PATAT 2010

8th International Conference on the Practice and Theory of Automated Timetabling

Queen's University Belfast, Northern Ireland, 10th - 13th August 2010

CONFERENCE PROCEEDINGS

PATAT 2010

Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling

10 - 13 August 2010 (Queen's University of Belfast)

Edited by:

Barry McCollum, Queen's University Belfast, UK
Edmund Burke, University of Nottingham, UK
George White, University of Ottawa, Canada

ISBN 08-538-9973-3
Published by Queen's University Belfast

Preface

On behalf of the Steering Committee and the Programme Committee of the PATAT (Practice and Theory of Automated Timetabling) series of conferences, we would like to welcome you to the eighth conference here in Belfast. The PATAT conferences, which are held every two years, bring together researchers and practitioners in all aspects of computer-aided timetable generation and related issues. This includes personnel rostering, school timetabling, sports scheduling, transportation timetabling and university timetabling. It is worthy pointing out that this conference is being held at a time when the current world wide economic downturn is fueling the need for innovative approaches to the management and planning of resources. Fostering the development of leading edge research techniques in underpinning innovate timetabling approaches has always been a fundamental aspect of the PATAT mission in bridging the gap between practitioners and researchers in this increasingly important field.

An addition to the PATAT Conference this year is the inclusion of a number of key note addresses from practitioners. The conference organisers believe that this is an important initiative in addressing the well recognised gap which exists between the practice and theory of automated timetabling. The idea is that the practitioners stream should integrate with the conference theory sessions in an attempt to bring both practitioners and theoreticians together. It is intended that this will be a springboard which will help future PATAT conferences to continue to integrate and combine both the research and practice agendas across all areas of timetabling.

The programme of this year's conference features 73 presentations which represent the state-of-the-art in automated timetabling: there are 4 plenary papers, 31 full papers, 28 extended abstracts, 2 system demonstrations and 8 key note practitioner talks. It is encouraging to see the number of submissions which are orientated towards timetabling systems which draw upon leading edge approaches. As was the case in Montreal in 2008, a post-conference volume of selected and revised papers is to be published in Annals or Operational Research. Authors of full papers and extended abstracts are encouraged to submit to this special issue after the conference.

We would like to express our gratitude to the large number of individuals who have helped organise the conference. We thank the members of the Steering Committee who continue to ensure the ongoing success of the series and the members of the Programme Committee who have worked hard to referee the conference submissions. As always we are grateful to all authors and delegates. We would particularly like to thank Dr Pat Corr, Director of the INTO Centre at Queen's University for hosting the conference. We hope you will agree that the surroundings lend themselves very well to the running of an intimate and successful conference. Special thanks go to the organising committee especially Evelyn Milliken and Brian Fleming for their organisational skills and tireless help

and support in ensuring that the conference runs to the highest possible standard. Finally we would like to thank our sponsors who not only have helped fund the conference but are also all making a valuable contribution in terms of presentations.

We are delighted to welcome you all to the Queens University of Belfast. We hope you enjoy the conference talks and networking opportunities provided. As another first for the conference, it is our intention to survey all participants after the conference to learn how we can continue to improve on the progress made by the series of conferences to date.

Barry McCollum and Edmund Burke

PATAT 2010 Conference Program Committee

Abdullah, Salwani
Alfares, Hesham
Bardadym, Viktor
Bean, James
Brucker, Peter
Burke, Edmund
Cowling, Peter
De Causmaecker, Patrick
Dowsland, Kathryn
Erben, Wilhelm
Di Gaspero, Luca
Gendreau, Michel
Hertz, Alain
Kendall, Graham
Kingston, Jeffrey
Kwan, Raymond
Lewis, Rhyd
Meisels, Amnon
McMullan, Paul
Murray, Keith
Ozcan, Ender
Paechter, Ben
Parkes, Andrew
Pesant, Gilles
Petrovic, Sanja
Potvin, Jean-Yves
Qu, Rong
Rousseau, Louis-Martin
Ribeiro, Celso C.
Rudova, Hana
Schaerf, Andrea
Schreuder, Jan
Thompson, Jonathan
Toth, Paolo
Trick, Michael
Van Hentenryck, Pascal
Vanden Berghe, Greet
Voss, Stefan
De Werra, Dominique
White, George
Wright, Michael
Yellen, Jay

PATAT Steering Committee

Table of Contents

Plenary Papers

Full Papers

Extended Abstracts

System Demonstrations

Plenary Presentations

Scheduling English Football Fixtures: Consideration of Two Conflicting Objectives

Graham Kendall · Barry McCollum · Frederico Cruz · Paul McMullan

Abstract In previous work the distance travelled by UK football clubs, and their supporters, over the Christmas/New Year period was minimised. This is important as it is not only a holiday season but, often, there is bad weather at this time of the year. Whilst searching for good quality solutions for this problem, various constraints have to be respected. One of these relates to *clashes*, which measures how many *paired* teams play at home on the same day. Whilst the supporters have an interest in minimising the distance they travel, the police also have an interest in having as few pair clashes as possible. This is due to the fact that these fixtures are more expensive, and difficult, to police. However, these two objectives (minimise distance and minimise pair clashes) conflict with one another in that a decrease in one intuitively leads to an increase in the other. This paper explores this question and shows that there are compromise solutions which allow fewer pair clashes but does not statistically increase the distance travelled. This paper provides a more comprehensive study of the initial results presented at the previous PATAT conference. We present a more detailed set of computational experiments, along with a greater number of datasets. We conclude that it is sometimes possible to reduce the number of pair clashes whilst not significantly increasing the overall distance that is travelled.

Keywords Sport · Football · Scheduling · Multiobjective

Graham Kendall
School of Computer Science, University of Nottingham, NG8 1BB, UK
Tel.: +44 (0) 115 846 6514
Fax: +44 (0) 115 951 4254
E-mail: gxk@cs.nott.ac.uk

Barry McCollum
School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT7 1NN, UK

Frederico Cruz
Departamento de Estatstica - ICEx - UFMG, Av. Antnio Carlos, 6627, 31270-901 - Belo Horizonte - MG, Brazil

Paul McMullan
School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, BT7 1NN, UK

1 Introduction

The English Premier League is one of the most high profile, and successful, football (soccer in the USA) leagues in the world. It comprises 20 teams which have to play each other both home and away (i.e. a double round robin tournament), resulting in 380 fixtures that have to be scheduled. The other three main divisions in England (the Championship, League One and League Two) each have 24 teams, resulting in 552 fixtures having to be scheduled for each division. Therefore, for the four main divisions in England 2036 fixtures have to be scheduled every season. The divisions operate a system of promotion and relegation such that the teams in each division changes each year so it is not possible to simply use the same schedule every time.

Of particular interest are the schedules that need to be generated for the Christmas/New Year period. At this time of the year it is a requirement that every team plays two fixtures, one on Boxing Day (26th December) and one on New Years Day (1st January). Whilst scheduling these two sets of fixtures the overriding aim is to minimise the total distance that has to be travelled by the supporters. An analysis of the fixtures that were actually used, and also following discussions with the football authorities, confirm that this is a real world requirement and that the distances travelled by the supporters are the minimum when compared against other fixtures when all teams play. In addition, there are various other constraints that have to be respected, which are described in sections 3 and 4.

The problem we tackle in this paper is to attempt to minimise two competing objectives to ascertain if there is a good trade off between them. The objectives we minimise are the distances travelled by the supporters and the number of *pair clashes*. Pairing matches two (or more) teams and dictates that these clubs should not play at home on the same day. If they do, this is termed a pair clash. In fact, a certain number of pair clashes are allowed. The exact number is taken from the number that were present in the published fixtures for a given season. Importantly, paired teams cannot play each other on the two days in question. This is treated as a hard constraint. It is this constraint that causes a problem. If we allow Liverpool and Everton (for example) to play each other, one set of supporters would only travel four miles. If these teams are paired (as they are) then they cannot play each other so the distances are likely to increase as either Liverpool or Everton would have to travel more than four miles. As pair clashes usually involve teams which are geographically close this gives rise to the conflicting objectives.

In [19], an initial study of the problem considered the 2003-2004 football season, suggesting that it may be possible to minimise both of these competing objectives but still produce results which are acceptable to both the supporters (who are interested in minimising the amount they travel) and the police (who are interested in having fewer pair clashes). In this paper, we carry out a more in depth study by considering more seasons and carrying out statistical analysis of the results in order to draw stronger conclusions.

2 Related Work

Producing a double round robin tournament is relatively easy in that the algorithms are well known, with the polygon construction method being amongst the most popular [9]. The problem with utlising such an algorithm is that the fixtures it generates,

although being a valid round robin tournament, will not adhere to all the additional constraints for a particular problem. Moreover, every problem instance will be subtly different and, often, a bespoke algorithm is required for each instance. This is even the case when faced with seemingly the same problem. For example, the English Football League consists of four divisions and 92 teams. It would be easy to assume that once an algorithm has been developed it can be used every season. This may indeed be the case but due to the promotion/relegation system the problem changes year on year and, perhaps, there are additional features/constraints in one season that were not previously present. Rasmussen and Trick [21] provide an excellent overview of the issues, methods and theoretical results for scheduling round robin tournaments.

