examination Timetabling Problems. In: Burke, E., De Causmaecker, P. (eds.): PATAT 2002—Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling 258–261
21. Di Gaspero, L., Schaerf, A.: A Tabu Search Techniques for Examination Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 104–117
22. Dueck, G.: New Optimization Heuristics for the Great Deluge Algorithm and the Record-to-Record Travel. J. Comput. Phys. **104** (1993) 86–92
23. Forster, G.: Syllabus Plus: A State-of-the-Art Planning and Scheduling System for Universities and Colleges. In: Burke, E., Ross, P. (eds.): PATAT 1995—Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling 244–252
24. Goltz, H.-J., Matzke D.: ConBaTT—Constraint-Based Timetabling. In: Burke, E. K., Erben, W. (eds.): PATAT 2000—Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling 491
25. Isaai, M. T., Singh, M. G.: Hybrid Applications of Constraint Satisfaction and Meta-heuristics to Railway Timetabling: A Comparative Study. IEEE Trans. on Systems, Man and Cybernetics, Part C: Applications and Reviews. **31** (2001) 87–95
26. JTAP: Central Timetabling By Computer: A review of Existing Information. Report by JISC Technology Applications Programme, June (1998). The report can be downloaded from url:http://www.jisc.ac.uk/uploaded_document/jtap-021.doc
27. Kendall G., Mohd Hussin N.: An Investigation of a Tabu Search Based Hyperheuristic for Examination Timetabling. In: Kendall, G.. et al. (eds.): Multidisciplinary Scheduling: Theory and Applications. Springer, Berlin (2005) 309–328. An extended abstract of this paper also appeared in the Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'03) 226–233

28. McCollum, B., Newall, J.: Introducing Optime: Examination Timetabling Software. In: Burke, E. K., Erben, W. (eds.): PATAT 2000—Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling 485
29. Merlot, L. T. G., Boland, N, Hughes, B. D., Stuckey, P. J.: A Hybrid Algorithm for the Examination Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 207–231
30. Petrovic, S., Burke, E. K.: University Timetabling. Chapter 45 in: Leung, J. (ed.): The Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton, FL (2004)
31. Petrovic, S., Bykov, Y.: A Multiobjective Optimisation Technique for Exam Timetabling Based on Trajectories. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 181–194
32. Rogalla, S.: CELCAT: A Practical Solution to Scheduling Problems, Corbett Engineering, UK. In: Proceedings of The Practice and Theory of Automated Timetabling II (PATAT'98) 368
33. Schaerf, A.: A Survey of Automated Timetabling. Artif. Intell. Rev. **13** (1999) 87–127
34. Terashima-Marín, H., Ross, P. M., Valenzuela-Rendón, M.: Evolution of Constraint Satisfaction Strategies in Examination Timetabling. In: Proceedings of the Genetic and Evolutionary Conference (1999) 635–642
35. Trick, M. A.: A Schedule-then-Break Approach to Sports Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 242–253
36. Wan Ya, Baharudin, N. Interview with Manager and System Analyst. Examination Unit, Center for Integrated Information System, MARA University of Technology, August 2001
37. White, G. M., Xie, B. S.: Examination Timetables and Tabu Search with Longer Term Memory. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 85–103
38. Yang, Y., Petrovic, S.: A Novel Similarity Measure for Heuristic Selection in Examination Timetabling. In: Burke, E. K., Trick, M. (eds.): Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (2004) 377–396

A Hybrid Multi-objective Evolutionary Algorithm for the Uncapacitated Exam Proximity Problem

Pascal Côté, Tony Wong, and Robert Sabourin

Department of Automated Manufacturing Engineering,
École de technologie supérieure, Université du Québec,
1100 Notre-Dame, Montréal (Québec), Canada
{tony.wong, robert.sabourin}@etsmtl.ca

Abstract. A hybrid Multi-Objective Evolutionary Algorithm is used to tackle the uncapacitated exam proximity problem. In this hybridization, local search operators are used instead of the traditional genetic recombination operators. One of the search operators is designed to repair unfeasible timetables produced by the initialization procedure and the mutation operator. The other search operator implements a simplified Variable Neighborhood Descent meta-heuristic and its role is to improve the proximity cost. The resulting non dominated timetables are compared with those produced by other optimization methods using 15 public domain datasets. Without special fine-tuning, the hybrid algorithm was able to produce timetables with good rankings in nine of the 15 datasets.

1 Introduction

This paper presents a hybrid Multi-Objective Evolutionary Algorithm (MOEA) designed for the uncapacitated exam proximity problem in which a timetable has to offer student maximum free time between exams while satisfying the clashing constraint (exam conflicts) and without regard to the seating capacity. The proposed multi-objective approach also considers timetable length as an optimization objective. It is thus possible to generate a set of alternative solutions without multiple execution of the optimization process. The hybridization is inspired by Radcliffe and Surry's Memetic Algorithm (MA) [22]. Its structure is comparable to other modern hybrid evolutionary timetabling algorithms as described in Silva et al. [23]. In the basic MA, local search operators are added to the genetic recombination and mutation operators and local optimization is performed following the genetic reproduction phase. To obtain a reasonable computation requirement, the local search operators are usually implemented as greedy hill climbers. It is possible to introduce more sophisticated local search heuristics but the optimization response time will increase as a function of search complexity. A way to incorporate advanced local search heuristics while maintaining acceptable computation time is to remove the genetic recombination operator from the

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 294–312, 2005.

MA. The genetic recombination can be viewed as an exploitation strategy where the search focuses on neighbors of good solutions. A local search heuristic can play the same exploitative role in exam timetabling problems.

This paper is organized as follows. Section 2 describes the problem model including the clashing constraint (exam conflicts). Section 3 presents a brief survey of previous methods. This survey is restricted to research carried out on the datasets provided by Carter et al. [6], Burke et al. [4] and Merlot et al. [15]. Section 4 explains the multi-objective approach investigated in this work. Section 5 details the results, and the conclusions follow in Section 6.

2 Problem Description

Given a set of exams $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$ and a set of timeslots $\mathcal{T} = \{1, 2, \ldots, |\mathcal{T}|\}$, the goal of examination timetabling is to obtain an assignment where each exam in $\mathcal{E}$ is allocated to a timeslot in $\mathcal{T}$. The result of such an assignment is a timetable represented here by a set h of ordered couples (t, e) where $t \in \mathcal{T}$ and $e \in \mathcal{E}$. A timetable h is called feasible if it satisfies all required constraints. Otherwise, h is identified as unfeasible. A fundamental requirement in exam timetabling is to prohibit clashing, or exam conflicts (a student having to take two or more exams in a given timeslot). In this work, clashing is a hard constraint and can be expressed as

$$\sum_{k=1}^{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{E}|} \sum_{j=1}^{|\mathcal{E}|} \eta_{ij} \varepsilon_{ik} \varepsilon_{jk} = 0\,. \tag{1}$$

In (1), η_{ij} is the number of students taking exam e_i and exam e_j, $\varepsilon_{jk} \in \{0, 1\}$ is a binary quantity with $\varepsilon_{jk} = 1$ if exam e_j is assigned to timeslot k. Otherwise, $\varepsilon_{jk} = 0$. A timetable is feasible if (1) is satisfied.

The basic examination timetabling problem is to minimize the number of timeslots used in a feasible timetable. This minimization problem is defined as

$$\begin{aligned} &\text{minimize } u_1 = |\mathcal{T}|\,, \\ &\text{s.t.} \qquad \sum_{k=1}^{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{E}|} \sum_{j=1}^{|\mathcal{E}|} \eta_{ij} \varepsilon_{ik} \varepsilon_{jk} = 0\,. \end{aligned} \tag{2}$$

Note that (2) is equivalent to the graph-coloring problem. A more elaborate problem is the exam proximity problem (EPP). A practical timetable should allow students to have more free time between exams. Thus, the objective of the EPP is to find a feasible timetable while minimizing the number of students having to take consecutive exams. Equation (3) is a variant of the EPP model where q is the number of timeslots per day, $N \geq 0$ is the number of free timeslots between exams and K is a constant representing the maximum timetable length. That is,

$$\text{minimize } u_2 = \frac{1}{2} \sum_{k=1}^{|\mathcal{T}|-(N+1)} \sum_{i=1}^{|\mathcal{E}|} \sum_{j=1}^{|\mathcal{E}|} \eta_{ij} \varepsilon_{ik} \varepsilon_{jk+(N+1)}\,, \forall k \text{ where k mod q} \neq 0\,,$$

$$\text{s.t.} \sum_{k=1}^{|\mathcal{T}|}\sum_{i=1}^{|\mathcal{E}|}\sum_{j=1}^{|\mathcal{E}|} \eta_{ij}\varepsilon_{ik}\varepsilon_{jk} = 0\,, \quad |\mathcal{T}| \leq K\,. \tag{3}$$

The above model represents an Uncapacitated Exam Proximity problem (UEPP) because it does not take into account classroom seating capacity.

3 Previous Methods

The UEPP has been investigated by many researchers. However, the problem formulation and enrollment data are often defined by the environment and requirements of a particular institution. As a result, many methods and algorithms have been proposed to solve particular instances of the UEPP.

This section surveys previous solution methods applied to a collection of publicly available datasets. The datasets used in this work are from Carter et al. [6], Burke et al. [4] and Merlot et al. [15]. They contain actual enrollment data taken from several universities and academic institutions. A common proximity metric has also been defined for the datasets which is a weighted version of (3) with $0 < N \leq 4$ (counting the number of students having 0–4 free timeslots between exams). This proximity metric can be expressed using (3) as follows:

$$\begin{aligned}
&f = \frac{\sum_{x=0}^{4} w_{i+1}\, u_2\big|_{q=|\mathcal{T}|,N=x}}{N_s}\,,\\
&u_2\big|_{q=|\mathcal{T}|,N=x} = \tfrac{1}{2} \sum_{k=1}^{|\mathcal{T}|-(x+1)} \sum_{i=1}^{|\varepsilon|}\sum_{j=1}^{|\varepsilon|} \eta_{ij}\varepsilon_{ik}\varepsilon_{jk+(x+1)}\,,\\
&\forall k \text{ where } k \bmod |\mathcal{T}| \neq 0\,.
\end{aligned} \tag{4}$$

In the above equation w_i are are the weighting factors, N_s is the total student enrollment and $u_2\big|_{q=|\mathcal{T}|,N=x}$ means computing the objective function using $q = |\mathcal{T}|$ and N varies from $x = 0$ to $x = 4$. In (4), the timeslots are numbered contiguously with no overnight gap. The weighting factors were proposed by Carter et al. [6]. They are $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$, and $w_5 = 1$. Thus, (4) can be viewed as the average proximity cost of a given timetable and the resulting UEPP is

$$\begin{aligned}
&\text{minimize } f = \sum_{i=0}^{4} w_{i+1}\, u_2\big|_{q=|\mathcal{T}|,N=i} \Big/ N_s\,;\\
&\text{s.t.} \qquad \sum_{k=1}^{|\mathcal{T}|}\sum_{i=1}^{|\mathcal{E}|}\sum_{j=1}^{|\mathcal{E}|} \eta_{ij}\varepsilon_{ik}\varepsilon_{jk} = 0\,,\\
&\qquad\qquad |\mathcal{T}| \leq K\,.
\end{aligned} \tag{5}$$

Early solution techniques were derived from sequential graph coloring heuristics. These heuristics attempt to assign each exam to a timeslot according to some ordering schemes. Carter et al. [6] successfully applied a backtracking sequential assignment algorithm to produce feasible timetables for the UEPP. The backtracking feature enables the algorithm to undo previous assignments and thus escape from cul-de-sacs. In all, 40 different strategies have been implemented. The results showed that the effectiveness of the sequential assignment algorithm is related to the ordering scheme and the nature of the datasets. Note that the backtracking sequential assignment is a deterministic algorithm. This means that, for a given dataset and ordering scheme, it will always produce the same timetable.

In Burke et al. [4], an initial pool of timetables is generated by grouping together exams with similar sets of conflicting exams. Then timetables are randomly selected from the pool, weighted by their objective value, and mutated by rescheduling randomly chosen exams. Hill climbing is then applied to the mutated timetable to improve its quality. The process continues with the new pool of timetables. Caramia et al. [5] developed a set of heuristics to tackle the UEPP with excellent results. First, a solution is obtained by a greedy assignment procedure. This procedure selects exams based on a priority scheme which gives high priority to exams with high clashing potential. Next, a spreading heuristic is applied to decrease the proximity penalty of the solution without lengthening the timetable. However, if the spreading heuristic failed to provide any penalty decrease then another heuristic is applied to decrease the proximity penalty by adding one extra timeslot to the solution. These heuristics are reapplied until no further improvement can be found. A perturbation technique is also described in which the search process is restarted by resetting the priority and proximity penalty.

The proximity problem was also investigated by Di Gaspero and Schaerf [10]. Their approach starts with a greedy heuristic to assign timeslots to all exams having no common students. The remaining unassigned exams are distributed randomly to different timeslots. The solution obtained is then improved by a tabu search algorithm using a short-term tabu list with random tabu tenure. The search neighborhood is defined as the set of exams that can be moved from one timeslot to another without violating the constraints. A further reduction of the neighborhood is obtained by using the subset of exams currently in constraint violation. To improve the proximity cost, Di Gaspero and Schaerf also implemented the shifting penalty mechanism from Gendreau et al. [14].

A Tabu Search algorithm (called OTTABU) with a recency-based and a frequency-based Tabu list was implemented by White and Xie [25]. An initial solution is first generated by a bin-packing heuristic ("largest enrollment first"). If the initial solution is unfeasible, then a Tabu Search is executed to remove all constraint violations using the set of clashing exams as neighborhood. Another Tabu Search is used to improve the quality of the feasible solution. This time, the neighborhood is the set of exams that can be moved from one timeslot to another without causing any clashes. White and Xie also devised an estimation technique for the Tabu tenure based on enrollment, the number of exams having the same pool of students and the number of students taking the same exams. More recently, Paquete and Stutzle [20] considered the UEPP by casting the constraints as part of an aggregated objective function. The search process is prioritized and is realized by the use of a Tabu Search algorithm with a short-term Tabu list and random tenure. The 1-opt neighborhood is defined by the subset of exams with constraint violations.

A three-stage approach using constraint programming, simulated annealing and hill climbing was proposed by Merlot et al. [15]. An initial timetable is generated by constraint programming. The resulting timetable is then improved by a simulated annealing algorithm using the Kempe chain neighborhood [17] and a slow cooling schedule. In the last stage, a hill climbing procedure is applied to further improve the final timetable. The GRASP meta-heuristic [12] was also used to solve the UEPP. Casey and Thompson [7] used a probabilistic version of the sequential assignment algorithm from Carter et al. to realize the construction phase of GRASP. In the improvement phase of GRASP, they ordered the exams according to their contribution to the objective value. Then, for each exam, a timeslot is found such that the objective value is decreased. The construction and improvement phases are restarted with a blank timetable a number of times and the best timetable is kept.

Burke and Newall [3] investigated the effectiveness of the local search approach to improve the quality of timetables. In their work, an adaptive technique is used to modify a given heuristic ordering for the sequential construction of an initial solution. They then compared the average and peak improvement obtained by three different search algorithms: Hill Climbing, Simulated Annealing and an implementation of the Great Deluge algorithm [11]. The reported results indicated that the Adaptive Heuristics and Great Deluge combination provided significant enhancement to the initial solution.

A fuzzy inference approach was developed by Asmuni et al. [1] to verify the effectiveness of multiple ordering in the sequential construction of exam timetables. To construct a timetable, exams are first scheduled sequentially according to an ordering scheme. However, some of the exams may remain unscheduled after this step. For the unscheduled exams, a fuzzy expert system is used to determine a new ordering. A modified backtracking algorithm is then executed to assign timeslots to the unscheduled exams according to the new ordering. This second step is repeated until all the exams are scheduled. As described in [1], the fuzzy expert system has two inputs taken from different combination of the following heuristic ordering criteria:

1. Largest degree first (LDF),
2. Largest enrollment first (LEF),
3. Greatest available timeslot first—saturation degree (SDF).

Fuzzification of the input variables resulted in a fuzzy degree of membership in the qualifying linguistic set (i.e. small, medium and high). These fuzzified inputs are then related to the output by a set of `if-then` rules. Since the rules may have several connectives (`AND, OR`), the standard min–max operators are used to deal with fuzzy inference. Finally, the centroid defuzzification technique is carried out to obtain a single crisp output value. This crisp output value represents the order of a given exam. The results given in [1] indicated that the fuzzy inference approach provided better proximity cost for several datasets than the traditional single-ordering scheme [6].

4 Multi-objective Approaches

Multi-objective ETP is a more general and flexible formulation than its single-objective counterpart. The following is a brief summary of three different multi-objective strategies investigated by researchers. Further details on multi-objective timetabling metaheuristics can be found in Silva et al. [23].

The method of compromise programming is used by Burke et al. [2] to seek a timetable which is minimally located from a given reference point. In their work, the reference point is obtained by generating good quality timetables in terms of each objective using the saturation degree ordering scheme. While the distance between the current timetables and the reference point is measured by the Euclidean distance. The resulting timetables are then improved by an iterative algorithm combining two variation operators: hill-climbing and mutation. Burke et al. were able to generate good quality timetables for the NOT-F-94 dataset with nine different objectives including the seating capacity.

Another interesting multi-objective approach is the one described by Petrovic and Bykov [21]. They developed a guided multi-objective optimization technique using a reformulated Great Deluge algorithm [11] and a previously generated reference timetable. The guidance is provided by defining a curve extending from the origin of the multi-dimensional objective space through the reference timetable's objective values. To drive the search along the predefined curve, the authors incorporated a variable weighting procedure within the Great Deluge algorithm—each objective function is now associated with a weighting factor. The search algorithm selects the largest objective value of the current timetable and increases its weight. This causes the Great Deluge algorithm to lower the acceptance level of the corresponding objective. The algorithm continues with its weighting adjustment, timetable generation and acceptance until the maximum number of iterations is reached. This approach resulted in high-quality timetables for several datasets using nine different objectives including the seating capacity (same as [2]).

A computational analysis involving multi-objective evolutionary algorithms is given by Paquete and Foncesa for the general examination timetabling problem [19]. They compared the effectiveness of several evolutionary operators using Foncesa and Fleming's constrained multi-objective evolutionary framework [13]. For the problem encoding, they implemented a direct representation where each position in the chromosome corresponds to an exam. Their analysis showed that the Pareto-ranking technique performed better than the linear-ranking technique. Also, when time constraint is considered, independent mutation of each chromosome position outperforms single-position mutation.