The Travelling Tournament Problem (TTP) [11] is probably the most widely used test bed in sports scheduling. The problem was inspired by work carried out for Major League Baseball [11]. The aim of the TTP is to generate a double round robin tournament, while minimising the overall distance travelled by all teams. Unlike the problem studied in this paper, it is possible to minmise the overall travel distance as teams go on *road trips* so, with a suitable schedule, the length of these trips can be reduced. The TTP is further complicated by the introduction of two constraints. The first says that no team can play more than three consecutive home or away games. The second stipulates that if team i plays team j in round, r, then team j cannot play team i in round $r+1$. These constraints add sufficient complexity to the problem so as to make it challenging, but it still does not reflect all the constraints that are present in the real world problem.

The TTP has received significant research attention. Some of the important papers being [12, 2, 8, 22, 25]. A recent annotated bibliography of TTP papers can be found in [18]. An up to date list of the best known solutions, as well as details of all the instances, can be found at the web site maintained by Michael Trick [23].

With respect to minimising travel costs/distances, previous studies have considered a variety of sports. Campbell and Chen [6] and Ball and Webster [3] both studied basketball, attempting to minimise the distance travelled. Bean and Birge [4] also studied basketball, attempting to minimise airline travel costs. Minimising travel costs was also the focus of [5], for baseball. Minimising travel distances for hockey [16] and umpires for baseball [15] have also been studied. Wright [28], as one part of the evaluation function, considered travel between fixtures for English cricket clubs. Costa [7] considered the National Hockey League, where minimisation of the distance travelled by the teams was just one factor in the objective function.

Urrutia and Ribeiro [24] have shown that minimising distance and maximising breaks (two consecutive home games (home break) or two consecutive away games (away break)) is equivalent. This followed previous work by de Werra [26, 27] and Elf et al. [14] who showed how to construct schedules with the minimum number of breaks.

The scheduling problem that we are considering in this paper is minimising the distance travelled for two complete fixtures (a complete fixture is defined as a set of fixtures when every team plays) while, at the same time, minimising the number of pair clashes. These two complete fixtures can then be used over the Christmas holiday period when, for a variety of reasons, teams wish to limit the amount of travelling undertaken. Note, that this is a different problem to the Travelling Tournament Problem as the TTP assumes that teams go on road trips, and so the total distance travelled over a season can be minimised. In English football, there is no concept of road trips. Therefore, over the course of a season, the distance cannot be minimised. However, we can minimise the distance on particular days. Kendall [17] adopted a two-phase approach to produce two

complete fixtures for this problem. A depth first search was used to produce fixtures for one day, for each division. A further depth first search created another set of fixtures for the second day. This process produced eight separate fixtures (two sets of fixtures for each division) which adhered to some of the constraints (e.g. a team plays at home on one day and away on the other) but had not yet addressed the constraints with regards to pair clashes (see [17] for a detailed description). The fixture lists from the depth first searches were input to a local search procedure which aimed to satisfy the remaining constraints, whilst attempting to minimise the overall distance travelled. The output of the local search, and a post-process operation to ensure feasibility, produced the results presented in the paper.

Overviews of sports scheduling can be found in [13, 9, 10, 21, 29, 20, 18].

3 Problem Definition

In previous work [17] the only objective was to minimise the total distance travelled by the teams/supporters. The aim of that study was to investigate if we were able to generate better quality solutions than those used by the football league. We demonstrated that it was possible. As stated in the Introduction, the police also have an interest in the fixtures that are played at this time of the year. If we are able to generate acceptable schedules, with fewer pair clashes then the policing costs would be reduced.

The purpose of this paper is to investigate if there is an acceptable trade off between the minimisation of distance and the minimisation of pair clashes. In order to do this we will utilise a multi-objective methodology.

4 Experimental Setup

We use a two stage algorithm. In [17] a depth first search (DFS) was used, followed by a local search. DFS was used as we wanted to carry out a preliminary study just to see if this area was worthy of further study. As we were able to produce superior solutions to the published fixtures we have now decided to utilise more sophisticated methods, due to the large execution times of DFS which were typically a few hours for each division. In this work we utilise CPLEX as a replacement for DFS and simulated annealing [1] as a replacement for the local search. This reduces the overall execution time from tens of hours to a few minutes.

4.1 Phase 1: CPLEX

The first phase uses CPLEX to produce an optimal solution to a relaxed version of the problem. In generating *relaxed* optimal solutions we respect the following constraints, whilst minimizing the overall distance.

1. Each of the 92 teams has to play on two separate days (i.e. 46 fixtures will be scheduled on each day).
2. Each team has to play at home on one day and away on the other.
3. Teams are not allowed to play each other on both days.
4. A team is not allowed to play itself.

The CPLEX model is executed four times. Each run returns the Boxing Day and New Years Day fixtures for a particular division. Each run takes less than 10 seconds.

In solving the CPLEX model we do not take into account many of the constraints that ultimately have to be respected. For example, pair clashes, geographical constraints such as the number of London or Manchester clubs playing at home on the same day etc. (see [17] for details).

4.2 Phase 2: Simulated Annealing

The schedules from CPLEX are input to the second phase, where we utilise simulated annealing. This operates across all the divisions in order to resolve any hard constraint violations whilst still attempting to minimise the distance.

The simulated annealing parameters are as follows:

Start_Temperature = 1000 The same value is used across all seven datasets and was found by experimentation. We could have used different values for each dataset but we felt that it was beneficial to be consistent across all the datasets.
Stop_Temperature The algorithm continues while the temperature is > 0.1.
Cooling Schedule $CurTemp = CurTemp$ * 0.95.
Number of Iterations 2000 iterations are carried out at each temperature.

4.3 Evaluation Function

The evaluation function we use for simulated annealing is dynamic in that the hard constraint violations are more heavily penalised as the search progresses. This enables more exploration at the start of the search, which gets tighter as the temperature is reduced. The objective function is formulated as follows:

$$f(x) = d\_fb + d\_fy + w \times penalty \tag{1}$$

where:

$d\_fb =$ total distance travelled by teams on Boxing Day.
$d\_fy =$ total distance travelled by teams on New Years Day.
$w =$ is a weight for the penalty (see below). It is given by ($Start\_Temperature$ - $CurTemp$). $Start\_Temperature$ is the maximum temperature for the simulated annealing algorithm. $CurTemp$ is the current temperature of the simulated annealing algorithm. As the simulated annealing algorithm progresses, the weight of the penalty gradually increases, driving the search towards feasible solutions, but allowing it to search the infeasible region at the start of the search.
$penalty =$ This is given by a summation of the following terms (the limits referred to are available in [17] and represent the values found by analyzing the published fixtures):

ReverseFixtures The number of *reverse* fixtures (the same teams cannot meet on both days).
Boxing Day Local Derby Clashes The number of paired teams playing each other on Boxing Day.

New Years Day Local Derby Clashes The number of paired teams playing each other on New Years Day.

Boxing Day London Clashes The number of London clubs playing at home on Boxing Day, which exceed a given limit.

New Years Day London Clashes The number of London clubs playing at home on New Years Day, which exceed a given limit.

Boxing Day Greater Manchester Clashes The number of Greater Manchester based clubs playing at home on Boxing Day, which exceed a given limit.

New Years Day Greater Manchester Clashes The number of Greater Manchester based clubs playing at home on New Years Day, which exceed a given limit.

Boxing Day London Premier Clashes The number of Premiership London clubs playing at home on Boxing Day, which exceed a given limit.

New Years Day London Premier Clashes The number of Premiership London clubs playing at home on New Years Day, which exceed a given limit.

Boxing Day Clashes The number of Boxing Day clashes greater than an allowable limit.

New Years Day Clashes The number of New Years Day clashes greater than an allowable limit.

4.4 Perturbation Operators

Simulated annealing often has a single neighborhood operator but we have defined sixteen operators in order to match the hard constraints within the model. The operators are as follows:

1. Examines the Boxing Day fixtures and if the number of clashes exceeds an upper limit, randomly select one of the clashing fixtures and swap the home and away teams.
2. Same as 1 expect that it considers New Years Day fixtures.
3. Examines the Boxing Day fixtures and if the number of London based clubs exceeds an upper limit, randomly select one of the fixtures that has a London based club playing at home and swap the home and away teams.
4. Same as 3 except that it considers Greater Manchester based clubs.
5. Same as 3 except that it considers London based premiership clubs.
6. Same as 3 except that it considers the New Years Day fixtures.
7. Same as 4 except that it considers the New Years Day fixtures.
8. Same as 5 except that it considers the New Years Day fixtures.
9. Examines the Boxing Day and New Years Day fixture lists, returning the number of reverse fixtures (where team i plays team j and team j plays team i). While there are reverse fixtures, one of the reverse fixtures on Boxing Day is chosen and the home team is swapped with a randomly selected home team, with the condition that the swaps must be made between teams in the same division. This operator iterates until all reverse fixtures have been removed from the fixture list.
10. Same as 9 except the swaps are made in the New Years Day fixtures.
11. This operator examines the Boxing Day and New Years Day fixture lists, returning the number fixtures where paired teams are playing each other. While this is the case, one of the Boxing Day fixtures is chosen and the home team is swapped with a randomly selected home team in the Boxing Day fixtures, with the condition that

the swaps must be made between teams in the same division. This operator iterates until all local pair clashes have been removed from the fixture lists.
12. Same as 11 except the swaps are made in the New Years Day fixtures.
13. This operator chooses a random fixture from a candidate list (we use a candidate list size of 250) which represents the potential fixtures that have the shortest distances. Swaps are carried out in the Boxing Day fixtures in order to allow the two teams from the selected item in the candidate list to play each other. The necessary swaps are also done in the New Years Day fixture to ensure feasibility.
14. Same as 13 except that it considers the New Years Day fixtures.
15. Selects a random fixture in the Boxing Day fixture list and swaps the home and away teams.
16. Same as 15, but swaps a random fixture in the New Years Day fixture list.

At each iteration, one of the sixteen operators is chosen at random. $Start\_Temperature$ is initially set to enable infeasible solutions during the early stages of the algorithm, but they are more heavily penalised at lower temperatures (eq. 1), ensuring that the final solution is feasible.

4.5 Experimental Methodology

We are investigating this problem from a multi-objective perspective but rather than using a multi-objective algorithm we run the same algorithm a number of times, adjusting the parameters for each run. As an example, for the 2002-2003 season the number of pair clashes, in the published fixtures, was 10 and 8 for Boxing Day and New Years Day respectively. We denote this as 10-8 in the tables below. Therefore, the first experiment fixes the values as 10 and 8 as the number of pair clashes that cannot be exceeded. In this respect, these values represent hard constraints. The next experiment reduces one of these values so that the next experiment uses 10-6. We then reduce the other value to run a further experiment using 8-8. There are two points worthy of note. Firstly, we reduce the value by two as a pair clash of, say, Everton and Liverpool actually counts as two pair clashes as both teams are considered to be clashing. Secondly, we do not reduce the total number of pair clashes below 16.

5 Results

Tables 1 thru 7 shows the results of each of the seven seasons that we use. The *Clashes* column shows the number of pair clashes (see section 4.5 for the notation that we use). Min represents the best solution found. Max is worst solution found and *Average* and *Std Dev* are self-explanatory. All experiments were runs 30 times.

Table 1 2002-2003: Summary of results from 30 runs

Clashes	Min	Max	Average	Std Dev
10-8	5243	6786	5630	288.46
10-6	5674	7222	6183	410.71
8-8	5562	6797	6070	309.50

Table 2 2003-2004: Summary of results from 30 runs

Clashes	Min	Max	Average	Std Dev
8-14	5464	6173	5698	165.46
8-12	5412	6519	5827	228.66
8-10	5511	7093	6053	417.00
8-8	5887	7674	6535	433.83
6-14	5550	6334	5805	176.02
6-12	5559	6587	6036	289.75
6-10	5898	7416	6454	395.37
4-14	5592	6911	6059	274.61
4-12	5886	7848	6635	484.59
2-14	6028	7704	6704	448.87

Table 3 2004-2005: Summary of results from 30 runs

Clashes	Min	Max	Average	Std Dev
10-10	5365	6986	5644	318.33
10-8	5345	6348	5727	259.17
10-6	5812	7714	6431	421.63
8-10	5443	6982	5923	469.01
8-8	5645	7612	6428	550.67
6-10	5810	7824	6486	487.26

Table 4 2005-2006: Summary of results from 30 runs

Clashes	Min	Max	Average	Std Dev
12-14	5234	6046	5575	184.74
12-12	5335	6002	5596	153.90
12-10	5240	6511	5641	238.58
12-8	5334	6423	5754	231.81
12-6	5481	6958	6010	339.63
12-4	6041	6989	6468	271.99
10-14	5171	6683	5606	304.33
10-12	5308	6322	5610	204.96
10-10	5460	6674	5846	359.65
10-8	5595	6380	5872	216.82
10-6	6027	7561	6660	421.25
8-14	5335	6674	5680	286.00
8-12	5334	6133	5722	211.02
8-10	5608	7078	5979	356.15
8-8	6146	7277	6587	302.48
6-14	5500	6694	5843	254.23
6-12	5528	6655	5951	233.54
6-10	5884	7291	6529	382.80
4-14	5713	7391	6161	331.25
4-12	6032	7904	6662	434.72
2-14	6084	7551	6682	399.34

In tables 8 and 9 we analyse the results from table 1. Table 8 shows the results of independent two-tailed t-tests (at the 95% confidence level) to compare the means of each experiment against every other experiment for that season. Where two experiments are statistically significant the relevant cell shows "Yes", otherwise the cell is

Table 5 2006-2007: Summary of results from 30 runs

Clashes	Min	Max	Average	Std Dev
14-8	5713	7040	6077	300.71
14-6	5735	7065	6117	270.59
14-4	5872	7000	6259	227.84
14-2	6110	7778	6741	402.35
12-8	5721	6784	6084	244.28
12-6	5714	6894	6234	326.99
12-4	6195	7546	6791	405.86
10-8	5762	7671	6209	411.02
10-6	5894	7376	6618	423.94
8-8	6071	6958	6513	251.33

Table 6 2007-2008: Summary of results from 30 runs

Clashes	Min	Max	Average	Std Dev
14-10	5366	5902	5595	145.26
14-8	5403	5975	5674	152.93
14-6	5425	7172	5870	372.17
14-4	5690	6995	6172	364.78
14-2	5905	7856	6698	435.98
12-10	5370	6506	5736	294.88
12-8	5321	7139	5850	338.15
12-6	5625	7394	6084	365.93
12-4	5961	7580	6575	411.41
10-10	5340	6552	5754	228.71
10-8	5616	6365	5944	183.52
10-6	6101	7468	6619	369.10
8-10	5536	7081	6056	369.47
8-8	6091	7884	6725	402.08
6-10	5951	7709	6647	381.12

Table 7 2008-2009: Summary of results from 30 runs

Clashes	Min	Max	Average	Std Dev
10-10	5564	6806	5833	246.11
10-8	5574	6235	5829	140.52
10-6	5736	6523	6106	208.78
8-10	5581	6817	5936	281.83
8-8	5790	6900	6148	230.42
6-10	5809	7194	6208	274.67

empty. As an example, if we compare 10-8 (column) with 10-6 (row) in table 8 we see that the means (i.e. the travel distances from 30 independent runs) are statistically different. By comparing the means in table 1, 5630 and 6183 respectively, we conclude that reducing the number of pair clashes from 18 (10-8) to 16 (8-8) the travel distances for the clubs/supporters increases by a significant amount. Looking at 10-6 and 8-8, there is no statistical difference. However, as both of these experiments represent 16 pair clashes it is, perhaps, not surprising that the average distance travelled over the 30 runs is (statistically) the same.

Table 9 summarises the results from table 8 by only showing those experiments where there are statistical differences, AND when the total number of pair clashes is different (i.e. it will ignore 10-6 and 8-8). We can see from table 9 that there are no experiments where we can reduce the number of pair clashes that leads to no statistical difference in the distance travelled.

Tables 10 and 11 show similar analysis for the 2003-204 season. Again, it is not possible to reduce the number of pair clashes without an (statistically) increase in the distance travelled.

Tables 12 and 13 are more interesting. Table 12 shows that there is no statistical difference between the 10-10 (20 pair clashes) experiment and the 10-8 (18 pair clashes) experiment. Removing all the *noise* from the table (see table 13) we can see that it is possible to reduce the number of pair clashes from 20 to 18 without a significant rise in the distance travelled (the respective means from table 3 are 5644 and 5727).

For the remaining four seasons, we only present the summary tables. Where a"Yes" appears in these tables (tables 14 thru 17) it indicates that it is possible to reduce the number of pair clashes and not have an (statistical) increase in travel distance. The tables show that there are a number of opportunities to reduce policing costs. We are probably most interested in the top rows as they represent the fixtures that were actually used.

Table 8 2002-2003: Are the Results Statistically Different?

Clashes	10-8	10-6	8-8
10-8	X	Yes	Yes
10-6		X	
8-8			X

Table 9 2002-2003: Are different total clashes significantly different?

Clashes	10-8	10-6	8-8
10-8	X		
10-6		X	
8-8			X

Table 10 2003-2004: Are the Results Statistically Different?

Clashes	8-14	8-12	8-10	8-8	6-14	6-12	6-10	4-14	4-12	2-14
8-14	X	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
8-12		X	Yes	Yes		Yes	Yes	Yes	Yes	Yes
8-10			X	Yes	Yes		Yes		Yes	Yes
8-8				X	Yes	Yes		Yes		
6-14					X	Yes	Yes	Yes	Yes	Yes
6-12						X	Yes		Yes	Yes
6-10							X	Yes		Yes
4-14								X	Yes	Yes
4-12									X	
2-14										X

Table 11 2003-2004: Are different total clashes significantly different?