4.1 Hybrid MOEA Implementation

As shown in Section 3, the UEPP is traditionally treated as a single objective combinatorial optimization problem. The timetable length is chosen a priori and is part of the constraint set. In the context of resource planning, it is often desirable to assess the impact of timetable length on the proximity cost. A timetable

```
Procedure HMOEA
$\mathcal{P}^{(t)}$: population at iteration $t$
$\mathcal{R}^{(t)}$: intermediate population at iteration $t$
$\mathrm{L}_1, \mathrm{L}_2$ : local search operators
U : archive update procedure
$\mathrm{M}_{\alpha_m}$: uniform mutation operator with mutation rate $\alpha_m$
S: constrained dominance binary tournament operator
OUTPUT
$\mathcal{Q}^{(t)}$: archive containing non dominated timetables

Initialize $\mathcal{P}^0$ and $\mathcal{Q}^0$ of size $N$ with random timetables
For each iteration $t \leftarrow 0, 1, \ldots, I_{\max}$ do
    // Step 1) Apply local searches to the combined population
    $\mathcal{R}^{(t)} \leftarrow \mathrm{L}_2(\mathrm{L}_1(\{\mathcal{P}^{(t)} \cup \mathcal{Q}^{(t)}\}))$
    // Step 2) Compute ranking for the resulting timetables
    $\mathrm{F}(h), \forall h \in \mathcal{R}^{(t)}$
    // Step 3) Update archive
    $\mathcal{Q}^{(t+1)} \leftarrow \mathrm{U}(\mathcal{R}^{(t)})$
    // Step 4) Create new population by mutation and selection
    $\mathcal{P}^{(t+1)} \leftarrow \mathrm{S}(\mathrm{M}_{\alpha_m}(\mathcal{R}^{(t)}))$
End for
```

Fig. 1. Working principle of the proposed hybrid MOEA

length versus proximity cost assessment can also provide the planner with compromise solutions to the timetabling problem. Equation (6) is a bi-objective formulation capable of realizing such an assessment:

$$\begin{aligned} \text{minimize } & f_1 = |\mathcal{T}| \;, \\ & f_2 = \sum_{i=0}^{4} w_{i+1} \; u_2\big|_{q=|\mathcal{T}|, N=i} \Big/ N_s \;, \\ \text{s.t.} \quad & \sum_{k=1}^{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{E}|} \sum_{j=1}^{|\mathcal{E}|} \eta_{ij} \varepsilon_{ik} \varepsilon_{jk} = 0 \,. \end{aligned} \tag{6}$$

Now the task is to find a feasible timetable while minimizing timetable length and proximity cost simultaneously.

The proposed hybrid MOEA is a Pareto-based optimization heuristic which uses an auxiliary population (archive) to maintain the best non-dominated solutions. Each potential solution in the population is a timetable (feasible or unfeasible). The timetables are assigned a rank based on the objective functions f_1 (timetable length) and f_2 (proximity cost). A special feature in the proposed hybrid MOEA is the substitution of the recombination operator by two local search operators. Local search algorithms are used here to remove constraint violations and to improve the proximity cost. The following pseudo-code explains the operating principle of the algorithm.

In Figure 1, the main population at iteration t is denoted by $\mathcal{P}^{(t)}$ and the archive by $\mathcal{Q}^{(t)}$. Both $\mathcal{P}^{(t)}$ and $\mathcal{Q}^{(t)}$ contain N timetables and their size remains constant during the optimization process. The initial timetables are generated randomly without regard to their feasibility. The first local search operator L_1 is used to remove constraint violations, while the second local search operator L_2 is used to decrease the proximity cost. The timetables produced by L_1 and L_2 form a combined intermediate population $\mathcal{R}^{(t)}$ of $2N$ timetables. Next, a ranking value is computed for each timetable in $\mathcal{R}^{(t)}$ using Zitzler's Pareto Strength concept [26]. The non-dominated timetables are then inserted into the archive using an archive update rule. Finally, each timetable in the intermediate population is mutated with probability α_m. Since there are $2N$ timetables in $\mathcal{R}^{(t)}$, N timetables are discarded from $\mathcal{R}^{(t)}$ using the constrained tournament selection technique [18]. The remaining N timetables form the new population $\mathcal{P}^{(t+1)}$, and the evolution process continues for $I_{\max}$ iterations.

4.2 Population and Archive Initialization

The same initialization procedure is applied to the main population $\mathcal{P}^{(t)}$ and the archive $\mathcal{Q}^{(t)}$. Both $\mathcal{P}^{(t)}$ and $\mathcal{Q}^{(t)}$ can each contain N timetables and are divided into β slots $l_0 > l_1, \ldots, > l_\beta$ representing different timetable lengths. For each slot i, N/β random timetables with length l_i are generated. The number of slots and the range of the timetable length are determined according to the published results available for the datasets. Note that the initialization procedure will also produce unfeasible timetables. These unfeasible timetables with be repaired with the help of local search operators. This is explained in more detail in the next section.

4.3 Search Operators $\mathbf{L_1}$ and $\mathbf{L_2}$

In the hybrid MOEA implementation, local search operators are used instead of the traditional genetic recombination operators. This hybridization scheme enables the evolutionary process focus better on the optimization task. Both local searches L_1 and L_2 are in fact Tabu Search algorithms. The search operator L_1 implements a classic Tabu Search using a simple 1-opt neighborhood. This neighborhood is defined by an ordered triple (e, t_i, t_j), where e is an exam in schedule conflict with at least one other exam, and $t_i \neq t_j$ are two different timeslots such that e can be moved from t_i to t_j without creating a new conflict. The idea is to decrease the number of constraint violations for the timetables currently in the main population $\mathcal{P}^{(t)}$ and in the archive $\mathcal{Q}^{(t)}$.

In order to improve the proximity cost of the timetables, the search operator L_2 implements a simplified version of the VND (Variable Neighborhood Descent) meta-heuristic [16]. Two neighborhoods, the Kempe chain interchange [17,24] and 1-move, are used in L_2 with Tabu Search as the search engine. In the simplified VND, local searches are executed in $n > 1$ neighborhoods sequentially. The initial solution of the current search is the best solution obtained from the previous search. Since there are $n > 1$ neighborhoods involved, it is conjectured that

VND has better search space coverage than do single neighborhood search techniques [16]. In our implementation, the Kempe chain interchange neighborhood is used first. Similar to the 1-opt neighborhood, a Kempe chain is defined by an ordered triple (e, t_0, t_1), where exam e is assigned to timeslot t_0 and $t_0 \neq t_1$. However, exam e is now selected by sampling the set of exams. Consider a graph G where the vertices are the exams and an edge exists between vertices e_i and e_j if at least one student is taking exams e_i and e_j. Each vertex in G is labeled with the exam's assigned timeslot, and an edge linking two exams indicates a potential clashing situation. A Kempe chain (e, t_0, t_1) is a connected subgraph induced by a subset of linked exams assigned to timeslots t_0 and t_1. The subset of linked exams must also contain the exam e. In other words, it is the subset of exams reachable from e in the digraph D given by

$$\begin{aligned} V(D) &= \{V_{t_0}\} \cup \{V_{t_1}\}, \\ E(D) &= \left\{(u, w) : (u, w) \in E(G), u \in V_{t_i} \wedge w \in V_{t_{(i+1) \bmod 2}}\right\}, \end{aligned} \tag{7}$$

where $E(G)$ represents the set of edges in graph G and V_{t_i} is the subset of exams assigned to timeslot t_i that are reachable from exam e. Thus, a Kempe chain interchange is the relabeling of each chain vertex in timeslot t_0 to timeslot t_1, and vice versa. This relabeling is conflict-free if the original timetable is also conflict-free. It is also applicable to unfeasible timetables.

In the Tabu Search implementation, we choose N_k Kempe chains and apply the best chain as the current move. To sample a chain, we choose two linked exams randomly without replacement from the set of exams and use their timeslots as t_0 and t_1. One disadvantage of the Kempe chain interchange neighborhood is that the number of useful chains decreases as the search progresses toward a local optimum [17]. To avoid this pitfall, we use another neighborhood to explore the search space. After N_I iterations without improvement by the Kempe chain interchange, we start another Tabu Search using the 1-move neighborhood. To select a move from the 1-move neighborhood, we sample N_m legal moves (moving one exam from its assigned timeslot to another timeslot without creating constraint violations) and the one with the best proximity cost is selected. The initial timetable for the 1-move neighborhood search is the current best timetable.

As shown in Figure 2, $f(\cdot)$ represents the proximity cost, $\mathrm{TS}_{\mathrm{kci}}$ designates a Tabu Search with the Kempe chain interchange neighborhood and $\mathrm{TS}_{1-\mathrm{move}}$ indicates the one using a 1-move neighborhood. For a given timetable, the search terminates when no further improvement can be obtained by $\mathrm{TS}_{\mathrm{kci}}$ and $\mathrm{TS}_{1-\mathrm{move}}$.

4.4 Ranking Computation

The timetables in the combined intermediate population $\mathcal{R}^{(t)}$ are to be ranked in order to determine their quality relative to the current population. The ranking computation process assigns a numerical value to each timetable according to their dominance performance in the current population [8]. An efficient ranking

```
Operator $\mathrm{L}_2(H)$
$f(\cdot)$: proximity cost
$h$: current best timetable
INPUT
$H$: set of timetables in the current population and in the archive,
    $H \equiv \mathcal{P}^{(t)} \cup \mathcal{Q}^{(t)}$.
OUTPUT
$\mathcal{R}^{(t)}$: combined intermediate population

For each $h \in H$ do,
    while true,
      // Apply Tabu Search with Kemp chain interchange
      // Stopping criterion: $N_I$ iterations
      $h' \leftarrow \mathrm{TS}_{\mathrm{Kci}}(h)$
      // $h'$  is better than $h$ ?
      If $f_2(h') > f_2(h)$,
        // No. Apply Tabu Search to $h$ using 1-move neighborhood
        $h' \leftarrow \mathrm{TS}_{1-\mathrm{move}}(h)$
        // $h'$ is better than $h$ ?
        If $f_2(h') > f_2(h)$,
           // No. Exit While loop and process next timetable
           next
        End If
      End If
      // update current best timetable
      $h \leftarrow h'$
    End While
    $\mathcal{R}^{(t)} \leftarrow \{\mathcal{R}^{(t)}\} \cup \{h\}$
End Do
```

Fig. 2. Search operator L_2 implements a simplified VND meta-heuristic

procedure is the one based on the Pareto Strength concept used in the SPEA-II multi-objective evolutionary algorithm [26]. In this procedure, a timetable's Pareto strength $\mathrm{C}(\cdot)$ is the number of timetables it dominates in the combined intermediate population. That is,

$$\mathrm{C}(h_i) = \left| \left\{ h_j : h_j \in \mathcal{R}^{(t)} \wedge h_i \succ h_j \right\} \right| , \tag{8}$$

where the symbol $\succ$ corresponds to the Pareto dominance relation. For a P objective minimization problem with objective functions $f_1, f_2, \ldots, f_p$, a timetable h_1 is said to dominate another timetable h_2, denoted here by $h_1 \succ h_2$, if and only if

$$\begin{array}{ll} 1. & f_i(h_1) \leq f_i(h_2)\,, \quad i = 1, 2, \ldots P\,, \\ 2. & \exists i \text{ such that } f_i(h_1) < f_i(h_2)\,. \end{array} \tag{9}$$

Using the Pareto strength given by (8), the ranking $\mathrm{F}(\cdot)$ of a timetable h_i is determined by the Pareto strength of its dominators,

$$\mathrm{F}(h_i) = \sum_{h_j \succ h_i, h_j \in \mathcal{R}^{(t)}} \mathrm{C}(h_j)\,. \tag{10}$$

Thus, the ranking of a timetable as given in (10) measures the amount of dominance applied to it by other timetables. In this context, a small ranking value indicates a good quality timetable.

4.5 Archive Update

The purpose of an archive is to memorize all current non dominated timetables. To admit a timetable $h_i \in \mathcal{R}^{(t)}$ into the archive $\mathcal{Q}^{(t)}$, no member of $\mathcal{Q}^{(t)}$ should dominate h_i, that is

$$\neg \exists h_j \in Q^{(t)}, h_j > h_i\,. \tag{11}$$

Equation (11) is the archive admission criterion. By contrast, h_i may dominate some members of $\mathcal{Q}^{(t)}$. In this case, all dominated members are removed and h_i is inserted into $\mathcal{Q}^{(t)}$. Another possible situation arises where h_i and the members of $\mathcal{Q}^{(t)}$ do not dominate each other. Then, $h_i \in \mathcal{R}^{(t)}$ replaces $h_j \in \mathcal{Q}^{(t)}$ if and only if the following conditions are met:

$$\begin{array}{ll} 1. & |h_i| = |h_j|\ , \\ 2. & \mathrm{F}(h_i) < \mathrm{F}(h_j)\,. \end{array} \tag{12}$$

Thus, a timetable replaces another timetable of same length but with a lower rank.

4.6 Mutation and Selection

The uniform mutation operator M_{α_m} is used in this work to provide diversification in the evolution process. Each exam within a timetable h_i has a mutation probability $\alpha_m = 1/|h_i|$. To mutate a timetable, we assign a random timeslot to the selected exams. The resulting effect is a slight perturbation to the scheduling composition of the timetables. However, this is a destructive process because it can introduce constraint violations into feasible timetables. The search operator L_1 will later be used to repair the unfeasible timetables created by the uniform mutation.

A selection procedure S is executed after all timetables have been mutated. The goal is to select N timetables from the combined intermediate population $\mathcal{R}^{(t)}$ to create the next population $\mathcal{P}^{(t+1)}$. Since the mutation operator can produce both feasible and unfeasible timetables, the selection procedure must be able to discriminate between them. This is accomplished by the use of the constrained dominance binary tournament [18] to select the timetables. A binary tournament involves two randomly selected timetables. The selected timetables are compared and the winner is inserted into the new population $\mathcal{P}^{(t+1)}$. In order

Table 1. Dataset characteristics

Dataset	Number of exams	Number of students
CAR-F-92	543	18419
CAR-S-91	682	16925
EAR-F-83	190	1125
HEC-S-92	81	2823
KFU-S-93	461	5349
LSE-F-91	381	2726
MEL-F-01	521	20656
MEL-S-01	562	19816
NOT-F-94	800	7896
RYE-F-92	486	11483
STA-F-83	139	611
TRE-S-92	261	4360
UTA-S-92	622	21266
UTE-S-92	184	2749
YOR-F-83	181	941

to decide which timetable is the winner, the constrained dominance relation is used [8]. Given two timetables h_1 and h_2 with constraint violations c_1 and c_2, timetable h_1 is said to constraint-dominate h_2, denoted here by $h_1 \succ_C h_2$, if one of the following conditions is met:

$$\begin{array}{ll} 1. & c_1 = 0 \,\text{and}\, c_2 > 0, \quad \text{or} \\ 2. & c_1 > 1, c_2 > 1 \text{ and } c_1 < c_2, \quad \text{or} \\ 3. & c_1 = c_2 \text{ and } h_1 \succ h_2. \end{array} \tag{13}$$

The conditions given by (13) always favor timetables with fewer conflict violations. However, when both timetables have identical conflict violations, the constrained dominance relation is reduced to the simple dominance relation.

5 Experimental Results

The hybrid MOEA described in Section 4 was tested on 15 datasets. Table 1 shows the number of exams and the number of students for each dataset. Datasets MEL-F-01 and MEL-F-02 were contributed by Merlot [15]. Dataset NOT-F-94 is by Burke [4]. All other datasets are taken from Carter [6].

Table 2 gives the algorithmic parameters and environmental settings used in the experiments. For the VND search operator L_2, N_k neighbors in the Kempe chain interchange neighborhood and N_m neighbors in the 1-move neighborhood were selected randomly to determine the current best move. The number of selected neighbors N_k and N_m are identified as the “neighborhood sample size” in Table 2. The range of timetable length (objective function f_1) depends on the datasets. Five timetable lengths centered on published values were retained

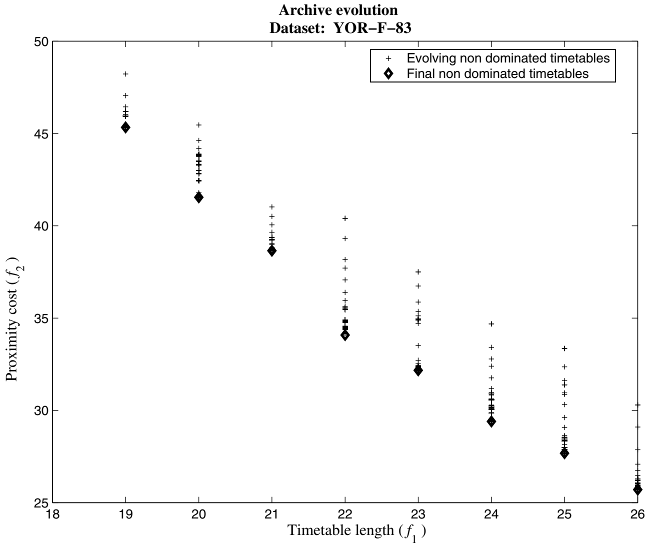


Fig. 3. Evolution of the non-dominated timetables in the archive (YOR-F-83)

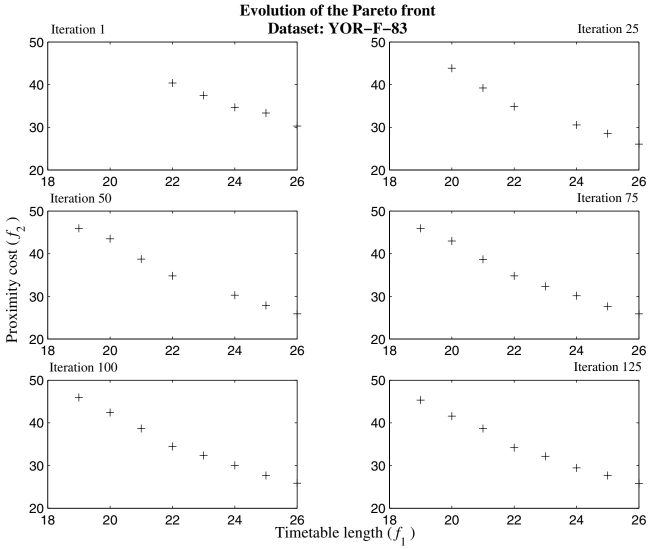


Fig. 4. Non-dominated timetables at different iterations (YOR-F-83)

Table 2. Hybrid MOEA parameters and environmental setting

Parameter	Value
Number of runs	5 per dataset
Number of iterations	$I_{\max} = 500$
Number of slots in archive	$\beta = 8$
Population and archive size	$\lvert\mathcal{Q}^{(t)}\rvert = \lvert\mathcal{P}^{(t+1)}\rvert = 80$
Neighborhood sample size	$N_k = N_m = 50$
Number of non-improvement iterations	$N_I = 100$
Mutation probability	$1/\lvert h_i\rvert$
Computer	Athlon XP 2.2 GHz, 512 MB RAM
OS	Linux 2.4.2-2
Compiler and optimization level	GNU v2.96, -O3

in the archive. The average and best proximity costs of the non-dominated timetables were computed using the weighting factors presented in Section 3. The results are detailed in Table 3. It is important to note that no fine-tuning of the hybrid MOEA has been performed and that the same parameters are used for all datasets. Although the numerical results (see Table 3) summarize the overall effectiveness of the hybrid MOEA well, it is often interesting to appreciate the dynamics of the search process. Figures 3 and 4 show the progress of the non dominated timetables in the archive for the dataset YOR-F-83. Eight different timetable lengths are used in the figures to help visualize the non-dominated front.