Clashes	8-14	8-12	8-10	8-8	6-14	6-12	6-10	4-14	4-12	2-14
8-14	X									
8-12		X								
8-10			X							
8-8				X						
6-14					X					
6-12						X				
6-10							X			
4-14								X		
4-12									X	
2-14										X

Table 12 2004-2005: Are the Results Statistically Different?

Clashes	10-10	10-8	10-6	8-10	8-8	6-10
10-10	X		Yes	Yes	Yes	Yes
10-8		X	Yes		Yes	Yes
10-6			X	Yes		
8-10				X	Yes	Yes
8-8					X	
6-10						X

Table 13 2004-2005: Are different total clashes significantly different?

Clashes	10-10	10-8	10-6	8-10	8-8	6-10
10-10	X	Yes				
10-8		X				
10-6			X			
8-10				X		
8-8					X	
6-10						X

Table 14 2005-2006: Are different total clashes significantly different?

Clashes	12-14	12-12	12-10	12-8	12-6	12-4	10-14	10-12	10-10	10-8	10-6	8-14	8-12	8-10	8-8	6-14	6-12	6-10	4-14	4-12	2-14
12-14	X	Yes	Yes				Yes	Yes				Yes									
12-12		X	Yes					Yes				Yes									
12-10			X	Yes			Yes						Yes								
12-8				X								Yes									
12-6					X				Yes												
12-4						X															
10-14							X	Yes				Yes	Yes								
10-12								X													
10-10									X	Yes		Yes		Yes			Yes				
10-8										X						Yes					
10-6											X										
8-14												X	Yes								
8-12													X								
8-10														X		Yes					
8-8															X						
6-14																X	Yes				
6-12																	X				
6-10																		X			
4-14																			X		
4-12																				X	
2-14																					X

Table 15 2006-2007: Are different total clashes significantly different?

Clashes	14-8	14-6	14-4	14-2	12-8	12-6	12-4	10-8	10-6	8-8
14-8	X	Yes			Yes	Yes		Yes		
14-6		X				Yes		Yes		
14-4			X							
14-2				X						
12-8					X			Yes		
12-6						X				
12-4							X			
10-8								X		
10-6									X	
8-8										X

Table 16 2007-2008: Are different total clashes significantly different?

Clashes	14-10	14-8	14-6	14-4	14-2	12-10	12-8	12-6	12-4	10-10	10-8	10-6	8-10	8-8	6-10
14-10	X														
14-8		X								Yes					
14-6			X			Yes					Yes		Yes		
14-4				X											
14-2					X										
12-10						X	Yes			Yes					
12-8							X				Yes				
12-6								X							
12-4									X						
10-10										X					
10-8											X				
10-6												X			
8-10													X		
8-8														X	
6-10															X

Table 17 2008-2009: Are different total clashes significantly different?

Clashes	10-10	10-8	10-6	8-10	8-8	6-10
10-10	X	Yes		Yes		
10-8		X				
10-6			X			
8-10				X		
8-8					X	
6-10						

6 Conclusion

We have demonstrated that it is sometimes possible to reduce the number of pair clashes without a statistical difference to the distance that has to be travelled by the

club/supporters. This provides the police with the ability to reduce their costs for these two days, which might have included paying overtime. We hope that we are able to discuss these results with the football authorities and the police in order for them to validate our work and to provide us with potential future research directions. We already recognise that some pair clashes might provide the police with more *problems* than others and it might be worth prioritising certain clashes so that these can be removed, rather than removing less high profile fixtures. As a longer term research aim, we would like to include in our model details about public transport as some routes might be more difficult than other routes, even if they are shorter. We also plan to run our algorithms for every future season, as well as for previous seasons. Executing the algorithm is not the main issue. Data collection provides the real challenge due to the distance data that has to be collected. To date, this has been carried out manually by using motoring organisation's web sites but we have recently started experimenting with services such as Google Maps^{TM}and Multimap which will speed up the data collection.

References

1. Aarts, E., Korst, J., Michels, W.: Simulated annealing. In: E.K. Burke, G. Kendall (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Methodologies, 1st edn., chap. 7, pp. 97–125. Springer (2005)
2. Anagnostopoulos, A., Michel, L., Van Hentenryck, P., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. Journal of Scheduling **9**, 177–193 (2006)
3. Ball, B.C., Webster, D.B.: Optimal scheduling for even-numbered team athletic conferences. AIIE Transactions **9**, 161–169 (1977)
4. Bean, J.C., Birge, J.R.: Reducing travelling costs and player fatigue in the national basketball association. Interfaces **10**, 98–102 (1980)
5. Cain, W.O.: The computer-aided heuristic approach used to schedule the major league baseball clubs. In: S.P. Ladany, R.E. Machol (eds.) Optimal Strategies in Sports, pp. 33–41. North Holland, Amsterdam (1977)
6. Campbell, R.T., Chen, D.S.: A minimum distance basketball scheduling problem. In: R.E. Machol, S.P. Ladany, D.G. Morrison (eds.) Management Science in Sports, *Studies in the Management Sciences*, vol. 4, pp. 15–25. North-Holland, Amsterdam (1976)
7. Costa, D.: An evolutionary tabu search algorithm and the NHL scheduling problem. INFOR **33**, 161–178 (1995)
8. Di Gaspero, L., Schaerf, A.: A composite-neighborhood tabu search approach to the traveling tournament problem. Journal of Heuristics **13**, 189–207 (2007)
9. Dinitz, J.H., Fronček, D., Lamken, E.R., Wallis, W.D.: Scheduling a tournament. In: C.J. Colbourn, J.H. Dinitz (eds.) Handbook of Combinatorial Designs, 2nd edn., pp. 591–606. CRC Press (2006)
10. Drexl, A., Knust, S.: Sports league scheduling: Graph- and resource-based models. Omega **35**, 465–471 (2007)
11. Easton, K., Nemhauser, G.L., Trick, M.A.: The travelling tournament problem: Description and benchmarks. In: T. Walsh (ed.) Principles and Practice of Constraint Programming, *Lecture Notes in Computer Science*, vol. 2239, pp. 580–585. Springer (2001)
12. Easton, K., Nemhauser, G.L., Trick, M.A.: Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In: E. Burke, P. de Causmaecker (eds.) The 4th International Conference on the Practice and Theory of Automated Timetabling, *Lecture Notes in Computer Science*, vol. 2740, pp. 100–109. Springer (2003)
13. Easton, K., Nemhauser, G.L., Trick, M.A.: Sports scheduling. In: J.T. Leung (ed.) Handbook of Scheduling, pp. 52.1–52.19. CRC Press (2004)
14. Elf, M., Jnger, M., Rinaldi, G.: Minimizing breaks by maximizing cuts. Operations Research Letters **31**(3), 343–349 (2003)
15. Evans, J.R.: A microcomputer-based decision support system for scheduling umpires in the American Baseball League. Interfaces **18**, 42–51 (1988)

16. Ferland, J.A., Fleurent, C.: Computer aided scheduling for a sport league. INFOR **29**, 14–25 (1991)
17. Kendall, G.: Scheduling English football fixtures over holiday periods. Journal of the Operational Research Society **59**, 743–755 (2008)
18. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography. Computers & Operations Research **37**, 1–19 (2010)
19. Kendall, G., While, L., McCollum, B., Cruz, F.: A multiobjective approach for UK football scheduling. In: E.K. Burke, M. Gendreau (eds.) Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (2008)
20. Knust, S.: Classification of literature on sports scheduling (2010). Available online at http://www.inf.uos.de/knust/sportssched/sportlit_class/, last visited 15th July 2010
21. Rasmussen, R.V., Trick, M.A.: Round robin scheduling – A survey. European Journal of Operational Research **188**, 617–636 (2008)
22. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research **179**, 775–787 (2007)
23. Trick, M.: Traveling tournament problem instances (2010). Available online at http://mat.gsia.cmu.edu/TOURN/, last accessed 15th July 2010
24. Urrutia, S., Ribeiro, C.: Minimizing travels by maximizing breaks in round robin tournament schedules. Electronic Notes in Discrete Mathematics **18-C**, 227–233 (2004)
25. Urrutia, S., Ribeiro, C.C., Melo, R.A.: A new lower bound to the traveling tournament problem. In: Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling, pp. 15–18. IEEE, Honolulu (2007)
26. de Werra, D.: Scheduling in sports. In: P. Hansen (ed.) Studies on Graphs and Discrete Programming, pp. 381–395. North Holland, Amsterdam (1981)
27. de Werra, D.: Some models of graphs for scheduling sports competitions. Discrete Applied Mathematics **21**, 47–65 (1988)
28. Wright, M.: Timetabling county cricket fixtures using a form of tabu search. Journal of the Operational Research Society **45**, 758–770 (1994)
29. Wright, M.: 50 years of OR in sport. Journal of the Operational Research Society **60**, S161–S168 (2009)

Estimating the limiting value of optimality for very large NP problems

George M. White

Abstract The search for better solutions to large NP-hard problems such as timetabling, personnel scheduling, resource allocation, etc., often requires approximation methods. These methods can often yield solutions that are often "very good", although it is generally impossible to say just how good these solutions are. Not only do we not know what the best solutions are, we also don't know how far away we are from the optimum solution - we may be very close but we may also be quite far. This paper describes a method of using historical data to estimate the limiting optimality of the solution to a problem if the problem arises from a situation taken from the real world and there is sufficient historical data about proposed solutions. Knowledge of the limiting optimality can provide guidance in estimating just how far a "very good" solution lies from the best solution, even when we don't know what the best solution is.

Keywords NP-hard problems · examination scheduling · penalty estimation

1 Introduction

If you are reading this text, you are likely very familiar with the difficulties of solving large scale problems. These are problems that are commonly known by names such as the travelling salesman problem, the graph (or vertex) colouring problem, examination scheduling, staff scheduling and the like.

Our challenge is nearly always to find a solution to these problems that is in some sense "the best", a concept that is more easily stated than defined. Most attempts to define just what is meant by "the best" are very context-sensitive but one that seems satisfactory in many circumstances is based upon the principle of utility proposed by Jeremy Bentham, that the right way to act is the way that causes "the greatest good for the greatest number of people". Thus the best solution is usually the one that

G.M. White
School of Information Technology and Engineering
University of Ottawa
Ottawa K1N 6N5 Canada
Tel.: +613-562-5800 x6677
Fax: +613-562-5664
E-mail: white@site.uottawa.ca

causes the least expense to travelling salesmen, uses the fewest colours to colour a map, minimizes the misery of exam writing students, or minimizes the complaints from staff when the teaching schedule, nursing schedule, employee roster ,..., is published.

When such problems are solved on our computers, this principle is used to formulate a mathematical expression strongly dependent on the exact nature of the problem, as is the method used to extract it. Our interest here is focussed on a class of problems known as non-deterministric, polynomial time hard problems (NP-hard), sometimes defined informally as those problems as hard as the hardest problems in NP. A much more detailed and precise definition and discussion is found in the classic book (Garey and Johnson (1979)).

Dispite their complexity some NP-hard problems have been solved to completeness. The travelling salesman problem, TSP, has been studied since at least 1832. Provably optimal solutions to this problem can be obtained by a variety of techniques entailing a prodidgeous amounts of computer time.

Examination scheduling problems have been in existance ever since there were examinations. Casting these schedules has evolved from a chore to be done by hand with pencils, paper and large erasers to a programming exercise to be solved by computer.

Sports scheduling has resisted computerization for a long time but is now slowly and with some reluctance being increasingly done by machine (Easton et al (2003)).

A discussion of the possibility of predicting future results based on the analysis of past results appears in the next section along with two examples of its potential in elite speed sports. The logistic equation is a potential candidate for obtaining this prediction and appears in section 3. The TSP and examination scheduling are developed in sections 4 and 5. Conclusions are found in section 6.

2 Is prediction possible?

Investigators working on the difficult problems discussed earlier continue to make progress year after year because of improvements to both hardware and software:

- processor speed
- algorithm used
- initial conditions
- parameter tuning
- manpower available

Similar conditions are true in elite sporting events. The goal is to produce a superior outcome and to achieve this goal choices must be made from a wide number of variables such as training, genetics, health, equipment, weather conditions, altitude, environment, diet, etc.

One example that can be cited is the men's 100 metre sprint. Whoever is the current record holder is often referred to as "The World's Fastest Man". A plot of the progressive world record in this event and the year in which it was achieved is shown in figure 1. The curved line is an attempt to fit the points to an equation and the dashed, straight line near the bottom of the graph is the asymptotic value of the curve.

A second example is the men's 5000 metre speed skate. The records for the fastest man on ice are shown in figure 2. For sporting events it is evident that although the times required to establish a new record are always being reduced, they will never be reduced to zero. In the case of sprinting, the dashed line represents an asymptotic

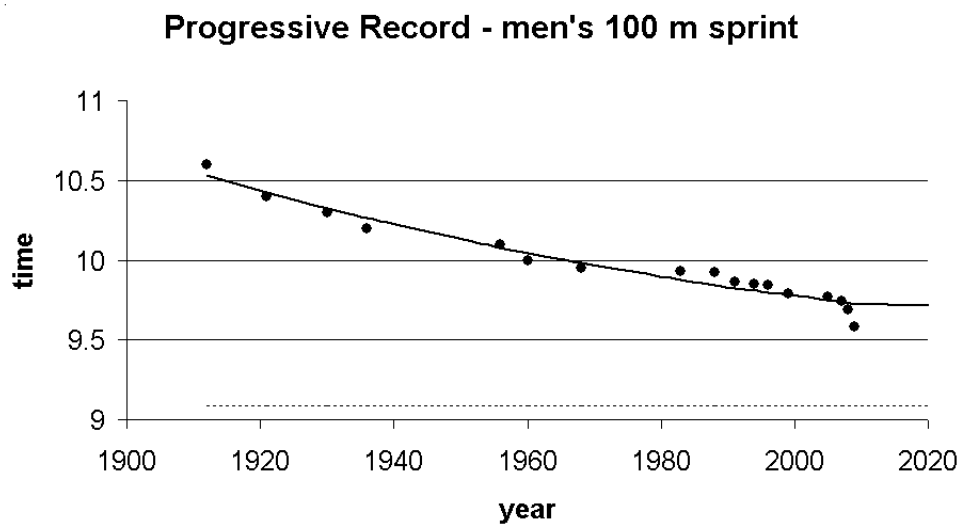


Fig. 1 Progressive record for men's 100 metre sprint

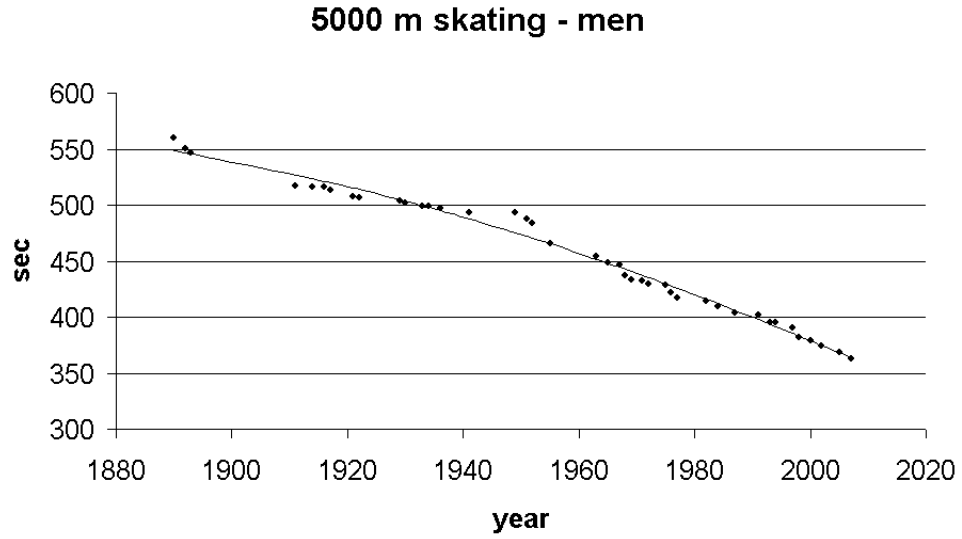


Fig. 2 Progressive record for men's 5000 metre skate

limiting value of the "ultimate" speed record and is suggested by the flattening of the curve at time goes on. For the skating event, the curve is steepening rather than flattening and suggests that there is some distance to go before the records converge on some limiting value. The analysis of curves such as this will be discussed in the next section.

This approach to records can analgously be applied to similar records in the area of large NP-hard problems touched on earlier. Perhaps a quantitative analysis can give some insight into the TSP problem and the examination problem. We may be able to predict future records, given a date, and perhaps also to estimate limiting values.

There is some evidence that other problems arising from the real world exhibit a similar behaviour.

3 The Logistic Curve

The shape of the curve that describes the experimental running best value leads to the conclusion that, at some time in the future, the best values will reach a limit. *i.e.*

$$\lim_{t\to\infty} \frac{dP(t)}{dt} = 0 \tag{1}$$

where $P(t)$ is the penalty obtained at time t. Since the exam scheduling problem is known to be NP-hard, the form of the derivative dP/dt is unknown, but it is reasonable to assume that it is some function of the current best penalty.

$$\frac{dP}{dt} = f(P) \tag{2}$$

Expanding this as a Maclaurin series yields

$$\frac{dP}{dt} = f(P) = a_0 + a_1P + a_2P^2 + a_3P^3 + ... \tag{3}$$

To simplify the form of the equation we might first try to approximate it as $dP/dt = a_0$. Then when P attains its limiting value, we have $dP/dt = 0$ and therefore $a_0 = 0$. This cannot possibly be the case. The next form to consider is $dP/dt = a_1P$ which equals 0 only if $P = 0$; this is probably not the case. The next simplest form is $\frac{dP}{dt} = a_1P + a_2P^2 = P(a_1 + a_2P)$ This has the desired properties. Recall that when P takes its limiting value, $dP/dt = 0$. It follows that $a_1 + a_2P_{limit} = 0$ or $P_{limit} = -a_1/a_2$.