The effects of the hybrid MOEA can be clearly identified in Figure 4. As the optimization progresses, more and more non dominated timetables of various timetable lengths were admitted to the archive. After 125 iterations, all the empty slots in the archive were occupied with non dominated timetables. Simultaneously, the hybrid MOEA tries to lower their proximity cost. The lowering of the proximity cost can be observed by noticing the vertical displacement of the points in Figure 4 and by the trace left by the non-dominated timetables in Figure 3.

A comparison with other published results was also conducted in order to asses the effectiveness of the hybrid MOEA against other optimization methods. Since most published results for the UEPP are based on the single-objective approach with a fixed timetable length, the performance of the hybrid MOEA will also be shown for that particular timetable length.

From the results given in Table 4, the hybrid MOEA obtained the best score in three datasets. It is worth mentioning that the hybrid MOEA also achieved a second-best position in six of the 15 datasets. In summary, the proposed multi-objective evolutionary algorithm was able to produce high-quality timetables in comparison to other optimization methods.

Table 3. Non-dominated timetables and their proximity cost (5 runs per dataset)

Dataset		Results					Time (min)
CAR-F-92	$\|\mathcal{T}\|$	30	31	32	33	34	
	Best	4.9	4.5	4.2	4.2	3.9	
	Avg	4.9	4.7	4.3	4.5	4.1	583
CAR-S-91	$\|\mathcal{T}\|$	32	33	34	35	36	
	Best	6.1	5.7	5.4	5.4	5.2	
	Avg	6.2	5.9	5.5	5.5	5.3	816
EAR-F-83	$\|\mathcal{T}\|$	23	24	25	26	27	
	Best	38.0	34.2	31.6	28.8	26.7	
	Avg	39.0	35.6	31.9	29.7	27.5	102
HEC-S-92	$\|\mathcal{T}\|$	17	18	19	20	21	
	Best	12.0	10.4	9.3	8.1	7.3	
	Avg	12.1	10.5	9.3	8.2	7.5	27
KFU-S-93	$\|\mathcal{T}\|$	19	20	21	22	23	
	Best	15.8	14.3	12.1	11.0	10.0	
	Avg	16.2	14.4	12.8	11.6	10.3	165
LSE-F-91	$\|\mathcal{T}\|$	17	18	19	20	21	
	Best	12.3	11.3	9.7	8.5	7.7	
	Avg	12.6	11.5	10.1	9.3	8.1	92
MEL-F-01	$\|\mathcal{T}\|$	26	27	28	29	30	
	Best	3.6	3.2	2.8	2.9	2.4	
	Avg	3.7	3.2	2.9	2.9	2.5	269
MEL-S-01	$\|\mathcal{T}\|$	29	30	31	32	33	
	Best	2.8	2.6	2.4	2.3	2.0	
	Avg	3.0	2.7	2.5	2.3	2.1	281
NOT-F-94	$\|\mathcal{T}\|$	22	23	24	25	26	
	Best	7.8	6.9	6.2	5.7	5.0	
	Avg	8.1	7.2	6.6	5.9	5.1	289
RYE-F-92	$\|\mathcal{T}\|$	22	23	24	25	26	
	Best	9.8	8.8	7.8	7.0	7.0	
	Avg	10.1	9.1	8.1	7.2	7.3	218
STA-F-83	$\|\mathcal{T}\|$	13	14	15	16	17	
	Best	157.0	140.2	125.2	112.7	101.4	
	Avg	157.1	140.4	126.0	113.2	101.6	26
TRE-S-92	$\|\mathcal{T}\|$	21	22	23	24	25	
	Best	10.3	9.4	8.6	7.9	7.2	
	Avg	10.5	9.4	8.8	8.1	7.3	126
UTA-S-92	$\|\mathcal{T}\|$	34	35	36	37	38	
	Best	3.7	3.5	3.3	3.2	3.2	
	Avg	3.9	3.6	3.4	3.2	3.2	265
UTE-S-92	$\|\mathcal{T}\|$	10	11	12	13	14	
	Best	25.3	20.7	16.8	13.9	11.5	
	Avg	25.5	21.2	17.1	14.2	11.6	25
YOR-F-83	$\|\mathcal{T}\|$	19	20	21	22	23	
	Best	44.6	40.6	36.4	33.8	31.6	
	Avg	45.6	41.0	37.6	34.5	31.9	89

Table 4. Comparison with other methods

Dataset		hMOEA	Car	Whi	Di1	Cara	Bur	Mer	Di2	Paq	Cas	Asm
CAR-F-92	Best	4.2	6.2	–	5.2	6.0	**4.0**	4.3	–	–	4.4	4.6
32 timeslots	Avg	4.4	7.0	4.7	5.6	–	**4.1**	4.4	–	–	4.7	–
CAR-s-91	Best	5.4	7.1	–	6.2	6.6	**4.6**	5.1	–	–	5.4	5.3
35 timeslots	Avg	5.5	8.4	–	6.5	–	**4.7**	5.2	–	–	5.6	–
EAR-F-83	Best	34.2	36.4	–	45.7	**29.3**	36.1	35.1	39.4	40.5	34.8	37
24 timeslots	Avg	35.6	40.9	–	46.7	–	37.1	**35.4**	43.9	45.8	35.0	–
HEC-S-92	Best	10.4	10.6	–	12.4	**9.2**	11.3	10.6	10.9	10.8	10.8	11.8
18 timeslots	Avg	**10.5**	15.0	–	12.6	–	11.5	10.7	11.0	12.0	10.9	–
KFU-S-93	Best	14.3	14.0	–	18.0	13.8	13.7	**13.5**	–	16.5	14.1	15.8
20 timeslots	Avg	14.4	18.8	–	19.5	–	**13.9**	14.0	–	18.3	14.3	–
LSE-F-91	Best	11.3	10.5	–	15.5	**9.6**	10.6	10.5	12.6	13.2	14.7	12.1
18 timeslots	Avg	11.5	12.4	–	15.9	–	**10.8**	11.0	13.0	15.5	15.0	–
MEL-F-01	Best	**2.8**	–	–	–	–	–	2.9	–	–	–	
28 timeslots	Avg	**2.9**	–	–	–	–	–	3.0	–	–	–	
MEL-S-01	Best	**2.4**	–	–	–	–	–	2.5	–	–	–	
31 timeslots	Avg	**2.5**	–	–	–	–	–	2.5	–	–	–	
NOT-F-94	Best	**6.9**	–	–	–	–	–	7.0	–	–	–	
23 timeslots	Avg	7.2	–	–	–	–	–	**7.1**	–	–	–	
RYE-F-92	Best	8.8	7.3	–	–	**6.8**	–	8.4	–	–	–	10.4
23 timeslots	Avg	9.1	**8.7**	–	–	–	–	**8.7**	–	–	–	–
STA-F-83	Best	157.0	161.5	–	160.8	158.2	168.3	157.3	157.4	158.1	**134.9**	160.4
13 timeslots	Avg	157.1	167.1	–	166.8	–	168.7	157.4	157.7	159.3	**135.1**	–
TRE-S-92	Best	8.6	9.6	–	10.0	9.4	**8.2**	8.4	–	9.3	8.7	8.7
23 timeslots	Avg	8.8	10.8	–	10.5	–	**8.4**	8.6	–	10.2	8.8	–
UTA-S-92	Best	3.5	3.5	–	4.2	3.5	**3.2**	3.5	–	–	–	3.6
35 timeslots	Avg	3.6	4.8	4.0	4.5	–	**3.2**	3.6	–	–	–	–
UTE-S-92	Best	25.3	25.8	–	29.0	**24.4**	25.5	25.1	–	27.8	25.4	27.8
10 timeslots	Avg	25.5	30.8	–	31.3	–	25.8	**25.2**	–	29.4	25.5	–
YOR-F-83	Best	36.4	36.4	–	41.0	**36.2**	36.8	37.4	39.7	38.9	37.5	40.7
21 timeslots	Avg	37.5	45.6	–	42.1	–	**37.3**	37.9	41.7	41.7	38.1	–

Car: Carter et al. [6]; Whi: White and Xie [25];
Di1: Di Gaspero and Shaerf [10]; Cara: Caramia et al. [5];
Bur: Burke and Newall [3]; Mer: Merlot et al. [15]
Di2: Di Gaspero [9]; Paq: Paquete and Stutzle [20];
Cas: Casey and Thompson [7]; Asm: Asmuni et al. [1]

6 Conclusions

The hybrid MOEA performed well in comparison to nine other methods. Most published results for the UEPP using these publicly available datasets are based on the single-objective approach. A systematic comparison of the non dominated sets was not possible. In spite of this, the hybrid MOEA demonstrated its effectiveness by producing timetables with competitive objective values in nine of the 15 datasets without special fine-tuning. Moreover, the MOEA approach was able to generate non dominated timetables for a range of timetable lengths as alternative solutions. A contribution of this work is the use of a single framework to cover all the necessary timetabling steps:

- Initialization: Random initialization of the population and the archive;
- Search (exploitation): Variable Neighborhood Search operator and the ranking of the timetable by Pareto Strength;
- Search (exploration): Destructive uniform mutation with repair operator to obtain feasible timetables;
- Solution selection: Archive admission and non dominated timetable replacement criteria.

All these steps are fully integrated into the hybrid MOEA presented in this paper.

Acknowledgements

The authors would like to thank the referees for their constructive and helpful advice.

References

1. Asmuni, H., Burke, E. K., Garibaldi, J. M.: Fuzzy Multiple Ordering Criteria for Examination Timetabling. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 51–65
2. Burke, E., Bykov, Y., Petrovic, S.: A Multicriteria Approach to Examination Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 118–131
3. Burke, E., Newall, J.: Enhancing Timetable Solutions with Local Search Methods. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 195–206
4. Burke, E., Newall, J., Weare, R.: Memetic Algorithm for University Exam Timetabling. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 241–250
5. Caramia, M., Dell'Olmo, P., Italiano, G.: New Algorithms for Examination Timetabling. In: 4th International Workshop on Algorithm Engineering (Saarbrucken, Germany, September, 2000). Lecture Notes in Computer Science, Vol. 1982. Springer, Berlin (2001) 230–241
6. Carter, M. W., Laporte, G., Yan Lee, S.: J. Oper. Res. Soc. **47** (1996) 373–383
7. Casey, S., Thompson, J.: Grasping the Examination Scheduling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 232–244

8. Deb, K. A.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Hoboken (2001)
9. Di Gaspero, L.: Recolor, Shake and Kick: A Recipe for the Examination Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 404–407
10. Di Gaspero, L., Schaerf, A.: Tabu Search Techniques for Examination Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 104–117
11. Dueck, G.: New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel. J. Comput. Phys. **104** (1993) 86–92
12. Feo, T. A., Resende, M. G.: Greedy Randomized Adaptive Search Procedures. J. Global Optim. **6** (1995) 109–133
13. Fonseca, C., Fleming, P.: Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms. I: A Unified Formulation. IEEE Trans. on Systems, Man and Cybernetics, Part A (Systems and Humans) **28** (1998) 26–37
14. Gendreau, M., Hertz, A., Laporte, G.: Tabu Search Heuristic for the Vehicle Routing Problem. Manage. Sci. **40** (1994) 1276–1290
15. Merlot, L., Boland, N., Hughes, B., Stuckey, P.: A Hybrid Algorithm for the Examination Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 207–231
16. Mladenovic, N., Hansen, P.: Variable Neighborhood Search. Comput. Oper. Res. **24** (1997) 1097–1100
17. Morgenstern, C., Shapiro, H.: Coloration Neighborhood Structures for General Graph Coloring. In: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms (San Francisco). Society for Industrial and Applied Mathematics (1990) 226–235
18. Osyczka, A., Krenich, S.: New Constraint Tournament Selection Method for Multicriteria Optimization Using Genetic Algorithm. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC2000), Vol. 1. IEEE, Piscataway, NJ (2000) 501–508
19. Paquete, L., Fonseca, C.: A Study of Examination Timetabling with Multiobjective Evolutionary Algorithms. In: 4th Metaheuristics International Conference (MIC 2001, Porto). 149–154
20. Paquete, L., Stutzle, T.: Empirical Analysis of Tabu Search for the Lexicographic Optimization of the Examination Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 413–420
21. Petrovic, S., Bykov, Y.: A Multiobjective Optimization Technique for Exam Timetabling Based on Trajectories. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 179–192
22. Radcliffe, N. J., Surry, P. D.: Formal Memetic Algorithms. In: Proceedings of the AISB Workshop on Evolutionary Computing (Leeds, UK, April, 1994). Lecture Notes in Computer Science, Vol. 865. Springer, Berlin (1994) 1–16

23. Silva Linda, J. D., Burke, E. K., Petrovic, S.: An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling. In: Gandibleux, X., Sevaux, M., Sorensen, K., T'Kindt, V. (eds.) MetaHeuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems, Vol. 535. Springer, Berlin (2004) 91–129
24. Thompson, J. M., Dowsland, K. A.: Robust Simulated Annealing Based Examination Timetabling System. Comput. Oper. Res. **25** (1998) 637–648
25. White, G., Xie, B.: Examination Timetables and Tabu Search with Longer-Term Memory. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 85–103
26. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In: Evolutionary Methods for Design Optimisation and Control (Barcelona, Spain, CIMNE 2002) 95–100

Examination Timetabling with Fuzzy Constraints

Sanja Petrovic, Vijay Patel, and Yong Yang

School of Computer Science and Information Technology,
The University of Nottingham, NG8 1BB, UK
{sxp, vxp01u, yxy}@cs.nott.ac.uk

Abstract. The aim of this paper is to consider flexible constraint satisfaction in timetabling problems. The research is carried out in the context of university examination timetabling. Examination timetabling is subject to two types of constraints: hard constraints that must not be violated, and soft constraints that often have to be violated to some extent. Usually, an objective function is introduced to measure the satisfaction of soft constraints in the solution by summing up the number of students involved in the violation of the constraint.

In existing timetabling models the binary logic strategy is employed to handle the satisfaction of the constraints, i.e. a constraint is either satisfied or not. However, there are some constraints that are difficult to evaluate using the binary logic: for example, the constraint that large exams should be scheduled early in the timetable. Fuzzy IF–THEN rules are defined to derive the satisfaction degree of this constraint, where both the size of the exam and the time period that the exams are scheduled in are expressed using the linguistic descriptors *Small*, *Medium* and *Large*, and *Early*, *Middle* and *Late*, respectively. In a similar way, the constraint that students should have enough break between two exams is modelled. A number of memetic algorithms with different characteristics are developed where corresponding fitness functions aggregate the satisfaction degrees of both fuzzy constraints. The proposed approach is tested on real-world benchmark problems and the results obtained are discussed.

1 Introduction

Examination and course timetabling are considered to be very important administrative activity at universities. This research is focused particularly on university examination timetabling problems [10], [29]. The examination timetabling problem is defined to be a problem of assigning a number of exams into a limited number of time periods, subject to constraints [37]. University usually offers a collection of modules and each student may choose a certain number of modules. This, of course, increases the complexity of timetabling problems. Examination timetabling is subject to two types of constraints: hard and soft. Hard constraints must not be violated, while soft constraints are desirable to satisfy, but often it is not possible to find a solution that satisfies all of them, i.e. they have to be violated to some extent. Hard and soft constraints differ significantly between

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 313–333, 2005.

universities. A typical hard constraint, that is also considered in this research, is that no student can be scheduled to sit two exams at the same time. Often, the constraint that total resources at any time should not exceed the resource available is considered to be a hard constraint. For example, for each time period there should be enough room space available for the exams that are scheduled in that time period. There exist a variety of constraints that are treated as soft, which could be grouped in the categories including:

- *Time assignment*: for example, some exams have to be scheduled in particular time periods, or large exams should be scheduled earlier in the timetable to give enough time for their assessment.
- *Spreading events out in time*: for example, the university aims to increase the free time periods between each student examinations.
- *Time constraints between events*: for example, an exam needs to be scheduled before/after the other.
- *Resource assignment*: for example, an exam must be scheduled in a particular room.

Comprehensive overviews of different constraints that are imposed by universities are given in [17].

There is a wide variety of different approaches to examination timetabling including graph colouring heuristics [5], variety of meta-heuristics including the great deluge algorithm [4], simulated annealing [35], tabu search [19], [36], and memetic algorithms [6], [14], [16], heuristic problem decomposition methods [13], adaptive heuristics [7], hyper-heuristic methods [9], and case-based reasoning [3], [28].

In the majority of these approaches, an objective function is introduced to measure the quality of the obtained timetables by measuring the satisfaction of soft constraints. However, the majority approaches consider only a single soft constraint on proximity of exams for students, i.e. they take into consideration only the requirement that students should have enough break between two examinations.