In the literature, this equation often appears in the form

$$\frac{dP}{dt} = rP - \frac{rP^2}{k} = rP(1 - \frac{P}{k}) \tag{4}$$

This is a differential equation whose solution is

$$P(t) = \frac{kP_0e^{rt}}{k + P_0(e^{rt} - 1)} \tag{5}$$

where $P(t)$ is the penalty obtained at time t and r, k and P_0 are adjustable parameters. When $t = 0, P(t) = P(0) = P_0$. As $t \to \infty$, $P(t) \to k$.

A sketch of this equation in the form $P(rt)$ with k set equal to $5 * P_0$ and $r = 1$ is shown in figure 3. This equation is used to describe birth-death processes and race results among other applications. Here we will use it to analyse some published NP-hard results.

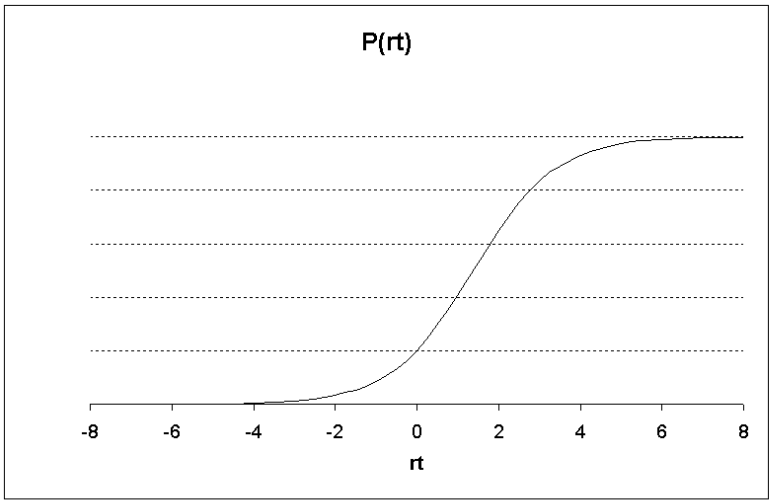


Fig. 3 Sketch of $P(rt)$ vs rt

4 The Travelling Salesman Problem (TSP)

The TSP is a real-world problem that has many practical applications and has therefore been intensively studied ever since it was introduced. The volumous history of the problem and its literature has been reviewed in several books (see for example the books by Lawler et al (1985); Applegate et al (2006)).

A book published in Germany in the 1830s described the problem in the context of actual travelling salemen who were wanting to cover their territory in the shortest possible time, but did no mathematical analysis. The first investigator to treat the problem in a mathematical setting was an Irishman, Sir William Rowan Hamilton, who studied the problem in the 1850s. Exact solutions to non-trivial problems were slow to appear because of the amount of calculation that had to be done but, by 1954, Danzig, Fulkerson and Johnson pubished the results for a 49 city instance, the capitals of the lower 48 United States plus Washington.

Since then the size of successfully solved TSP problems has grown steadily with the present record holder being D. Applegate and 6 colleagues, (Applegate et al (2006)) who solved a 85900 city instance in 2006. A graph of the sizes of solved instances and the year they were obtained is shown in figure 4.

The plot on the right shows the size of the successful instances vs. the year published (large points) and the best fitted curve of $P(t)$ to these points. Because of the large range of values, the same data has been plotted on a semi-log scale on the right. The smooth curve is a plot of equation 5 fitted by the Solver tool of Microsoft Excel to the data points. The fit is remarkable considering the passage of time between the first and the last points (about 36 years) and the variety of computers and software used to perform the calculations.

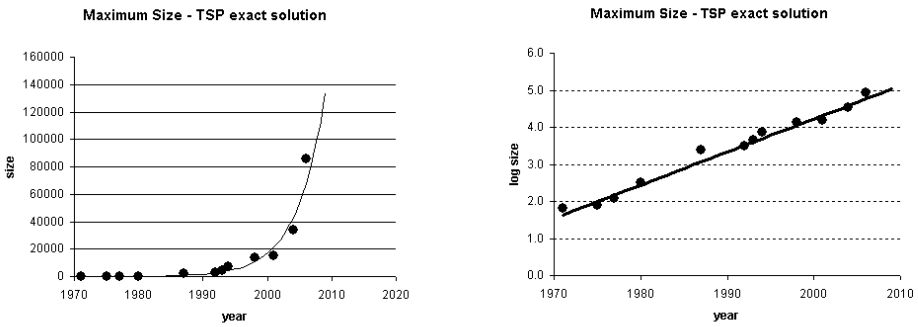


Fig. 4 Progressive record for TSP solved problem instances

5 The Examination Scheduling Problem

Examination scheduling is one of the earliest applications of computer technology to an academic problem. The possibility of finding new methods to study the problem, the promise of a useful application and the availability of real data from real sources, gave a strong impetus for academics to study the problem. Accordingly computer researchers started investigating this problem (see Broder (1964); Peck and Williams (1966)) and a few programs were written and used in practice (White and Chan (1979); Carter (1983)).

Researchers have employed many techniques in order to find better solutions. Discussions of methods and summaries of progress have been well treated in Qu et al (2009).

A *feasible* exam timetable is one in which no student is required to sit for more than one exam at a time. Although any feasible timetable will work, some of these timetables are worse than others. Several measures of the "badness" of a timetable have been proposed, such as

- the total number of consecutive exams a student must write
- the total number of consecutive exams plus the total number of exams separated by exactly one free timeslot

In 1996, a seminal paper (Carter et al (1996, 1997)) proposed a penalty, based on some earlier work, that is equal to the weighted sum of course pair penalties (Laporte and Desroches (1984)). Two exams taken by one student separated by n timeslots incurs a penalty p_n. The number of such penalties incurred by all the students is w_n. The penalty of the entire timetable is then defined to be

$$\sum_{i=1}^{5} p_i w_i \tag{6}$$

where $p_1 = 16, p_2 = 8, p_3 = 4, p_4 = 2, p_5 = 1$, and the summation is calculated over all students involved. The penalty so obtained is then divided by the number of students involved to get a *standard penalty*. The authors also referenced a depository of 13 data sets taken from real institutions that they used to test their algorithms. The benchmarks that resulted have been used ever since as a basis of comparison.

Some problems in the original data sets have been detected and corrected (see Qu et al (2009)).

Progress during this time has been made by many researchers who have employed many different approaches with a view to lowering the standard penalty when using new algorithms with the same data. The results are not unlike those obtained in track and field events where athletes attempt to lower the time required to cover a specified distance, say 100 metres, where basic conditions are unchanged. The outcomes of a foot race depend on a large number of variables: the individual racer, training, genetics, health, equipment, weather conditions, altitude, environment, diet, and the use of banned substances, among other things. The outcomes of experiments that cast examination timeables likewise depend on a large set of variables such as the algorithm used, initial conditions, parameter tuning, manpower available, processor speed, etc.

An examination of the recent timetabling literature shows that a wide variery of techniques has indeed been used and many researchers have published tables of their best results for the Toronto data base. A plot of the published values of the standard penalty against the year in which this result was published for the data set *yor-f-83* is shown in figure 5 (left). The running best penalty of a schedule for a given year is just

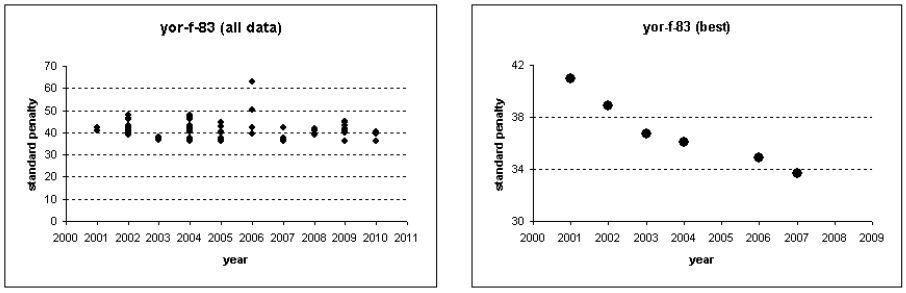


Fig. 5 Data points available for the data set *yor-f-83*

that value, obtained in that year, that had the lowest value. The set of running best penalties obtained yet for that data set at a given year is formed by choosing only those values that are better than any preceding lowest value. A plot of these best values is shown in figure 1 (right). Note that the axes of this graph have been rescaled.

Most of the data sets for which sufficient data is available show the same general tendency exhibited by the *yor-f-83* set. The data is sparse as of this writing but the behaviour of the best points for each year appears to indicate a trend. The improvement in later years is smaller than it was in earlier years and it is very unlikely that the value of the penalty will ever reach zero. This suggests that the best penalty points for each year may fall along a smooth curve that starts with some initial value, decreases slowly over the years and approaches some asymptotic non-zero positive value. This raises the question as to whether past performance can be used to forecast future results. If this is true, then perhaps an analytical study of past attempts to obtain lower standard penalties can be used to predict future lower standard penalties.

The problem of finding the best schedule arises from the sheer size of the solution space and the fact that the problem itself is NP-hard. For the *yor-f-83* or *yor83 I* data, the problem involves (a) partitioning the 181 exams into 21 timeslots and then

(b) permuting the order of these timeslots in order to find the resulting schedule having the lowest penalty.

The number of ways of partitioning n exams into k timeslots is given by a Stirling Number of the second kind

$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\} + k \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\} \tag{7}$$

with

$$\left\{ \begin{matrix} n \\ 1 \end{matrix} \right\} = \left\{ \begin{matrix} n \\ n \end{matrix} \right\} = 1 \tag{8}$$

Each set of partitions can be arranged in $k!$ ways.

Thus for the *yor83 I* data, there are $\left\{ \begin{matrix} 181 \\ 21 \end{matrix} \right\} = 4.09 \times 10^{219}$ ways to partition and $21! = 5.11 \times 10^{19}$ permutations of these partitions giving a total solution space of 2.09×10^{239} entries. As a basis of comparison, the total number of protons in the universe has been estimated to be roughly 10^{80}.

For the data set *yor-f-83*, the progressive "world record" was tabulated along with the year in which the work was published (see figure 5 right). The records correspond to the best result (if any) published during the corresponding calendar year. If the record was broken more than once during the year, the best result was taken.

The original data points obtained by Carter et al (1996) were not used in the analysis because it was the first time that the data and the penalty used were presented to the research world. The long time before the next result was published is not representative of the interval separating the next improvements. The logistic curve (6) was fitted to the remaining data points using the Solver tool in Microsoft Excel. The goal of the solver was to minimize the squared deviations between the published results and the fitted equation while adjusting the constants k, P_0 and r. When this is done, the limiting value of the standard penalty is calculated to be 32.44. A graph of the best fitted logistics curve, the data points and the limiting value P_{lim} is shown in figure 6. The same procedure was followed for some of the other data sets. The results obtained by this analysis is shown in table 1.

Table 1 Limiting penalty values

data set	no. points	P_{lim}	std.dev.	notes
car-f-92	5	3.82	0.08	1
tre-s-92	5	7.30	0.20	
ute-s-92	5	23.08	0.57	
yor-f-83	6	32.44	0.30	

The meaning of the first two columns of this table is obvious. Column three lists the limiting value of the penalty, $P_{lim} = k$, as calculated by the least squares fit. Column four, labelled *std.dev*, is a measure of the expected deviation of this limit and is calculated as:

$$\sqrt{\frac{1}{n-1} \sum (P_i - y_i)^2} \tag{9}$$

where

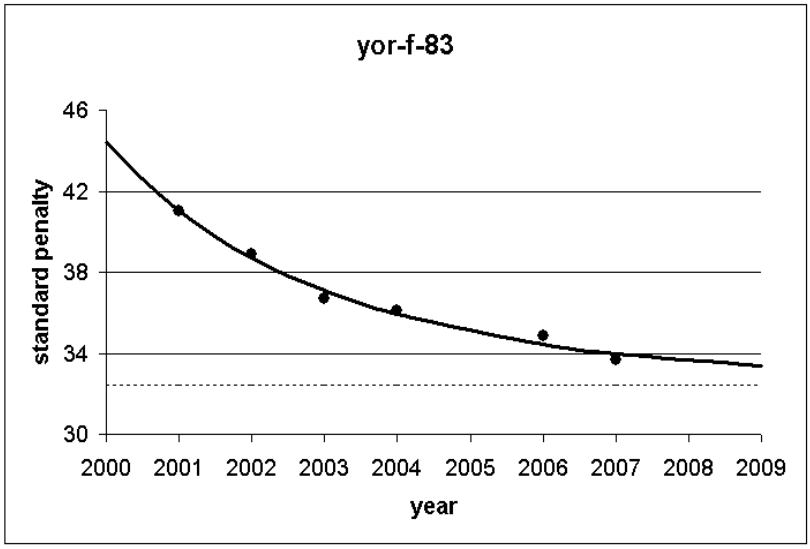


Fig. 6 The points, the best fit and the limiting value for the data set *yor-f-83*

- P_i is the value of the fitted curve for the year in question (referred to as $P(t)$ in the continuous domain of equation (6)).
- y_i is the value of the corresponding best penalty found in that year (if any).
- n is the number of points.

The column marked *notes* refers to the explanatory caveat listed below.

1. One paper, published in 2001, reports solution values that could be interpreted as being too good, too soon (see example in figure 1 (left)). This may be because of an error in the authors' calculations or it may be because of the superiority of their algorithms. In our calculation the reported value was omitted and the best value for that year was taken from the remaining values.

6 Conclusions

The data available for the TSP is far larger than the data for examination scheduling. The goodness of fit as defined by equation (9) equals 32.9, a value that is very small in comparison to the size of the problems now being successfully solved.

The numerical values of the limiting penalty given in table 1 should be used with caution. They are based on small amounts of data and their values will change as better solutions with smaller penalties emerge. The largest number of points available for any of the data sets is 6. The smallest number is 5; only just large enough for a curve to be fitted but not large enough to inspire great confidence in the result. Most of the data sets have few values in their running "best value" sets. One data set, *pur-s-93*, has so few results available that we were unable to even begin an analysis.

It must be realized that the values are asymptotic. Solutions having the limiting penalty will never be realised in finite time.

Not all the data sets yield a curve fit as well as does *yor-s-83* (see figure 6). The standard deviation, calculated as 0.030, is very good in this case. Values of accuracy that are not so good demonstrate either that the logistics curve is not a good description of the data or that the numerical parameters were badly fitted. This situation may be improved as more data becomes available.

There is no underlying theory developed as yet that supports the behaviour of the running best penalties observed over time. The basic examination scheduling problem is NP-hard and the best solution can be obtained only by exhaustive search.

The values observed and the limiting values calculated cannot be used in reverse to determine the schedule that produced them, *i.e.* there is no bijective relation between the solution space and the penalty space.

The x axis of the graphs corresponds to the year in which the data was published, not the year in which it was first obtained. This results in an uncertainty in these values. Also various publications use different accuracies in their reporting of the best penalties obtained. The difference in these accuracies may result in too many or too few data points selected in the list of best current values for a given year.

The limiting value cannot be used to rank solution methods. The fact that a certain algorithm yielded the latest best solution does not imply that a solution having that limiting penalty could be generated using that algorithm, or any other known algorithm.

However, the calculations can be used to estimate how close a given penalty is to a plausible minimum. This knowledge may then be used to calculate if the solution in question is "good enough" to use in practice. The best estimates of limiting values can be incorporated into algorithms and used as part of a stopping criterion.

They may also be used to estimate whether a penalty instance is reasonable, given the other penalties and the dates when they were obtained.

References

Applegate DL, Bixby RE, Chvatal V, Cook WJ (2006) The Travelling Salesman Problem: A Computational Study. Princeton University Press

Broder S (1964) Final examination scheduling. Communications of the ACM 7:494–498

Carter M, Laporte G, Lee S (1996) Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society 47:373–383

Carter M, Laporte G, Lee S (1997) Corrigendum. Journal of the Operational Research Society 48:225

Carter MW (1983) A decomposition algorithm for practical timetabling problems. Working Paper 83-06, Industrial Engineering, University of Toronto

Easton K, Nemhauser G, Trick M (2003) Solving the traveling tournament problem: A combined integer and constraint programming approach. In: Burke E, De Causmaecker P (eds) PATAT IV: Lecture Notes in Computer Science, Springer, Berlin, vol 2740, pp 63–77

Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman

Laporte G, Desroches S (1984) Examination timetabling by computer. Computers and Operations Research 11:351–360

Lawler E, Lenstra J, Kan AR, Shmoys D (1985) The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization. John Wiley and Sons

Peck JEL, Williams MR (1966) Algorithm 286: Examination scheduling. Communications of the ACM 9:433–434

Qu R, Burke EK, McCollum B, Merlot LTG, Lee SY (2009) A survey of search methodologies and automated system development for examination timetabling. J Scheduling 12(1):55–89

White GM, Chan PW (1979) Towards the construction of optimal examination schedules. INFOR 17(3):219–229

Timetable Construction: The Algorithms and Complexity Perspective (Plenary Talk)

Jeffrey H. Kingston

Abstract This paper advocates approaching timetable construction from the algorithms and complexity perspective, in which analysis of the specific problem under study is used to find efficient algorithms for some of its aspects, or to relate it to other problems. Examples are given of problem analyses leading to relaxations, phased approaches, very large-scale neighbourhood searches, bipartite matchings, ejection chains, and connections with standard NP-complete problems.