The main aim of the research presented in this paper is to consider other constraints that can be imposed on timetabling. These constraints are of different nature, i.e., incompatible with each other and even more in conflict with each other, i.e. an attempt to improve the satisfaction of one constraint may lead to worsening of the other constraint. Recent years have seen an acceleration of interest in multi-criteria approaches to examination timetabling, although the field has not been fully explored yet [21]. In these approaches examination timetabling problems are stated as multicriteria problems, while criteria measure the violations of constraints. Paquete and Fonseca [26] developed a multiobjective evolutionary algorithm based on Pareto-ranking. The idea of compromise programming was used in order to find the solution which is closest to the ideal solution which does not violate any of the constraints [12]. Petrovic and Bykov

[30] developed a multicriteria approach in which weights are dynamically changed in order to direct the search of the criteria space along the specified trajectory. These approaches can better handle multiple criteria (constraints) than can a single objective function.

Our research is focused on constraints that are difficult to describe using the binary logic strategy (i.e. it is difficult to state whether a constraint is fully satisfied or not). A typical example is the constraint on large exams that should be put in the early part of the timetable, which, to the best of our knowledge, has not been handled in approaches described in the timetabling literature. Such a constraint requires a description of imprecise terms *large exams* and *early part of the timetable*, together with the definition of the measure of the satisfaction of this constraint in the constructed timetable. Fuzzy sets and fuzzy logic have been successfully applied to address imprecise and vague terms in various application domains [33]. A fuzzy set is a very general concept that extends the notion of a standard set defined by a binary membership of objects in the set, by introducing various degrees of memberships [20]. Fuzzy scheduling models have recently attracted increased interest among the scheduling research community [34]. However, the application of fuzzy theory to timetabling problems in particular has been very limited. Fuzzy constraints have been employed in nurse rostering [23] and in university timetabling problems, although in a different context, namely in ordering exams for scheduling using a number of graph colouring heuristics [1] and in determining the weights of conflicts among the exams in [38].

Apart from the constraint on large exams, we also consider the constraint that students should have a break between two consecutive exams. These two constraints are incommensurable in the sense that their violations in the timetable are measured by different measurement units with different scales. Usually, the violation of the second constraint is assessed by the number of students who have exams “close” to each other. Following this approach the violation of the first constraint can be measured by the number of large exams that are not scheduled in appropriate “early” time slots eventually weighted by the number of students sitting these exams. However, an objective function that adds these two measurements with different scales would not be appropriate. Therefore, we employ fuzzy sets to introduce gradualism in the constraint satisfaction. Namely, the degree of a constraint satisfaction is described by five fuzzy sets: *Low*, *Medium-low*, *Medium*, *Medium-high*, and *High*. Fuzzy IF-THEN rules are defined to derive the satisfaction degree of each of the constraints. A number of memetic algorithms with different characteristics are developed whose fitness function aggregates the satisfaction degrees of both fuzzy constraints.

The paper is organised as follows. Section 2 introduces fuzzy sets and fuzzy IF–THEN rules. Fuzzy constraints considered in this research are described in Section 3. Section 4 presents fuzzy memetic algorithms which employ fuzzy constraints. The experimental results obtained on benchmark problems are given in Section 5, followed by conclusions.

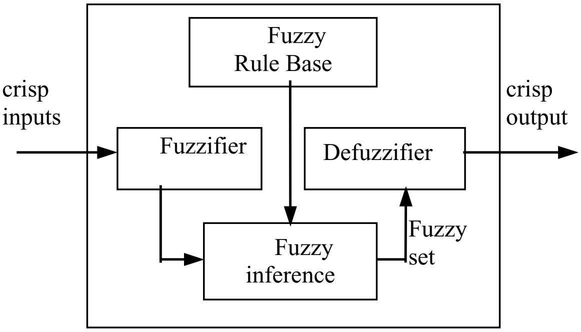


Fig. 1. A fuzzy rule-based system

2 Fuzzy Sets and Approximate Reasoning

Fuzzy sets provide an appropriate modelling tool for treating imprecise information [39]. They are generalisations of *classical* or so-called *crisp sets.* A crisp set dichotomises the objects of a given universe of discourse in the sense that an object either belongs or does not belong to the set. However, in many real-life situations, such as categorisation of exams with respect to their size, a rigorous definition of the set boundary is not appropriate. A *fuzzy set* A can be defined by a membership function $\mu_A(x)$ which assigns to each object x in the universe of discourse X a value representing its grade of membership in this fuzzy set:

$$\mu_A : X \to [0, 1] .$$

Therefore an object may belong to the fuzzy set to a greater or smaller degree which is conventionally represented by a real-number from the interval [0, 1].

Fuzzy IF–THEN rules can be used to capture human knowledge/experience of a particular domain using imprecise assertions [25]. A fuzzy rule-based system comprises four components which are depicted in Figure 1: a fuzzifier, a fuzzy rule-base, a fuzzy inference engine and a defuzzifier:

1. A *fuzzifier* takes the crisp input values and determines the degrees to which they belong to fuzzy sets in the premises of the rules.
2. A *fuzzy rule base* (FRB) consists of fuzzy rules

   $$\text{FRB} = \{R_1, \ldots, R_m, \ldots, R_M\}$$

   where R_m, $m = 1, \ldots, M$ denote fuzzy rules. Rules connect premises to conclusion. For instance, a rule R_m with two premises and one conclusion is represented by

   $$R_m : \text{IF}(x \text{ is } A_m) \text{ AND } (y \text{ is } B_m) \text{ THEN } z \text{ is } C_m$$

   where A_m, B_m and C_m, $m = 1, \ldots, M$ are fuzzy sets.

3. A *fuzzy inference engine* (FIE) implements approximate reasoning, which is the process of deriving imprecise conclusions using imprecise premises [40]. The first step is to evaluate premises of each rule. In our case a conjunction of two premises has to be evaluated. Different triangular norms can be used [27]. The definition of triangular norms is given in the Appendix. In this paper, we use the algebraic product as the triangular norm. The product of the values of the corresponding membership degrees of the premises is calculated and taken as the truth value of the premises of each rule:

   $$\mu_{A_m \times B_m}(x, y) = \mu_{A_m}(x) * \mu_{Bm}(y) \quad m = 1, \ldots, M\,.$$

   For crisp inputs x_0 and y_0 the truth value of the premises of the rule is

   $$\mu_{A_m}(x_0) * \mu_{Bm}(y_0)\,.$$

   The next step is to infer a possible conclusion taking into consideration the calculated truth value of the premises of the rule. Different triangular conorms are applicable here [27], and their definition is given in the Appendix. The most common method is to use the truth value of the rule premises and "to cut" the membership function of the conclusion of the rule at that level. As a result, the fuzzy set denoted by $A_m(x_0) \times B_m(y_0) \rightarrow C_m$ with the following membership function is obtained:

   $$\mu_{A_m(x_0)\times B_m(y_0)\rightarrow C_m}(z) = \min\left\{\mu_{A_m \times B_m}(x_0, y_0), \mu_{C_m}(z)\right\}.$$

   While in the operation of a classical rule based system only one rule whose premises are true is fired, in the fuzzy rule based system all the rules fire. The final fuzzy set is determined by aggregation of all the obtained fuzzy sets in the rule conclusions into a single fuzzy set R:

   $$\mu_R(z) = \max\left\{\mu_{A_1(x_0)\times B_1(y_0)\rightarrow C_1}(z), \ldots, \mu_{A_M(x_0)\times B_M(y_0)\rightarrow C_M}(z)\right\}.$$

4. A *defuzzifier* maps a final fuzzy set obtained as a result of firing all the rules in the FRB to a crisp value which "most appropriately" represents the fuzzy set. Different mappings can be defined and used in the defuzzification. We use a well-known defuzzifier which calculates the centre of gravity:

   $$\overline{r} = \frac{\int\limits_{-\infty}^{\infty} x\mu_R(x)\,\mathrm{d}x}{\int\limits_{-\infty}^{\infty} \mu_R(x)\,\mathrm{d}x}\,.$$

3 Fuzzy Constraints

In our research fuzzy sets are used to model imprecise constraints imposed on examination timetabling. For each constraint we define a number of fuzzy IF–THEN rules in which both premises and the conclusion are described by fuzzy sets. The conclusion of each rule describes the satisfaction of the constraint on the basis of the premises. Before describing fuzzy rules defined for each constraint the notation that will be used in the description of fuzzy rules is given:

- $N = 1, \ldots, N$ indicate exams,
- N is the total number of examinations,
- M is the number of students,
- P is the given number of time periods,
- S is the number of time periods per day,
- $C = (c_{ij})_{N \times N}$ is a symmetrical conflict matrix where elements c_{ij}, $i, j = 1, \ldots, N$, $i \neq j$ represent the number of students taking both examinations i and j, while the diagonal elements c_{nn}, $n = 1, \ldots, N$ of this matrix represent the number of students who take exam n,
- $T = (t_n)_N$ represents a timetable as a vector, where t_n indicates the assigned time period for exam $n = 1, \ldots, N$.

3.1 Constraint on Large Exams

The constraint that large exams should be placed early in the timetable is difficult to model using crisp sets with strict boundaries. It is too rigorous to use a crisp number of students who take the exam as a discriminator for the size of an exam. Also, it is difficult to define a set of early periods in the timetable using the ordinal number of time periods. Fuzzy sets are used to represent both linguistic variables *Size of Exam* and *Time Period* that the exam is scheduled into. *Size of Exam* is described by terms *Small*, *Medium* and *Large*, while *Time Period* is assessed as *Early*, *Middle* and *Late* with respect to its position in the timetable.

The linguistic terms that are assigned to *Size of Exam* are given in Figure 2. The membership functions defined are subjective. They should capture the timetabling officer's system of values and are dependent on the examination timetabling context. We define the support of each of these linguistic terms with respect to the size of the largest exam, which is denoted by X. It can be determined from the conflict matrix C:

$$X = \max_{n=1,\ldots,N} c_{nn} .$$

For example, the exam whose size is $(X + 1)/8$ belongs to a set of *Small* size exams with a high degree (0.8). It also belongs to a set of *Medium* size exams but with a lower degree (0.2).

Similarly, linguistic terms for Time Period, *Early*, *Middle*, *Late*, are defined with respect to the total number of time periods P. They are given in Figure 3.

The satisfaction degree of the constraint on large exams is derived using these two linguistic variables *Size of Exam* and *Time Period.* The satisfaction degree is assessed as *Small*, *Medium-small*, *Medium*, *Medium-high* and *High* whose membership functions are presented in Figure 4.

Nine fuzzy rules are defined to derive for each exam the satisfaction degree (*SD*) of the constraint on large exams. They cover all combinations of values of linguistic variables in the premises of the rules. They reflect the timetabling officer's preferences to the allocation of exams with respect to their size. For example, large exams should be placed in early time periods. If this is the case

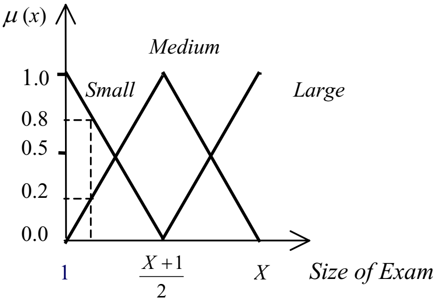


Fig. 2. Linguistic variable *Size of Exam*

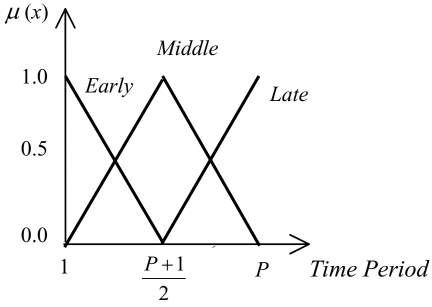


Fig. 3. Linguistic variable *Period*

the satisfaction degree of the constraint of such an exam is *High*. Small exams can be placed anywhere with light preferences toward the middle and the late part of the timetable. The fuzzy rules for the constraint on large exams are given in Figure 5.

As an illustration, let us assume that the size of exam is 100 (X=799), while the time period the exam is scheduled in is 6 (P=23). The truth value of the first rule whose premises are *Size of Exam* is *Small* AND *Time Period* is *Early* is equal to $0.8 \times 0.5 = 0.4$. That value is used to "cut" the fuzzy set *Medium-high* in the conclusion of the rule. All the rules are fired leading to the resultant fuzzy set that was represented as greyed in Figure 6. Its centre of gravity (0.765) presents the satisfaction degree of the constraint on large exams for that particular exam.

The overall satisfaction degree of the constraint on large exams, denoted by f_1, of the timetable T is calculated as the minimum of satisfaction degrees obtained for all exams e_n, $n = 1, \ldots, N$:

$$f_1(T) = \min\{\mu_{f_n}(e_n) | n = 1, \ldots, N\} .$$

We consider such defined non-compensatory overall satisfaction degree to be appropriate for this constraint because it evaluates the timetable taking into consideration the worst placed exam.

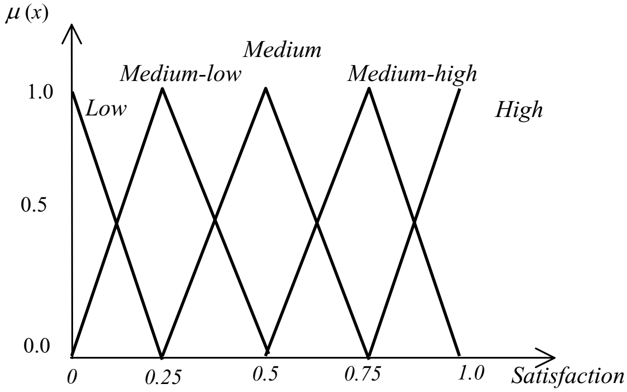


Fig. 4. Linguistic variable *Satisfaction* of the constraint on large exams

IF *Size of Exam* is *Small* AND *Time Period* is *Early* THEN *Satisfaction* is *Medium-high*
IF *Size of Exam* is *Small* AND *Time Period* is *Middle* THEN *Satisfaction* is *High*
IF *Size of Exam* is *Small* AND *Time Period* is *Large* THEN *Satisfaction* is *High*
IF *Size of Exam* is *Medium* AND *Time Period* is *Early* THEN *Satisfaction* is *Medium-high*
IF *Size of Exam* is *Medium* AND *Time Period* is *Middle* THEN *Satisfaction* is *Medium*
IF *Size of Exam* is *Medium* AND *Time Period* is *Large* THEN *Satisfaction* is *Medium-low*
IF *Size of Exam* is *Large* AND *Time Period* is *Early* THEN *Satisfaction* is *High*
IF *Size of Exam* is *Large* AND *Time Period* is *Middle* THEN *Satisfaction* is *Medium*
IF *Size of Exam* is *Large* AND *Time Period* is *Large* THEN *Satisfaction* is *Low*

Fig. 5. Fuzzy rules for the constraint on large exams

3.2 Constraint on Proximity of Exams

Similarly to the constraint on large exams, fuzzy IF–THEN rules are defined to derive the satisfaction degree of the constraint on proximity of exams for each pair of exams that are in conflict. Two linguistic variables are defined: *Common Students* to denote the number of students common for two exams in conflict and *Period Difference* between these exams. The linguistic terms *Small*, *Medium*, and *Large* are used to evaluate both linguistic variables. Of course, these linguistic terms have different membership functions for each linguistic variable. Membership functions of the linguistic terms defined for *Common Students* and for *Period Difference* are given in Figures 7 and 8, respectively.

Linguistic terms for common students are defined taking into consideration the maximum number of students common for two exams:

$$C_{\max} = \max\{c_{kl} | k, l = 1, \ldots, N, k \neq l\}.$$

The membership functions of linguistic terms for *Period Difference* is based on the total number of time periods P and the number of time periods per day S. For example, a two days difference between two exams with common students is considered to be *Large* with membership degree 1 ($\mu_{\text{Large}}(2S) = 1$).

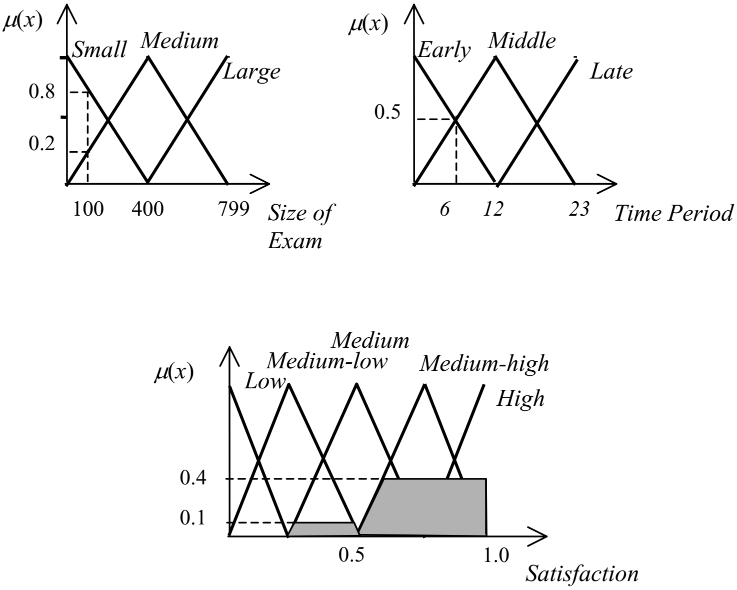


Fig. 6. Example of firing fuzzy rules for the constraint on large exams when *Size of Exam* is 100 and *Time Period* is 6

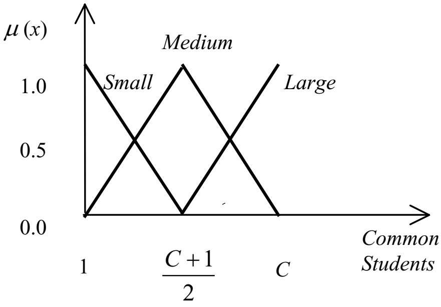


Fig. 7. Linguistic variable *Common Students*

Nine fuzzy IF–THEN rules are defined to deduce the satisfaction degree of the constraint on proximity of exams for each pair of exams that are in conflict and are given in Figure 9. For example, if there is a *Large* number of common students for a pair of exams and there is a *Small* period difference between these two exams then the satisfaction degree is *Low*. On the other hand, if there is a *Large* period difference for a pair of exams the satisfaction is *Low* independently from the number of students common for these two exams.

Overall satisfaction of the constraint on proximity of exams, denoted by f_2, of timetable T is calculated in the following way:

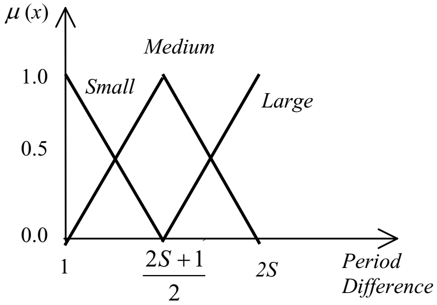


Fig. 8. Linguistic variable *Period Difference*

$$f_2(T) = \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \mu_{f_2}(e_i, e_j)}{C}$$

where e_i and e_j , $i, j = 1, \ldots, N$, are exams, while C is the total number of pairs of exams in conflict. Such defined overall satisfaction is compensatory one, which means that a higher satisfaction degree of one pair of exams can compensate, to a certain extent, for a lower satisfaction degree of another pair of exams.

4 Fuzzy Memetic Algorithms for Examination Timetabling

Genetic algorithms mimic the natural evolution process throughout the search for a solution. A genetic algorithm handles a population of solutions through generations following the principles of natural selection, namely the fittest solutions have greater chances to survive to the next generation. The fitness of a solution is determined by the value of the objective function achieved by the solution. The members of the new population are created by application of evolutionary operators, which change the parts of the current solutions.

A memetic algorithm attempts to improve the performance of a genetic algorithm by performing local neighbourhood search [24]. The local search explores the neighbourhood of the solutions produced by the genetic algorithm trying to reach local optima for each solution. Improved solutions are then passed to the next generation of the genetic algorithm.

In the memetic algorithms developed for timetabling the objective function aggregates the satisfaction degree of both constraints:

$$F(T) = w_1 * f_1(T) + w_2 * f_2(T) \tag{1}$$

where w_1 and w_2 denote relative importance of each constraint $(w_1 + w_2 = 1)$.

IF *Common Students* is *Small* AND *Period Difference* is *Small* THEN *Satisfaction* is *Medium*
IF *Common Students* is *Small* AND *Period Difference* is *Medium* THEN *Satisfaction* is *Medium-high*
IF *Common Students* is *Small* AND *Period Difference* is *Large* THEN *Satisfaction* is *High*
IF *Common Students* is *Medium* AND *Period Difference* is *Small* THEN *Satisfaction* is *Medium-low*
IF *Common Students* is *Medium* AND *Period Difference* is *Medium* THEN *Satisfaction* is *Medium*
IF *Common Students* is *Medium* AND *Period Difference* is *Large* THEN *Satisfaction* is *High*
IF *Common Students* is *Large* AND *Period Difference* is *Small* THEN *Satisfaction* is *Low*
IF *Common Students* is *Large* AND *Period Difference* is *Medium* THEN *Satisfaction* is *Medium-low*
IF *Common Students* is *Large* AND *Period Difference* is *Large* THEN *Satisfaction* is *High*

Fig. 9. Fuzzy rules for the constraint on proximity of exams

The initial solutions of the memetic algorithms are constructed by Brelaz's saturation degree graph colouring sequencing heuristic [2]. This heuristic schedules exams sequentially, one by one, commencing with the exams with the least number of available periods for placement. The exams from the largest clique of the graph (i.e., the largest set of mutually conflicting exams) are scheduled first. In order to maintain the diversity of the solutions, an exam to be scheduled next is selected from a subset of randomly chosen exams from the clique (the size of the subset was set as 30% of the total number of exams). Laporte and Desroches' backtracking method was used to schedule the exams that cannot be assigned to any time period [22].

Six different memetic algorithms have been developed and tested on university examination timetabling benchmarks problems. All these algorithms employ the same fitness function given in (1) and construct the initial solutions in the described way. The components that differ these algorithms are given in Table 1 and are described in detail below followed by the list of components comprised by each algorithm.

Evolutionary Operators. These manipulate members of the current population to form solutions of the next generation.

The use of crossover operators which combine two different solutions was shown not to be beneficial in genetic algorithms for university timetabling [32]. Therefore, only mutation operators, which manipulate single solution aiming to escape local optima in the search space are defined. They are labelled as *light* if they have a random element or *heavy* if they do not have it. They are defined as follows:

1. *Light Mutation 1* chooses randomly an exam from the timetable and reschedules it into another randomly chose valid time period;
2. *Light Mutation 2* chooses randomly an exam from the timetable and reschedules it into another valid time period that causes the least decrease to the objective function value;
3. *Light Mutation 3* swaps exams from randomly chosen time periods;
4. *Heavy Mutation* reschedules the exam with the least satisfaction degree and places it into the best valid time period with respect to the objective function.

Hill-Climbing Local Search. This is applied to the offspring solutions obtained by the evolutionary operators. It reschedules each exam into the best valid time period with respect to the objective function. The exams are ordered for rescheduling using a number of heuristics:

1. Largest Degree (LD): exams with the largest number of conflicts are rescheduled first.
2. Largest Weighted Degree (LWD): exams with the largest sum of the weighted conflicts are rescheduled first, where each conflict is weighted by the number of students who are enrolled in both exams.

3. Number of Enrolments (ENR): exams with the largest student enrollment are rescheduled first.
4. Order of the Timetable (TT). exams are rescheduled following their schedule in the solution.
5. Random order (RO).

Operator Selection Method. Three different operator selection methods are developed:

– Non-adaptive: the list of evolutionary and Hill Climbing operators is predefined and does not change during the search. Two different predefined lists are used:
  - MA1: *Light Mutation 1* and *Heavy Mutation* while Hill climbing operator randomly chooses LD, LWD, NOR, and TT to order exams.
  - MA2 *Light Mutation 2* and *Heavy Mutation* while Hill climbing operator randomly chooses LD, LWD, NOR, and TT to order exams.

– Adaptive MA: the list of evolutionary and Hill Climbing operators changes depending on the improvement of the average value of the objective function of the last two generations. If the improvement is smaller than a threshold (defined as 0.0002), then *Light Mutation 1* and *Light Mutation 3* are applied successively, and the exams ordering method for hill climbing is randomly selected from LD, LWD and NOR. Otherwise, the applied mutation operator is randomly selected from *Light Mutation 2* and *Heavy Mutation*, and hill climbing orders exams using RO.

The rationale behind combining components in an algorithm is as follows. If the improvements between generation is little we allow more randomness in the search. Mutation operators are combined in order to change the current solution more than a single mutation can do, thus enabling the search to avoid the local optimality. To some extent, this combination functions similarly to the increase of the mutation rate in a standard evolutionary algorithm. Hill climbing in non-adaptive algorithms is given more choice for ordering exams than in adaptive algorithms to enable them to generate better solutions.

Population Selection Methods. Two different selection methods have been utilised to select individuals for the reproduction for the next generation:

1. Static Selection: an evolutionary operator is applied to every solution in a population.
2. Dynamic Selection: solutions are selected from a population using the classic roulette wheel where the probability of the solution to be selected increases with its fitness.

Table 1 summarises the components of all six memetic algorithms.

Table 1. Fuzzy memetic algorithms developed for examination timetabling

Algorithm	Population Selection	Change in operators?	Mutation		Hill Climbing
Adaptive Static Memetic Algorithm (ASMA)	Static (Each Individual)	Yes (Adaptive)	*Improvement* Little	*Light Mutation 1* and then *Light Mutation 3*	Randomly ordered exams
			Improvement Significant	Randomly selected from *Light Mutation 2* and *Heavy Mutation*	Randomly selected from *LD*, *LWD*, *ENR*
Static Memetic Algorithm 1 (SMA1)	Static (Each Individual)	No (Non-Adaptive)	Randomly selected from *Light Mutation 1* and *Heavy Mutation*		Randomly selected from *LD*, *LWD*, *ENR*, *TT*
Static Memetic Algorithm 2 (SMA2)	Static (Each Individual)	No (Non-Adaptive)	Randomly selected from *Light Mutation 2* and *Heavy Mutation*		Randomly selected from *LD*, *LWD*, *ENR*, *TT*
Adaptive Dynamic Memetic Algorithm (ADMA)	Dynamic (Roulette Wheel)	Yes (Adaptive)	*Improvement* Little	*Light Mutation 1* and then *Light Mutation 3*	Randomly ordered exams
			Improvement Significant	Randomly selected from *Light Mutation 2* and *Heavy Mutation*	Randomly selected from *LD*, *LWD*, *ENR*
Dynamic Memetic Algorithm 1 (DMA1)	Dynamic (Roulette Wheel)	No (Non-Adaptive)	Randomly selected from *Light Mutation 1* and *Heavy Mutation*		Randomly selected from *LD*, *LWD*, *ENR*, *TT*
Dynamic Memetic Algorithm 2 (DMA2)	Dynamic (Roulette Wheel)	No (Non-Adaptive)	Randomly selected from *Light Mutation 2* and *Heavy Mutation*		Randomly selected from *LD*, *LWD*, *ENR*, *TT*

Table 2. The benchmark examination timetabling problems

Data	Institution	Periods (P)	No. of exams (N)	No. of students (M)	No. of enrol-ments	Density of conflict matrix
Ear-f-83	Earl Haig Collegiate Institute, Toronto	24	190	1,125	8,109	0.29
Hec-s-92	Ecole des Hautes Etudes Commercials, Montreal	18	81	2,823	10,632	0.20
Lse-f-91	London School of Economics	18	381	2,726	10,918	0.06
Sta-f-83	St Andrew's Junior High School, Toronto	13	139	611	5,751	0.14
Tre-s-92	Trent University, Peterborough, Ontario	23	261	4,360	14,901	0.18
Ute-s-92	Faculty of Engineering, University of Toronto	10	184	2,750	11,793	0.08
Yor-f-83	York Mills Collegiate Institute, Toronto	21	181	941	6,034	0.27

5 Experiments on Benchmark Problems

A number of real-world examination problems from different universities have been collected and used as benchmark problems within the timetabling community. They can be obtained from the Web page with the following URL: ftp://ftp.mie.utoronto.ca/pub/carter/testprob/. Characteristics of these problems are shown in Table 2.

All the experiments were run on a PC with a 1800 Mhz P-4 processor and 256 MB RAM. Six fuzzy memetic algorithms were used to solve the above benchmark problems. The following parameters were set for all memetic algorithms: the population size was 10, the total number of generations was 30, $w_1 = 0.5$, $w_2 = 0.5$. Each experiment was run five times and average results are obtained. The results are summarised in Table 3. The table also shows the average time (in seconds) spent on the search.

From Table 3 it can be seen that among the six developed methods for majority of benchmark problems (6 out of 7) the best results are obtained by applying the Adaptive Static Memetic Algorithm (ASMA).

In the current literature majority of the state-of-the-art approaches to examination timetabling methods consider only constraint of the proximity of exams and use Carter's objective function [18]. Due to the different objective function

Table 3. Results for benchmark problems obtained by different fuzzy Memetic Algorithms

	ADMA		ASMA		DMA1		DMA2		SMA1		SMA2	
Data	*F*(*T*)	Time (s)	*F*(*T*)	Time (s)	*F*(*T*)	Time (s)	*F*(*T*)	Time (s)	*F*(*T*)	Time (s)	*F*(*T*)	Time (s)
Ear-f-83	0.80091	348	**0.801327**	392	0.796377	163	0.796496	138	0.797271	198	0.796765	155
Hec-s-92	**0.773989**	132	0.77267	162	0.770908	25	0.770466	25	0.770394	60	0.770972	44
Lse-f-91	0.84014	603	**0.840573**	603	0.839105	465	0.838687	429	0.839364	474	0.839161	431
Sta-f-83	0.755078	159	**0.757132**	146	0.738802	68	0.738806	31	0.739524	84	0.738907	49
Tre-s-92	0.809965	488	**0.810312**	490	0.809351	300	0.809123	252	0.809763	318	0.809612	265
Ute-s-92	0.774878	488	**0.776616**	490	0.760635	300	0.758995	252	0.760694	318	0.760883	265
Yor-f-83	0.776604	372	**0.77723**	332	0.77412	113	0.774217	117	0.774723	162	0.774361	106

Table 4. Evaluation of the timetables produced by ASMA method ($w_1 = 0.5$, $w_2 = 0.5$)

Problem	Av. period / *P*	Adj.	Same day	Next day	Rem.
Ear-f-83	9 / 24	8	8	8	76
Hec-s-92	7 / 18	14	10	15	61
Lse-f-91	7 / 18	7	11	11	71
Sta-f-83	8 / 13	16	14	14	55
Tre-s-92	10 / 23	8	8	8	75
Ute-s-92	4 / 10	19	16	17	47
Yor-f-83	8 / 21	8	10	9	72

Table 5. Evaluation of the timetables produced by ASMA method ($w_1 = 0.95$, $w_2 = 0.05$)

Problem	Av. period / P	Adj.	Same day	Next day	Rem.
Sta-f-83	6 / 13	17	15	14	54

used in this approach it is not possible to compare it with other state-of-the-art approaches. Nevertheless, some form of evaluation of the obtained timetables needs to be performed. The best solution for each benchmark problem found by the ASMA algorithm was thoroughly investigated. In order to assess the placement of large exams, we employ the following method. An exam is regarded as a large one if its enrolment is greater than $0.75 \times X$, where X is the maximum size among all exams. Here we use a crisp set boundary to define large exams (which can be considered as a deficiency of this evaluation, but can serve the purpose). The two numbers in the first column in the table shows the average period numbers that the large exams are scheduled into and the total number of time periods. It can be noticed that in all but one solution large exams are placed in the first half of the timetable, even more around the first third of the timetable. The remaining columns in the table are related to the constraint on proximity of exams.

Let us consider a weighted a graph which represents an examination timetabling problem. Vertices represent exams, while there is an edge between two vertices if the corresponding exams are in conflict. The weight associated with an edge represents the number of students who sit both exams. The column *Adj* in Table 4 shows the percentage of weighted edges in such a graph which connect exams scheduled into adjacent periods. Similarly, columns *Same Day* and *Next Day* show the percentage of weighted edges in the graph which connect exams scheduled on the same day or on the next day (period difference between exams is greater than 3). The column *Rem* shows that the largest percentage of weighted edges connects exams that are scheduled with at least two days between them.

The alterations of weights will lead to improvement of the satisfaction of one constraint at the expense of the other one. For example let us assume that the timetable officer wants to "improve" the placement of large exams in the timetable constructed for Sta-f-83 benchmark problem. New pair of weights, $w_1 =$

0.95, $w_2 = 0.05$, will lead to better timetables with respect to large exams at the reasonable expense of the constraint of proximity of the exams. The evaluation of the timetables obtained with the new weights is illustrated in Table 5.

6 Conclusions

This paper presents a novel fuzzy-sets-based approach to modelling constraints imposed on university examination timetabling that are difficult, or even impossible, to describe using the binary logic (i.e. a constraint is either satisfied or not). A typical such constraint is considered, namely the constraint that large exams should be put in the early part of the timetable. Fuzzy IF-THEN rules are defined to derive the satisfaction degree of the constraint, which is expressed using the linguistic descriptors. In the similar way the constraint that students should have enough break between two exams is modelled. The membership functions of the fuzzy sets used in the fuzzy IF-THEN rules are subjective in their nature and enable the timetable officer to express his/her preferences. An additional advantage of using fuzzy sets to derive the satisfaction degrees is that they have the same range of values in the interval $[0, 1]$. It enables a multiple constraints with different measurement units of different scales to be treated together, i.e. they are combined in an overall satisfaction degree of all the constraints. A number of memetic algorithms were developed which used the aggregated satisfaction degree of all the constraints as the objective function. In the described approach a sum of the satisfaction degrees of the two constraints is used to aggregate the individual satisfaction degrees of each constraint.

Future research work will be carried out in the following directions:

1. Other types of aggregation of satisfaction degrees of the constraints and their effect of the timetable will be investigated.
2. Other constraints that can be imposed on examination timetabling will be considered to be included in the described framework.
3. Modelling of uncertain constraints using fuzzy sets is subjective in nature and it relies on timetable officer's subjective opinion. Therefore, the sensitivity of a constructed timetable to small changes to membership functions used in the fuzzy IF-THEN rules will be examined.

References

1. Asmuni, H., Burke, E. K., Garibaldi, J.: Fuzzy Multiple Ordering Criteria for Examination Timetabling. In: Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 51–65
2. Brelaz D.: New Methods to Color the Vertices of a Graph. Commun. ACM **22** (1979) 251–256
3. Burke, E. K., Petrovic, S., Qu, R.: Case Based Heuristic Selection for Timetabling Problems. J. Scheduling (2006) accepted for publication

4. Burke, E. K., Bykov, Y., Newall, J. P., Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Trans. on Oper. Eng. **36** (2004) 509–528
5. Burke, E. K., Kingston J., De Werra, D.: Applications to Timetabling. Section 5.6 in: Gross, J. Yellen, J. (eds.): Handbook of Graph Theory. Chapman and Hall/CRC Press, London (2004) 445–474
6. Burke E. K., Landa, J. D.: Design of Memetic Algorithms for Scheduling and Timetabling Problems. In: Krasnogor, N., Hart, W., Smith, J. (eds.): Recent Advances in Memetic Algorithms and Related Search Technologies. Springer, Berlin (2004) 289–312
7. Burke, E. K., Newall, J. P.: Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings. Ann. Oper. Res. **129** (2004) 107–134
8. Burke, E. K., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (Pittsburgh, August 18–20)
9. Burke, E. K., Kendall, G., Soubeiga, E.: A Tabu Search Hyper-heuristic for Timetabling and Rostering. J. Heuristics **9** (2003) 451–470
10. Burke, E. K., Petrovic, S.: Recent Research Directions in Automated Timetabling. Eur. J. Oper. Res. **140** (2002) 266–280
11. Burke, E. K., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Konstanz, Germany, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001)
12. Burke, E. K., Bykov, Y., Petrovic, S.: A Multicriteria Approach to Examination Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 118–131
13. Burke, E. K., Newall, J. P.: A Multi-Stage Evolutionary Algorithm for the Timetable Problem. IEEE Trans. on Evol. Comput. **3** (1999) 63–74
14. Burke, E. K., Newall, J. P., Weare, R. F.: Initialisation Strategies and Diversity in Evolutionary Timetabling. Evol. Comput. **6** (1998) 81–103
15. Burke, E. K., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996)
16. Burke, E. K., Newall, J. P., Weare, R. F.: A Memetic Algorithm for University Exam Timetabling. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 241–250
17. Burke, E. K., Elliman, D. G., Ford, P., Weare, R. F.: Examination Timetabling in British Universities—A Survey. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 76–92.
18. Carter M. W., Laporte G., Lee, S. Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
19. Di Gaspero L., Schaerf A.: Tabu Search Techniques for Examination Timetabling. In: Burke, E. K., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Konstanz, Germany, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 104–117
20. Klir, G., Folger, T.: Fuzzy Sets, Uncertainty, and Information. Prentice-Hall, Englewood Cliffs, NJ (1988)

21. Landa Silva, J. D., Burke E. K., Petrovic, S.: An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling. In: Gandibleux, X., Sevaux, M., Sorensen, K., T'Kindt, V. (eds.): MetaHeuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems, Vol. 535. Springer, Berlin (2004) 91–129
22. Laporte, G., Desroches, S.: Examination Timetabling by Computer. Comput. Oper. Res. **11** (1984) 351–360
23. Meyer auf'm Hofe, H.: Solving Rostering Tasks as Constraint Optimization. In: Burke, E. K., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Konstanz, Germany, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 191–212
24. Moscato, P., Norman, M.: A "Memetic" Approach for the Travelling Salesman Problem—Implementation of a Computational Ecology for Combinatorial Optimisation on Message Passing Systems. In: Proceedings of the International Conference on Parallel Computing and Transputer Applications. IOS Press, Amsterdam (1992) 177–186
25. Negnevitsky M.: Artificial Intelligence—A Guide to Intelligent Systems. Addison-Wesley, Reading, MA (2002)
26. Paquete, L. F., Fonseca, C. M.: A Study of Examination Timetabling with Multiobjective Evolutionary Algorithms. In: Proceedings of 4th Metaheuristic International Conference (MIC 2001, Porto) 149–154
27. Pedrycz, W., Gowide, F.: An Introduction to Fuzzy Sets—Analysis and Design. MIT Press, Cambridge, MA (1998)
28. Petrovic, S., Yang, Y., Dror, M.: Case-based Initialisation of Metaheuristics for Examination Timetabling. In: Kendall, G., Burke, E., Petrovic, S., Gendreau, M. (eds.): Multidisciplinary Scheduling Theory and Applications. Springer, Berlin (2005) 289–308
29. Petrovic, S., Burke, E.: Educational Timetabling. Chapter 45 in: Leung, J. (ed.): Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman and Hall/CRC Press, London (2004) 45.1–45.23
30. Petrovic, S., Bykov, Y.: A Multiobjective Optimisation Technique for Exam Timetabling Based on Trajectories. In: Burke, E. K., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 179–192
31. Prade, H.: Computational Approach to Approximate and Plausible Reasoning. IEEE Trans. on Pattern Analysis and Machine Intelligence **7** (1985) 260–283
32. Ross P., Hart E., Corne E. D.: Some Observations about GA based Timetabling. In: Burke, E. K., Carter, M. (eds.): The Practice and Theory of Automated Timetabling II (PATAT'97, Selected Papers). Lecture Notes in Computer Science, Vol. 1408. Springer, Berlin (1998) 115–129
33. Ruspini, E., Bonissone, P., Pedrycz, W. (eds.): Handbook of Fuzzy Computation. Institute of Physics Publishing, Bristol (1998)
34. Slowinski, R., Hapke, M.: (eds.): Scheduling Under Fuzziness. Physica, Heidelberg (2000)
35. Thompson, J. M., Dowsland, K. A.: Variants of Simulated Annealing for the Examination Timetabling Problem. Ann. Oper. Res. **63** (1996) 105–128
36. White, G. M., Xie, B. S., Zonjic, S.: Using Tabu Search with Longer-term Memory and Relaxation to Create Examination Timetables. Eur. J. Oper. Res. **153** (2004) 80–91

37. Wren, A.: Scheduling, Timetabling and Rostering—A Special Relationship? In: Burke, E. K., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 46–75
38. Yang, Y., Petrovic, S.: A Novel Similarity Measure for Heuristic Selection in Examination Timetabling. In: Burke, E., Trick, M. (eds.): PATAT 2004—Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling 377–396 (also in this volume, pp. 245–267)
39. Zadeh, L. A.: Fuzzy Sets. Inform. Control **8** (1965) 338–353
40. Zadeh, L. A.: Theory of Approximate Reasoning. In: Hayes, J., Michie D., Mikulich, L. (eds.): Machine Intelligence, Vol. 9. Halstead Press, New York (1979) 149–194

Appendix

A triangular norm $*$:[0,1]×[0,1]→[0,1] is a function with the following properties [31]:

(i) *commutativity* $a*b = b*a$
(ii) *associativity* $a *(b*c) = (a*b)*c$
(iii) *monotonicity* if $a \leq b$ and $c \leq d$, then $a*c \quad \leq \quad b*d$
(iv) $a*1 = a$ *and* $a*0 = 0$.

A triangular conorm $+$:[0,1]×[0,1]→[0,1] is a function with the following properties [31]:

(i) *commutativity* $a+b = b+a$
(ii) *associativity* $a +(b+c) = (a+b)+c$
(iii) *monotonicity* if $a \leq b$ and $c \leq d$, then $a+c \quad \leq \quad b+d$
(iv) $a+0 = a$ *and* $a+1 = 1$.

Fuzzy Multiple Heuristic Orderings for Examination Timetabling

Hishammuddin Asmuni[1], Edmund K. Burke[1],
Jonathan M. Garibaldi[1], and Barry McCollum[2]

[1] School of Computer Science and Information Technology,
University of Nottingham, Jubilee Campus, Wollaton Road,
Nottingham, NG8 1BB, UK
{hba, ekb, jmg}@cs.nott.ac.uk
[2] School of Computer Science,
Queen's University Belfast,
Belfast BT7 1NN, UK
b.mccollum@qub.ac.uk

Abstract. In this paper, we address the issue of ordering exams by simultaneously considering two separate heuristics using fuzzy methods. Combinations of two of the following three heuristic orderings are employed: largest degree, saturation degree and largest enrolment. The fuzzy weight of an exam is used to represent how difficult it is to schedule. The decreasingly ordered exams are sequentially chosen to be assigned to the last slot with least penalty cost value while the feasibility of the timetable is maintained throughout the process. Unscheduling and rescheduling exams is performed until all exams are scheduled. The proposed algorithm has been tested on 12 benchmark examination timetabling data sets and the results show that this approach can produce good quality solutions. Moreover, there is significant potential to extend the approach by including a larger range of heuristics.

1 Introduction

Examination timetabling is essentially the problem of allocating exams to a limited number of time periods in such a way that none of the specified hard constraints are violated. A timetable which satisfies all hard constraints is often called a *feasible* timetable. In addition to the hard constraints, there are often many soft constraints whose satisfaction is desirable (but not essential). The set of constraints which need to be satisfied is usually very different from one institution to another, as reported by Burke et al. [10]. However, a common hard constraint across all problem instances is the requirement to avoid any student being scheduled for two different exams at the same time.

In practice, each institution usually has a different way of evaluating the quality of a feasible timetable. In many cases, the measure of quality is calculated based upon a penalty function which represents the degree to which the soft constraints are satisfied.

E. Burke and M. Trick (Eds.): PATAT 2004, LNCS 3616, pp. 334–353, 2005.

Over the years, numerous approaches have been investigated and developed for exam timetabling. Such approaches include constraint programming [18], [26], [30], [34], graph colouring [13], [15], [21], case-based reasoning [17], hyper-heuristics [13] and various metaheuristic approaches including greedy local search [19], [23], genetic algorithms [11], [28], tabu search [29], simulated annealing [41], the great deluge algorithm [7], [14] and hybridized methods [34] which draw on two or more of these techniques. Discussions about other approaches can be found in papers by Bardadym [4], Burke et al. [12], Burke and Petrovic [16], Carter [20], Carter and Laporte [22], De Werra [27], Petrovic and Burke [35] and Schaerf [39].

Since being introduced by Zadeh in 1965 [43], fuzzy methodologies have been successfully applied in a wide range of real-world applications. In scheduling and timetabling applications, fuzzy evaluation functions have been utilised in a number of different applications. For example, Dahal et al. [25] considered such approaches in generator maintenance scheduling, and Abboud et al. [1] used fuzzy target gross sales (fuzzy goals) to find "optimal" solutions of a manpower allocation problem, where several company goals and salesmen constraints need to be considered simultaneously. Fuzzy methodologies have been investigated for other timetabling problems such as aircrew rostering [40], driver scheduling [31] and nurse rostering [3].

In the specific context of examination timetabling, fuzzy methods have been implemented for measuring the problem similarity in case-based reasoning by Yang and Petrovic [42]. In this work, a fuzzy similarity measure is used to retrieve a good heuristic ordering for a new problem based on comparison with previous problems that are stored in the case base. The selected heuristic ordering is then applied to the new problem for generating an initial solution before applying the Great Deluge Algorithm in the improvement stage. Their results indicated that the performance of this algorithm is better when this fuzzy similarity measure is applied in the initialisation stage compared to other initialisation approaches.

In [37], Petrovic et al. employed fuzzy methodologies to measure the satisfaction of various soft constraints. The authors described how they modelled two soft constraints, namely *constraint on large exam* and *constraint on proximity of exams*, in the form of fuzzy linguistic terms and defined the related rule set. A memetic algorithm was then implemented to evaluate their approach on the same 12 benchmark problem instances that are considered in this paper.

Approaches which order the events prior to assignment to a period have been discussed by several authors including Boizumault et al. [5], Brailsford et al. [6], Burke et al. [9], Burke and Newall [15], Burke and Petrovic [16] and Carter et al. [21]. In the context of the same benchmark data sets used in our experiments, this sequencing strategy has been implemented by Carter et al. [21], Burke and Newall [15] and Burke et al. [13]. In [21], the authors used four different types of heuristic ordering to rank the exams in decreasing order to estimate how difficult it is to schedule each of the exams. They considered largest degree, saturation degree, largest weighted degree and largest enrolment. These heuristics were used individually each time they wanted to order the exams. Then, the exams were selected sequentially and assigned to a time slot that satisfied all the specified constraints. If no clash-free time slot was found, backtracking was implemented. The process was continued until all the exams were scheduled and a feasible solution was produced. Burke and Newall [15] applied an adaptive heuristic technique in which they start ordering by a particular heuristic and then alter that heuristic ordering to take into account the penalty that exams are imposing upon the timetable. Recently, Burke et al. [13] proposed a new hyper-heuristic approach for solving timetabling problems. Instead of using a single heuristic to find solutions for course and examination timetabling problems, a sequence of heuristics is applied. The authors used tabu search and deepest descent local search in order to find the best list of heuristics to guide the constructive algorithm in finding the "best solution" for each problem instance.

In this paper, a fuzzy methodology is used to rank exams based on an assessment of how difficult they are to schedule taking into account multiple heuristics. This paper is motivated by the observation that the consideration of more than one heuristic to rank the exams may lead to rankings that better reflect the actual difficulty of placing the exam, as several factors are simultaneously taken into account. The fuzzy multiple heuristic ordering method described in this paper should not be confused with multi-criteria approaches to examination timetabling, such as those described in Arani and Lotfi [2], Burke et al. [8], Lotfi and Cerveny [33], and Petrovic and Bykov [36]. In our approach, two heuristic orderings are simultaneously considered to rank the exams, whereas [2], [8], [33], [36] employ multi-criteria approaches to evaluate timetabling solutions and describe approaches which can handle this.

In the following section, the proposed algorithm and the experiments carried out are explained in detail. Section 3 describes the results obtained. These results are discussed and some concluding comments presented in Sections 4 and 5 respectively.

2 Methods

2.1 The Basic Sequential Construction Algorithm

There is a well known analogy between a basic version of the timetabling problem (no soft constraints) and the graph colouring problem (see [9]). Indeed, some of the best known timetabling heuristics are based upon graph colouring heuristics and these can be employed within a basic and simple timetabling algorithm (see Figure 1). We consider three different versions of the basic algorithm, which employ three different heuristic orderings respectively and require the following steps to assign all exams to a time slot. First, the exams are ordered (most difficult first) by applying one of the ordering heuristics. The following heuristics are employed as ordering criteria:

Largest Degree (LD) First. The degree of an exam is simply a count of the number of other exams which conflict in the sense that students are enrolled in both exams. This heuristic orders exams in terms of those with the highest degree first.

```
Sort unscheduled exams using selected heuristic ordering;
Insert exams into the last timeslot with least penalty;
While there exist unscheduled exam
   Perform the process for scheduling the unscheduled exams;
   Sort unscheduled exams using selected heuristic ordering;
End while
```

Fig. 1. Pseudo-code for general framework of sequential construction algorithm

Largest Enrolment (LE) First. The number of students enrolled for each exam is used to order the exams (the highest number of students first).

Least Saturation Degree (SD) First. The number of time slots available is used to order the exams. The basic motivation is that exams with less time slots available are more likely to be difficult to be scheduled. The fewer time slots that are available, the higher up the ordering is the exam.

Then, exams are selected sequentially from the ordering and assigned a valid time slot that will cause the minimum penalty cost for that exam. If no clash-free time slot is available, the exam is skipped and the process continued with the next exam. The skipped exams are then revisited and a process for scheduling the unscheduled exams is carried out (see Figure 2).

The sequential construction algorithm used here is similar to the approach applied by Carter et al. [21] with some modification. Basically, there are three differences between these two algorithms. The first difference is in the initial stage of the algorithm. In our algorithm we apply the heuristic ordering to *all* exams, whereas Carter et al.'s algorithm first finds the maximum-clique of examinations and assigns them to different time slots, and then applies heuristic ordering to the remaining exams. The second difference is in the selection of a free time slot. A search is carried out to find the clash free time slot with least penalty cost in order to assign each exam to a time slot. In our algorithm, if several time slots are available, then the last available time slot in the list will be selected. (It was found that the choice of assigning exams to the last available time slot or the first available time slot did not make much difference, as the main purpose of this was simply to spread out the student's timetable.) In contrast, Carter et al. chose the first clash-free time slot found in which to assign the exam.

Thirdly, for reshuffling a scheduled exam, we randomly select a time slot from the list of time slots with the same minimum number of scheduled exams that needed to be "bumped back", whereas Carter et al. used *minimum disruption cost* to break any ties. A detailed overview of the "unschedule and reschedule scheduled exams" algorithm is shown in Figure 2.

2.2 The Fuzzy Multiple Heuristic Ordering

In many decision making environments, it is often the case that several factors are simultaneously taken into account. Often, it is not known which factor(s) need to be emphasized more in order to generate a better decision. Somehow a trade-off between the various (potentially conflicting) factors must be made.

```
k := number of unscheduled exams;
For u := 1 to k
    Select exam[u];
    Find timeslots where exam[u] can be inserted with minimum number of
    scheduled exams need to be removed from the timeslot;
    If found more than one slot with the same number of scheduled exams
    need to be removed
      Select a timeslot randomly from the candidate list of slots, ts;
    End if
    c :=  number of exam in timeslot $ts_u$ that conflict with exam[u];
    For m := 1 to c
       Select exam[m];
      If found another timeslot with minimum cost to move exam[m]
        Move exam[m] to the timeslot;
      else
        Bump back exam[m] to unscheduled exam list;
      End if
    End for
    Insert exam[u] to timeslot $ts_u$;
    Remove exam[u] from unscheduled exam list;
End for
```

Fig. 2. Pseudo-code for rescheduling the scheduled exams

The general framework of fuzzy reasoning facilitates the handling of such uncertainty. A fuzzy set A of a universe of discourse X (the range over which the variable spans) is characterized by a *membership function* $\mu_A : X \rightarrow [0, 1]$ which associates with each element x of X a number $\mu_A(x)$ in the interval $[0, 1]$, with $\mu_A(x)$ representing the *grade of membership* of x in A. The precise meaning of the membership grade is not rigidly defined, but is supposed to capture the "compatibility" of an element to the notion of the set.

Fuzzy systems are used for representing and employing knowledge that is imprecise, uncertain, or unreliable. They usually consist of four main interconnected components: an input fuzzifier, a set of rules, an inference engine, and an output processor (defuzzifier). Rules which connect input variables to output variables in "IF ... THEN ..." form are used to describe the desired system response in terms of *linguistic* variables (words) rather than mathematical formulae. The "IF" part of the rule is referred to as the "antecedent", the "THEN" part is referred to as the "consequent". The number of rules depends on the number of inputs and outputs, and the desired behaviour of the system. Once the rules have been established, such a system can be viewed as a non-linear mapping from inputs to outputs. It is not appropriate to present a full description of the functioning of fuzzy systems here; the interested reader is referred to Cox [24] for a simple treatment or Zimmerman [44] for a more complete treatment.

The fuzzy inference process is illustrated for a three-rule system based on two input variables, LD and LE. Each of the input and output variables are associated with three linguistic terms; fuzzy sets corresponding to meanings of *small*, *medium* and *high*. These membership functions are chosen arbitrarily to span the universe of discourse of the variable. A rule set connecting the input variables (LD and LE) to a single output variable, *examweight*, is constructed. The following three rules are used to illustrate the behaviour of this example system (note that this is only an illustrative example; the membership functions and rules used in each actual experiment are described in Section 2.3):

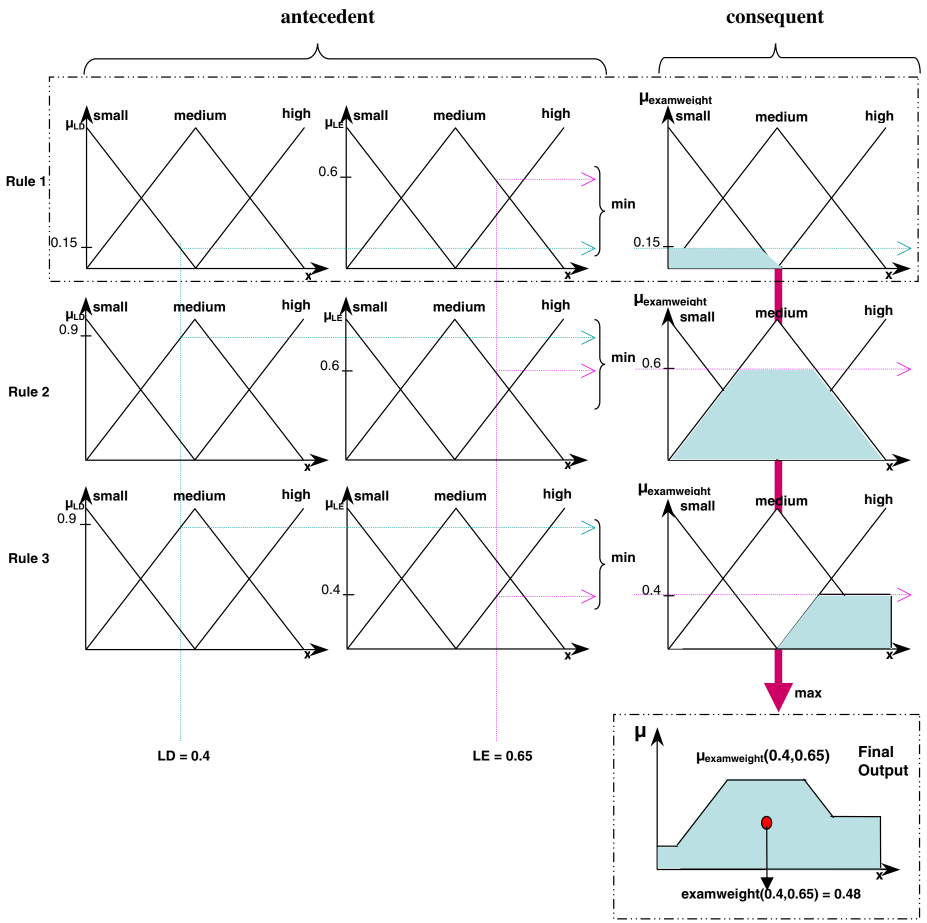


Fig. 3. A three-rule Mamdani inference process

Rule 1: IF (*LD* is *small*) AND (*LE* is *medium*) THEN (*examweight* is *small*)
Rule 2: IF (*LD* is *medium*) AND (*LE* is *medium*) THEN (*examweight* is *medium*)
Rule 3: IF (*LD* is *medium*) AND (*LE* is *high*) THEN (*examweight* is *high*)

The first stage is to normalize the input values within the range $[0, 1]$. The transformation is as follows:

$$v' = \frac{(v - minA)}{(maxA - minA)}$$

where v is the actual value in the initial range $[minA, maxA]$. For example, if $v = 10$ in $[0, 20]$, the normalized value v' is 0.5 in the new range $[0, 1]$.

Figure 3 illustrates the inferencing of this system (a Mamdani inference process) with normalized values for *LD* and *LE* of 0.4 and 0.65, respectively. For each rule in turn, the fuzzy system operates as follows. The input component ("fuzzifier") computes the membership grade for each input variable based on the membership functions defined. That is, in *Rule 1*, the membership grade is computed for LD in the fuzzy set *small* and for LE in the fuzzy set *medium*. As shown in the figure, the determined grades of membership for each input variable are

$$\mu_{small}(LD = 0.4) = 0.15$$

and

$$\mu_{medium}(LE = 0.65) = 0.6\,.$$

With these fuzzified values, the inference engine then computes the overall truth value of the antecedent of the rule (*Rule 1*) by applying the appropriate fuzzy operators corresponding to any connective(s) (AND or OR). In the example, the fuzzy AND operator is implemented as a minimum function:

$$\begin{aligned} &\textit{Rule 1} \quad \text{IF } (LD \text{ is } \textit{small}) \text{ AND } (LE \text{ is } \textit{medium}) \\ \mu_{Rule1} &= \mu_{small}(LD = 0.4) \ \wedge \ \mu_{medium}(LE = 0.65) \\ &= min(0.15, 0.6) \\ &= 0.15\,. \end{aligned}$$

Next, the inference engine applies the implication operator to the rule in order to obtain the fuzzy set to be accumulated in the output variable. In this case, inferencing is implemented by truncating the output membership function at the level corresponding to the computed degree of truth of the rule's antecedent. The effect of this process can be seen in the *consequent* part of *Rule 1* in which the membership function for the linguistic term *small* was truncated at the level of 0.15. The same processes are applied to the rest of the rules in turn.

Finally, all the truncated output membership functions are aggregated together to form a single fuzzy subset (labelled as *Final Output* in Figure 3) by taking the maximum across all the consequent sets. A further step (known as "defuzzification") is then performed if (as is usual) the final fuzzy output is to be translated into a crisp output. We applied a common form of this process, termed "centre of gravity defuzzification" as it is based upon the notion of finding the centroid of a planar figure, as given by

$$\sum_i \frac{\mu(x_i) \cdot x_i}{\mu(x_i)}\,.$$

In the example of Figure 3, the output for the fuzzy system (that represents how difficult the exam is to be scheduled) is 0.48 for the given inputs (i.e an exam with LD and LE of 0.4 and 0.65, respectively).

All exams in the given problem instance are evaluated using the same fuzzy system, and the sequential constructive algorithm uses the crisp output of each exam for ordering all exams. The exam with the biggest crisp value is selected to be scheduled first, and the process continues until all the exams are scheduled without violating any of the hard constraints.

2.3 Description of Experiments

A number of experiments were carried out in which progressively more sophisticated fuzzy mechanisms were created to order the exams. In each experiment this ordering is simply inserted into the basic general algorithm presented in Figure 1.

Single Heuristic Ordering. In order to provide a comparative test, the algorithm was initially run without implementing fuzzy ordering. That is, in this approach, the exams in the problem instances were ordered based on a single heuristic ordering. All the exams were then selected to be scheduled based on this ordering.

Fixed Fuzzy LD+LE Model. Next, a fixed fuzzy model that took into account multiple heuristic ordering was implemented. Two out of the three ordering heuristics described in Section 2.1, namely *largest degree* (LD) and *largest enrolment* (LE), were selected as input variables. The membership functions used in this experiment are shown in Figure 4. The choice of these membership functions was based on "trial and error" to test how the algorithm would work when exams were ordered with the aid of fuzzy reasoning.

The fuzzy rules used in this experiment are shown in Table 1. For simplicity, the fuzzy rules are expressed as a linguistic matrix (see [32]). In such a linguistic matrix, the left-most column and the first row denote the variables involved in the antecedent part of the rules. The second column contains the linguistic terms applicable to the input variable shown in the first column; those in the second row correspond to the input variable shown in the first row. Each entry in the main body of the matrix denotes the linguistic values of the consequent part of a rule. For instance, the bottom-right entry in Table 1 is read as *"IF LD is high AND LE is high THEN examweight is very high"*. The same representation is also used to express the fuzzy rule sets for the experiments explained in the following sections. Note that, in addition to the three basic terms, the *hedge* "very" was utilized to create two extra terms for the output variable. The "very" hedge squares the membership grade $\mu(x)$ at each x of the fuzzy set for the term to which it is applied. Thus the membership function of the fuzzy set for "very small" is obtained by squaring the membership function of the fuzzy set "small".

Tuned Fuzzy LD+LE Model. Fuzzy modelling can be thought of as the task of designing a fuzzy inference system. The selection of important parameters

Table 1. Fuzzy rule set for fixed fuzzy LD+LE model

		LE		
		S	M	H
LD	S	VS	VS	M
	M	M	M	H
	H	S	M	VH

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

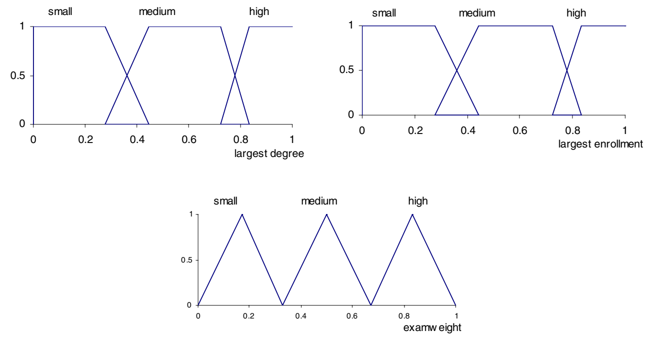


Fig. 4. Membership functions for fixed fuzzy LD+LE model

Table 2. Fuzzy rule set for tuned fuzzy LD+LE model

		LE		
		S	M	H
LD	S	VS	S	M
	M	S	M	H
	H	M	H	VH

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

for the inference system is crucial, as the overall system behaviour is highly dependent on a large number of factors, such as how the membership functions are chosen, the number of rules involved, the fuzzy operator used, and so on. For the purpose of finding a better fuzzy model, a relatively straightforward tuning procedure was implemented in order to investigate whether the initial choice of fuzzy model was appropriate. This tuning procedure was then applied to different combinations of multiple ordering heuristics.

As an initial extension to the *Fixed Fuzzy LD+LE Model*, a restricted form of exhaustive search was used to find the most appropriate shape for the fuzzy membership functions in the system. There are very many alternatives that may be used when constructing a fuzzy model. In our implementation, we arbitrarily restricted the search based on the membership functions as shown in Figure 5. Triangular shape membership functions were employed to represent *small*, *medium* and *high*. However, the fuzzy model was then altered by moving the point *cp* along the universe of discourse. This single point corresponded to the right edge for the term *small*, the centre point for the term *medium* and the left edge for the term *high*. Thus, there was one *cp* parameter for each fuzzy variable (two inputs and one output).

A search was then carried out to find the best set of *cp* parameters. During this search, each point *cp* (for any of the fuzzy variables) can take a value between 0.0 and 1.0 inclusive. Increments of 0.1 were used (i.e. the values 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0) for data sets that have 400 and fewer exams, and 0.25 increments (i.e. the values 0.0, 0.25, 0.5, 0.75 and 1.0) for data

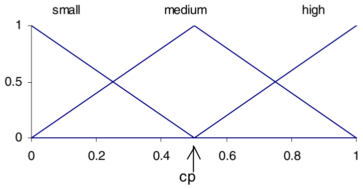
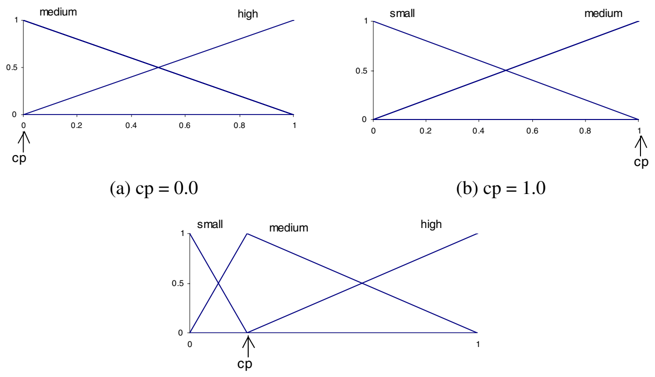


Fig. 5. The membership function for tuned fuzzy model



(a) cp = 0.0

(b) cp = 1.0

(c) 0 < cp <1

Fig. 6. Range of possible membership functions

Table 3. Fuzzy rule set for tuned fuzzy SD+LE model

		SD		
		S	M	H
LE	S	M	S	VS
	M	H	M	S
	H	VH	H	M

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

sets that have more than 400 exams. The effect of varying the point *cp* from 0.0 to 1.0 is shown in Figure 6.

In this experiment, the combination of LD and LE heuristics were again used as the fuzzy input variables. The fuzzy rule set used is shown in Table 2.

Tuned Fuzzy SD+LE Model. In this experiment, the same approach as above was employed, but now the combination of SD and LE were used as the fuzzy input variables. A new fuzzy rule set was required as the SD heuristic is reversed compared to the LD and LE heuristics. The fuzzy rule set is presented in Table 3.

3 Experimental Results

In this section the results obtained in each experiment are presented. In all experiments, the basic algorithm of Figure 1 was employed. The only difference was the heuristic ordering used. The experiments were carried out with 12 benchmark data sets made publicly available by Carter et al. Table 4 reproduces the problem characteristics.

A proximity cost function was used to measure the timetable quality. The maximum capacity for each time slot was not taken into account. Only feasible timetables were accepted. The penalty function is taken from Carter et al. [21]. It is motivated by the goal of spreading out each student's examination schedule. If two exams scheduled for a particular student are t time slots apart, the weight is set to $w_t = 2^{5-t}$ where $t \in \{1, 2, 3, 4, 5\}$. The weight is multiplied by the number of students that sit for both of the scheduled exams. The average penalty per student is calculated by dividing the total penalty by total number of students. The following formulation was used (adapted from Burke et al. [7]), in which the goal is to minimize

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

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

The algorithm was developed using Java-based object oriented programming. The fuzzy inference engine developed by Sazonov et al. [38] was implemented. The experiments were run on a PC with a 1.8 GHz Pentium 4 and 256MB of RAM. In the case of the *Single Heuristic Ordering* and the *Fixed Fuzzy LD+LE Model* each instance was run five times. In the other experiments (that involved tuning the fuzzy model), the aim was to search for the best fuzzy model to guide the constructive algorithm. In order to reduce the size of the search space, only the membership functions are tuned, whereas the fuzzy rule set is fixed. In this tuning process, for problem instances that have 400 and fewer exams, the algorithm was tested on 1331 (three variables and 11 options: 11^3) membership function combinations. Problem instances that have more than 400 exams were tested on 125 (three variables and five options: 5^3) membership function combinations. Because of this, each instance was only run twice. For all experiments, only the best results are selected and presented in Table 5.

For comparison, the best results obtained by Carter et al. [21] when using various different heuristics to order the exams are shown in column 2 of Table 5. The results obtained for our three varieties of *Single Heuristic Ordering* are presented in columns 3–5. The results obtained for the *Fixed Fuzzy LD+LE Model* are shown in column 6. In general, these results are worse than for the best *Single Heuristic Ordering*, except for the STA-F-83 data set, where the fixed fuzzy model obtained the best result. This observation suggested that there might be promise in the fuzzy approach and prompted us to undertake further investigations with tuned fuzzy models. The results for the *Tuned Fuzzy LD+LE*

Table 4. Examination timetabling problem characteristics

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

Table 5. Experimental results for single and fuzzy heuristic orderings

		Single Heuristic Ordering			Fixed fuzzy	Tuned fuzzy	Tuned fuzzy
Data set	Carter et al. [21]	LD	LE	SD	LD+LE model	LD+LE model	SD+LE model
CAR-F-92	6.2	5.56	5.03	5.50	5.65	4.62	**4.56**
CAR-S-91	7.1	6.38	5.90	5.91	6.31	5.60	**5.29**
EAR-F-83	36.4	40.58	45.88	49.10	48.14	38.41	**37.02**
HEC-S-92	10.8	14.98	14.94	14.27	16.93	12.53	**11.78**
KFU-S-93	14.0	18.63	16.46	18.60	18.29	16.53	**15.81**
LSE-F-91	10.5	15.08	14.52	13.46	16.84	12.35	**12.09**
RYE-F-92	7.3	12.95	11.12	11.60	12.98	11.75	**10.38**
STA-F-83	161.5	173.09	171.87	178.24	161.21	**160.42**	160.75
TRE-S-92	9.6	10.98	9.93	10.81	10.36	9.05	**8.67**
UTA-S-92	3.5	4.48	4.78	3.83	5.16	3.87	**3.57**
UTE-S-92	25.8	35.19	28.80	33.14	30.54	28.65	**28.07**
YOR-F-83	41.7	45.60	43.53	45.27	46.41	41.37	**39.80**

Model are shown in column 7 and those for the *Tuned Fuzzy SD+LE Model* in column 8.

The best fuzzy results obtained in Table 5 are highlighted in bold font. The corresponding membership functions of the fuzzy model which obtained the best result for each data set are presented in Figures 7 and 8. It can be seen that the membership functions differ in each case: i.e., there is no generic fuzzy model which suits all the data sets.

4 Discussion

Amongst the three *Single Heuristic Ordering*, it would appear that *LE* is the "best" in this context as it produced the best solution for eight out of the 12 data sets, compared to only one for LD (for EAR-F-83) and three for *SD* (for HEC-S-92, LSE-F-91 and UTA-S-92). It also can be seen that, when compared to Carter et al.'s best results, our simplified version of their algorithm produced worse results in 10 out of the 12 data sets, but a slightly better timetable was obtained for the CAR-F-92 and CAR-S-91 cases. The *Fixed Fuzzy LD+LE Model* only achieves a better result than the best *Single Heuristic Ordering* in one out of the 12 data sets (STA-F-83). However, the rules and membership functions for this initial fuzzy model were completely arbitrary, so it may be surprising that it achieved a best result even once.

It is evident that the *Tuned Fuzzy LD+LE Model* produced better results than the *Fixed Fuzzy LD+LE Model* in all cases. Although entirely expected, this observation was taken as confirmation that the fuzzy system was capturing meaningful information and that the tuning procedure, although not finding the truly optimal fuzzy model (in the sense of the globally best set of membership functions for the given set of rules and other fixed aspects of the fuzzy system), was operating successfully. In comparison with best *Single Heuristic Ordering*, the *Tuned Fuzzy LD+LE Model* obtained better results in all cases except for the KFU-S-93, RYE-F-92 and UTA-S-92 data sets.

The *Tuned Fuzzy SD+LE Model* went on to produce better results than the *Tuned Fuzzy LD+LE Model* for all cases except the STA-F-83 data set. When compared to Carter et al.'s original results, the tuned fuzzy models operating on two heuristics simultaneously (taking the best tuned fuzzy model for each data set) obtained better results for five out of the 12 data sets. These were the CAR-F-92, CAR-S-91, STA-F-83, TRE-S-92 and YOR-F-83 data sets. Although these results have since been bettered by many authors (see the discussion of Table 6 below), these have been based on iterative improvement techniques rather than the constructive approach employed by Carter et al. and ourselves.

Initially, the choice to use a combination of the *LD* and *LE* heuristics was based on the fact that these heuristics are static in the sense that they only have to be calculated once at the beginning of the ordering process. In contrast, the *SD* heuristic must be recalculated after each exam is assigned to a slot. Thus, it was felt that tuning the fuzzy model based on the $LD + LE$ combination would be quicker. The choice to use the $SD + LE$ combination in the subsequent model was based on the observation that the *LE* heuristic ordering, when used alone, obtained the minimum penalty cost for eight out of the 12 data sets while the *SD* heuristic ordering obtained the minimum cost for three out of 12. Thus it was felt that these offered the most promising combination of two heuristics.

The design of the fuzzy rule sets was based on three assumptions:

- if *LD* is *High* then examweight is *High*
- if *LE* is *High* then examweight is *High*
- if *SD* is *Small* then examweight is *High*.

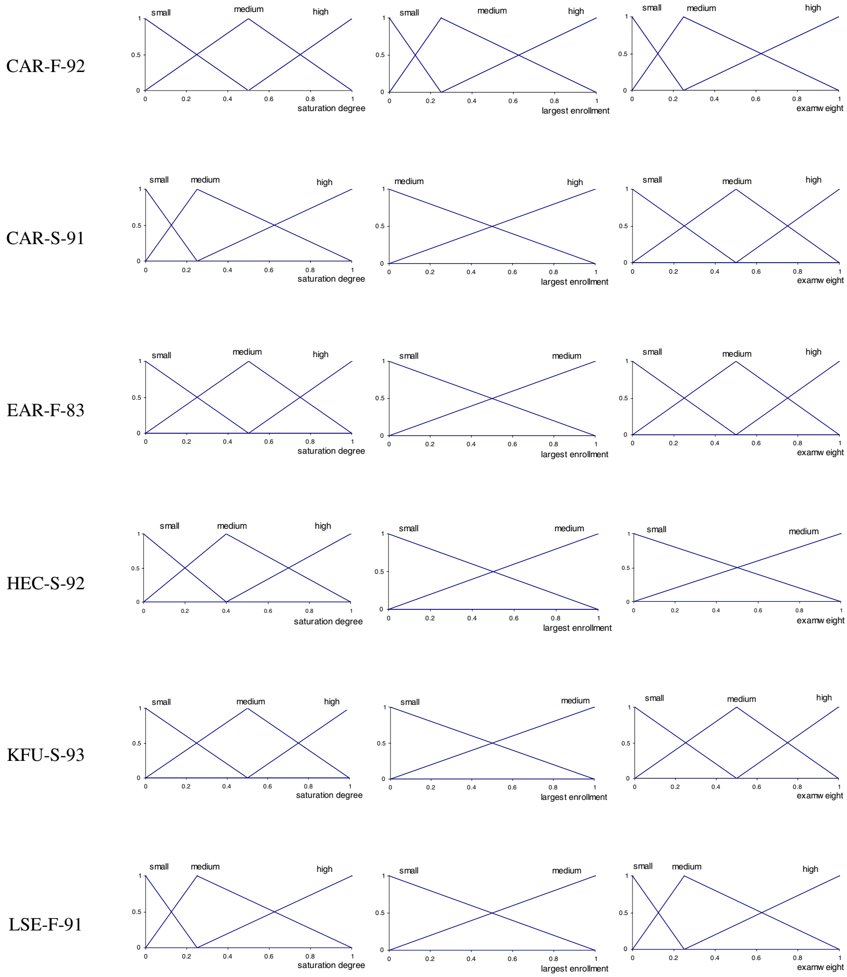


Fig. 7. Best fuzzy model for data sets CAR-F-92, CAR-S-91, EAR-F-83, HEC-S-92, KFU-S-93 and LSE-F-91

However, it must be emphasized that the rule sets specified in Tables 1–3 are only one possible instance (in the case of each experiment) out of a very large number of alternatives. Due to the very large number of degrees-of-freedom in any fuzzy model, it is very rare that the first fuzzy system constructed will perform at an acceptable level. Usually some form of optimization or performance tuning of the system will need to be undertaken. The most significant influences on performance of a fuzzy system are likely to be the number and location of the membership functions and the number and form of the rules. In our implementation, the number and form of the rules are kept fixed in all cases. Although the fuzzy membership functions were, to a certain extent, tuned to

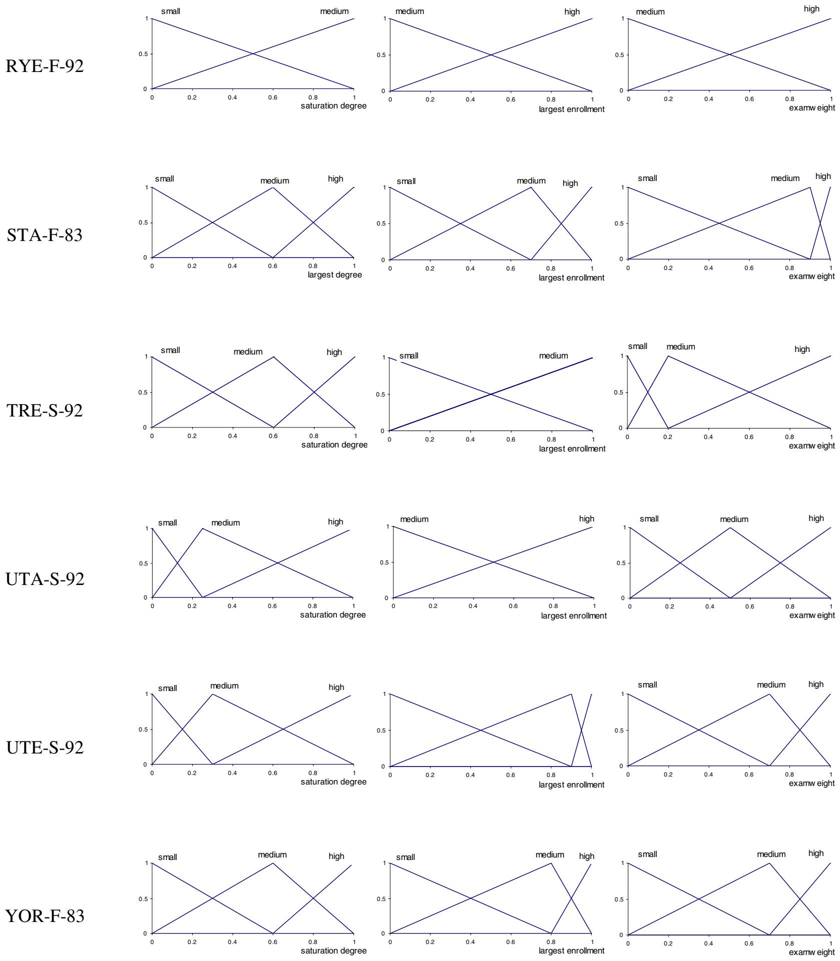


Fig. 8. Best fuzzy model for data sets RYE-F-92, STA-F-83, TRE-S-92, UTA-S-92, UTE-S-92 and YOR-F-83

obtain good performances, there was no attempt in the current work to tune the rule sets. It is highly likely that, given sufficient time to perform the tuning, a set of fuzzy rules leading to better performance of the fuzzy models could be obtained.

In Table 5, we have demonstrated that, in all cases, tuning the fuzzy model produces better results, as might be expected. This confirms our hypothesis that simultaneous ranking of multiple heuristic ordering *can* produce better results. The fact that the best fuzzy results are all obtained using different fuzzy membership functions, as shown in Figures 7 and 8, means that no *generic* fuzzy model has been obtained at this stage. Such a generic model would be necessary if the approach is to be applied quickly and efficiently to novel data sets. The

Table 6. Results comparison

Data set	Our best results	Burke and Newall [14]	Burke et al. [7]	Caramia et al. [19]	Casey and Thompson [23]	Merlot et al. [34]
CAR-F-92	4.56	**4.10**	4.2	6.0	4.4	4.3
CAR-S-91	5.29	**4.65**	4.8	6.6	5.4	5.1
EAR-F-83	37.02	37.05	35.4	**29.3**	34.8	35.1
HEC-S-92	11.78	11.54	10.8	**9.2**	10.8	10.6
KFU-S-93	15.81	13.90	13.7	13.8	14.1	**13.5**
LSE-F-91	12.09	10.82	10.4	**9.6**	14.7	11.0
RYE-F-92	10.35	–	8.9	**6.8**	–	8.4
STA-F-83	160.42	168.73	159.1	158.2	**134.9**	157.3
TRE-S-92	8.67	8.35	**8.3**	9.4	8.7	8.4
UTA-S-92	3.57	**3.20**	3.4	3.5	–	3.5
UTE-S-92	27.78	25.83	25.7	**24.4**	25.4	25.1
YOR-F-83	40.66	37.28	36.7	**36.2**	37.5	37.4

lack of such a generic fuzzy model may cast doubt regarding the usability and flexibility of this approach. This indicates that care must be taken when applying fuzzy techniques: it is certainly not the case that just because it is fuzzy it is necessarily better.

Table 6 shows the performance of our algorithm in comparison with selected recently published results on Carter et al.'s benchmarks. The best result amongst the compared techniques for each data set is highlighted in bold font. Collectively, these results have been selected to show the best known results for each data set. Although our algorithm has not beaten the best known result for any data set, its performance is broadly competitive with the others in the sense that it it *not* the worst in 6 out of the 12 data sets. It is also worth pointing out that our algorithm produces solutions for all 12 data sets, and that in two of the cases where ours produces the worst result, at least one of the other papers did not quote any result. However, it has to be kept in mind that our method is a simple constructive initial solution, compared to the other methods which are iterative improvement approaches. Although our results are well behind more recent results, especially those of Caramia et al., interestingly our fuzzy constructive algorithm can beat Caramia et al.'s results for data sets CAR-F-92, CAR-S-91 and TRE-S-92.

Finally, some remarks should be made concerning the time required for our algorithm. In doing so, it is vital that a distinction must be made between the time taken to perform the tuning of the fuzzy models and the time taken to construct a solution once each fuzzy model is fixed. Once the fuzzy model is fixed, the time taken to construct a solution is no longer (in a practical sense) than the time taken when using a single heuristic ordering: that is, the additional time taken for the fuzzy system to perform its ordering is negligible. Indeed, there is some evidence (which we are investigating further at present) that, once the fuzzy model is fixed, solutions are constructed more quickly using the fuzzy ordering. It seems that this may be due to the lack of required backtracking when the fuzzy ordering is used. However, the time taken in tuning each fuzzy model is very significant. Of course, if a generic fuzzy model could be found—that is a single fuzzy model that produces good quality initial solutions for all data sets (including the 12 benchmark data sets used here and novel data sets)—then the approach could be widely adopted, with significant impact.

5 Conclusions

As far as the authors are aware, no other published work has described the exploration of fuzzy methodologies for simultaneously ordering exams in the construction of examination timetables. In this study, we have investigated a fuzzy methodology to use multiple heuristic ordering simultaneously. Our evaluation indicates that better solutions can be produced by this approach when compared against each of the heuristics alone. This is the key point of the paper. Our method does not produce any of the best benchmark results, but we only used a limited number of heuristics to demonstrate the potential. This paper has established that there is significant potential in taking this approach further by adding more heuristics.

We are encouraged by these promising initial results and aim to extend this work further. Future research avenues may include:

- investigating other combinations of heuristic ordering (using combinations of three or more heuristics),
- investigating different sets of fuzzy rules and fuzzy membership functions,
- exploring the use of more sophisticated optimization algorithms when tuning these and other fuzzy models, and
- testing the algorithms on course timetabling problems.

Acknowledgements

This work is supported by the Universiti Tekno-logi Malaysia (UTM) and the Ministry of Science, Technology and Innovation (MOSTI) Malaysia. The authors would also like to thank the four anonymous referees for their helpful suggestions.

References

1. Abboud, N., Inuiguichi, M., Sakawa, M., Uemura, Y.: Manpower Allocation Using Genetic Annealing. Eur. J. Oper. Res. **111** (1998) 405–420
2. Arani, T., Lotfi, V.: A Three Phased Approach to Final Exam Scheduling. IIE Trans. **21** (1989) 86–96
3. Meyer Aufm Hofe, H.: Solving Rostering Tasks by Generic Methods for Constraint Optimization. Int. J. Found. Comput. Sci. **12** (2001) 671–693
4. Bardadym, V. A.: Computer Aided School and University Timetabling: The New Wave. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 22–45

5. Boizumault, P., Delon, Y., Peridy, L.: Constraint Logic Programming for Examination Timetabling. J. Logic Programm. **26** (1996) 217–233
6. Brailsford, S. C., Potts, C. N., Smith, B. M.: Constraint Satisfaction Problems: Algorithms and Applications. Eur. J. Oper. Res. **119** (1999) 557-581
7. Burke, E. K., Bykov, Y., Newall, J., Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Trans. **36** (2004) 509–528
8. Burke, E. K., Bykov, Y., Petrovic, S.: A Multicriteria Approach to Examination Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 118–131
9. Burke, E. K., de Werra, D., Kingston, J.: Applications in Timetabling. Chapter in: Handbook of Graph Theory. Chapman and Hall/CRC Press, London (2003) 445–474
10. Burke, E. K., Elliman, D. G., Ford, P. H., Weare, R. F.: Examination Timetabling in British Universities—A Survey. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 76–90
11. Burke, E. K., Elliman, D. G., Weare, R. F.: A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. In: Proceedings of the 6th International Conference on Genetic Algorithms (ICGA'95, Pittsburgh, PA). Morgan Kaufmann, San Francisco (1995) 605–610
12. Burke, E. K., Jackson, K., Kingston, J. H., Weare, R. F.: Automated University Timetabling: The State of the Art. Comput. J. **40** (1997) 565–571
13. Burke, E. K., Meisels, A., Petrovic, S., Qu, R.: A Graph-Based Hyper Heuristic for Timetabling Problems. Eur. J. Oper. Res. (2005) to appear
14. Burke, E. K., Newall, J. P.: Enhancing Timetable Solutions with Local Search Methods. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 195–206
15. Burke, E. K., Newall, J. P.: Solving Examination Timetabling Problems Through Adaption of Heuristic Orderings. Ann. Oper. Res. **129** (2004) 107–134
16. Burke, E. K., Petrovic, S.: Recent Research Directions in Automated Timetabling. Eur. J. Oper. Res. **140** (2002) 266–280
17. Burke, E. K., Petrovic, S., Qu, R.: Case Based Heuristic Selection for Timetabling Problems. J. Scheduling (2005) to appear
18. Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996)
19. Caramia, M., DellOlmo, P., Italiano, G. F.: New Algorithms for Examination Timetabling. In: Naher, S., Wagner, D. (eds.): Algorithm Engineering 4th International Workshop, Proceedings WAE 2000 (Saarbrucken, Germany, September). Lecture Notes in Computer Science, Vol. 1982. Springer, Berlin (2001) 230–241
20. Carter, M. W.: A Survey of Practical Applications of Examination Timetabling Algorithms. Oper. Res. **34** (1986) 193–202
21. Carter, M. W., Laporte, G. G., Lee, S. Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
22. Carter, M. W., Laporte, G. G.: Recent Development in Practical Examination Timetabling. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 3–21

23. Casey, S., Thompson, J.: GRASPing the Examination Scheduling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 232–244
24. Cox, E., O'Hagen, M.: The Fuzzy Systems Handbook : A Practitioner's Guide to Building, Using and Maintaining Fuzzy Systems. AP Professional, Cambridge, MA (1998)
25. Dahal, K. P., Aldridge, C. J., McDonald, J. R.: Generator Maintenance Scheduling Using a Genetic Algorithm with a Fuzzy Evaluation Function. Fuzzy Sets and Systems **102** (1999) 21–29
26. Davenport, A. J., Tsang, E. P. K.: Solving Constraint Satisfaction Sequencing Problems by Iterative Repair. In: 1st Int. Conf. on the Practical Application of Constraint Technologies and Logic Programming (PACLP'99) 345–357
27. De Werra, D.: An Introduction to Timetabling. Eur. J. Oper. Res. **19** (1985) 151–162
28. Deris, S., Omatu, S., Ohta, H., Saad, P.: Incorporating Constraint Propagation in Genetic Algorithm for University Timetabling Planning. Eng. Appl. Artif. Intell. **12** (1999) 241–253
29. Di Gaspero, L., Schaerf, A.: Tabu Search Techniques for Examination Timetabling. In: Burke, E., Erben, W. (eds.): The Practice and Theory of Automated Timetabling III (PATAT'00, Selected Papers). Lecture Notes in Computer Science, Vol. 2079. Springer, Berlin (2001) 104–117
30. Guéret, C., Jussien, N., Boizumault, P., Prins, C.: Building University Timetables Using Constraint Logic Programming. In: Burke, E., Ross, P. (eds.): The Practice and Theory of Automated Timetabling I (PATAT'95, Selected Papers). Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin (1996) 130–145
31. Li, J., Kwan, R. S. K.: A Fuzzy Genetic Algorithm for Driver Scheduling. Eur. J. Oper. Res. **147** (2003) 334–344
32. Lim, M. H., Rahardja, S., Gwee, B. H.: A GA Paradigm for Learning Fuzzy Rules. Fuzzy Sets and Systems **82** (1996) 177–186
33. Lotfi, V., Cerveny, R.: A Final Exam-Scheduling Package. J. Oper. Res. Soc. **42** (1991) 205–216
34. Merlot, L. T. G., Boland, N., Hughes, B. D., Stuckey, P. J.: A Hybrid Algorithm for Examination Timetabling Problem. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 207–231
35. Petrovic, S., Burke, E. K.: University Timetabling. Chapter 45 in: The Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton, FL (2004)
36. Petrovic, S., Bykov, Y.: A Multiobjective Optimisation Technique for Exam Timetabling Based on Trajectories. In: Burke, E., De Causmaecker, P. (eds.): Practice and Theory of Automated Timetabling IV (PATAT'02, Selected Papers). Lecture Notes in Computer Science, Vol. 2740. Springer, Berlin (2003) 179–192
37. Petrovic, S., Patel, V., Yang, Y.: University Timetabling with Fuzzy Constraints. In: Burke, E. K., Trick, M. (eds.): Practice and Theory of Automated Timetabling V (PATAT'04, Selected Papers). Lecture Notes in Computer Science, Vol. 3616. Springer, Berlin (2005) to appear
38. Sazonov, E. S., Klinkhachorn, P., Gangarao, H. V. S., Halabe, U. B.: Fuzzy Logic Expert System for Automated Damage Detection from Changes in Strain Energy Mode Shapes. Non-destructive Testing and Evaluation **18** (2002) 1–20

39. Schaerf, A.: A Survey of Automated Timetabling. Artif. Intell. Rev. **13** (1999) 87–127
40. Teodorovic, D., Lucic, P.: A Fuzzy Set Theory Approach to the Aircrew Rostering Problem. Fuzzy Sets and Systems **95** (1998) 261–271
41. Thompson, J. M., Dowsland, K. A.: A Robust Simulated Annealing Based Examination Timetabling System. Comput. Oper Res. **25** (1998) 637–648
42. Yang, Y., Petrovic, S.: A Novel Similarity Measure for Heuristic Selection in Examination Timetabling. In: Burke, E. K., Trick, M. (eds.): Practice and Theory of Automated Timetabling V (PATAT'04, Selected Papers). Lecture Notes in Computer Science, Vol. 3616. Springer, Berlin (2005) to appear
43. Zadeh, L. A.: Fuzzy Sets. Inform. Control **8** (1965) 338–353
44. Zimmerman, H. J.: Fuzzy Set Theory and Its Applications. 3rd edn. Kluwer, Dordrecht (1996)

Author Index