Keywords Timetabling · Algorithms · NP-completeness

1 Introduction

When tackling a problem, a researcher utilizes a certain perspective, or set of techniques, which he or she understands and has experience with. There is a tendency to stay within one's own perspective, which, though natural, may not be the most scientific thing to do.

One of the strengths of the PATAT conference series is that its contributors bring a variety of perspectives to the timetabling problems they study, thereby exposing its participants to healthy doses of unfamiliar techniques. Judging by the contributions to the most recent conference (Burke and Gendreau 2008), the field is strongly dominated by local search, especially simulated annealing (Dowsland 1993; Kirkpatrick et al. 1983) and tabu search (Glover and Laguna 1998); but there are also papers written from the operations research perspective, and indeed their number is growing as integer programming packages improve. Other perspectives are also represented, although on a smaller scale: constraint programming, machine learning, and cooperating agents are three examples.

Some of the most interesting papers apply techniques from one perspective to problems that had previously been studied only from another. For example, the Travelling Tournament Problem (Easton et al. 2003) was formulated by researchers associated with the operations research perspective, but good solutions were later obtained with local search (Ribeiro and

Jeffrey H. Kingston
School of Information Technologies
The University of Sydney, NSW 2006, Australia
http://www.it.usyd.edu.au/~jeff
E-mail: jeff@it.usyd.edu.au

Urrutia 2004). And, in the reverse direction, two simplified university course timetabling data sets, compiled by researchers who typically use local search, were recently solved to optimality using an integer programming package (Burke et al. 2008).

This paper advocates the *algorithms and complexity* perspective on timetable construction. In general terms, a researcher who utilizes the algorithms and complexity perspective will devote considerable time to analysing the specific problem under study. The outcome of this *problem analysis* might be the discovery that some aspect of the problem is amenable to efficient solution, leading to the design of an algorithm which exploits that fact. In other cases, the outcome might be the discovery of a close connection with another NP-complete problem, which can be helpful in pointing to a body of relevant prior work.

Although problem analysis is practised by all researchers, it is less emphasized within the dominant perspectives. Researchers who use local search, for example, typically devote most of their time to empirical work and parameter tuning. Indeed, a major advantage of local search is the ease with which it can be applied to a wide range of problems. The idea of basing a solution approach on detailed properties of the problem under study may even be deprecated, as tying the algorithm too closely to specific conditions that could change.

Operations researchers tend to concentrate on modelling, leaving the algorithmic aspects to software packages. Nevertheless a great deal of interesting problem analysis has been carried out by operations researchers over the years, leading to redundant but beneficial additional constraints, or to phased approaches, Lagrangean relaxation (see below), and so on. A major advantage of the operations research approach is its emphasis on lower as well as upper bounds: either an optimal solution is found, or a lower bound is produced which gives some indication of how close to optimality the solution lies.

It is emphatically not the aim of this paper to show that algorithms and complexity techniques will always be superior, or even that they will always yield anything useful. They are too dependent on specific properties of particular problems for that. Instead, this paper offers some examples where the techniques are useful, enough, I hope, to show that they are worth adding to the mental toolkit of the timetabling researcher. The topics covered are relaxation, the phased approach, very large-scale neighbourhood search, bipartite matching, ejection chains, and NP-completeness analysis. Although this paper includes a few original constructions and experiments, it is offered more as a tutorial than as a research contribution.

2 Relaxation

One way to find problems of low complexity within an NP-complete problem is to *relax* the NP-complete problem: loosen some of its constraints, or discard them altogether. Although the solution of the resulting *relaxed problem* is not usually a solution of the original problem, it may contain useful information. In particular, the badness of an optimal solution of the relaxed problem is a lower bound on the badness of any solution of the original problem.

Finding relaxations requires problem analysis. On the one hand, the relaxed problem must be sufficiently close to the original for its solution to be relevant; on the other, it must be efficiently solvable, since otherwise nothing is gained by using it.

Relaxation is an important tool for researchers using the operations research perspective. The archetypal relaxation replaces the *integrality* constraints of an integer program (specifying that each variable x_i must be assigned an integer in some range $a_i \ldots b_i$) with *linear* constraints (specifying that each variable x_i must be assigned some value in the range $a_i \leq x_i \leq b_i$, with fractional values allowed), replacing an integer program, which in general describes an NP-complete problem, by a linear program which can be solved in polynomial

time. Another technique well known to operations researchers is *Lagrangean relaxation* (Beasley 1993), in which different versions of the relaxed problem are solved repeatedly.

Relaxation is useful for determining whether a problem is likely to have a feasible solution. For example, the nurse rostering problem has many complex constraints on the layout of each nurse's shifts. Discarding them leaves a much simpler problem, solvable by bipartite matching (Sect. 5), which checks whether there are enough nurses of the right kinds to cover the required work. There is little point in starting a long solution process if not.

3 The phased approach

The *phased approach* divides the problem into parts, called *phases*, and solves them one by one. Each phase has only limited information about the other phases, so there is little hope of the overall solution being optimal. A few scattered examples exist where the results of a later phase are fed back to a subsequent run of an earlier phase.

Problem analysis is needed to find a decomposition into phases which are efficiently solvable separately, and independent enough to satisfy concerns about the loss of optimality.

Large timetabling problems, such as whole-university course timetabling and student sectioning problems, are typically solved in phases (Carter 2000; Murray et al. 2007). A key module in a student sectioning system is a branch-and-bound algorithm for finding the best possible timetable for one student, holding the rest of the timetable fixed. One run of such an algorithm constitutes one phase.

In unpublished work by staff of the University of Sydney more than a decade ago, after timetabling each student in this way, a second pass over the student list was made, and each student was removed and re-timetabled. This introduces the feedback mentioned above, as well as being an example of very large-scale neighbourhood search, to be described next.

4 Very large-scale neighbourhood search

Very large-scale neighbourhood (VLSN) search (Ahuja et al. 2002) is a form of local search. To move from one solution to its neighbour, a large piece of the solution is deassigned, then reassigned in a different and hopefully improved way.

Although the reassignment stage can be carried out by any simple constructive heuristic, the method is particularly interesting when problem analysis identifies a piece to deassign whose reassignment may be carried out optimally.

Several examples of the application of VLSN search to timetabling problems are given in Meyers and Orlin (2007). The point is made there that in some cases where other local search methods proceed by swaps, a more general and potentially more effective VLSN search based on the 'cyclic exchange neighbourhood' is possible. This author has used this neighbourhood in high school teacher assignment, to permute the assignments at a given set of times among the available teachers, and found it effective in removing certain kinds of bad split assignments (Kingston 2008). For variety, this paper offers a different example, again from teacher assignment.

The literature contains a smattering of timetabling papers which improve their resource assignments by deassigning all the work assigned to two resources and then reassigning that work to those same two resources. This clearly qualifies as VLSN search, and when the resources are high school teachers, the reassignment can be done to optimality in practice.

Take the example of teachers *Art01* and *Art02* in Fig. 1. These make good candidates

	Avail	W1	W2	R1	R2	M8	F5	M3	M4	W5	W6	R8	F2	T3	T4	R5	R6	W8	F3
Art01	0	12-3A-P	7CK02-	12-3A-Photograph						7AS2-1			History	8CKO2-1		8CKO4-2			
Art02	4	7CK02-	11-3A/1			11-3A/12-3A-Cera				10-4-Art				12-1-VisualArts					
Art03	0	7CKO1-1								7AS1-1				8CKO1-1		8CKO3-2			
Unassigned																			

	M1	M2	T5	T6	W7	F4	M5	M6	T1	T2	R7	F1	W3	W4	R3	R4	M7	T7	F6	F7	F8	T8
Art01	11-4-VisualArt						12-2-Photography-2U						11-6-Photography						Sport			StaffMe
Art02	8AS3-3		7AS3-3		9-4Art-1				9-4Art-1				7CKO4-2		10-4-Art			10-4-Art		8AS2-1		
Art03													7CKO3-2							8AS1-1		
Unassigned			12-4B-VisualDesig			12-4B-V																

Fig. 1 Part of a high school timetable, showing the assignments of the school's three Art teachers. Each teacher occupies one row, with a fourth row holding one Art class which failed to be assigned. Each column holds one of the 40 times of the cycle, except the column adjacent to the teachers' names, which shows the remaining available workload of the teachers.

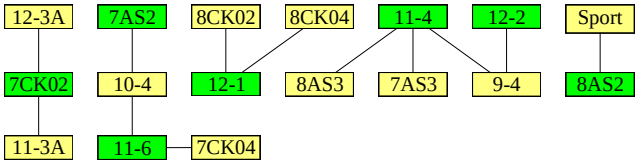


Fig. 2 The clash graph for the meetings assigned to teachers *Art01* and *Art02* in Fig. 1, slightly simplified, and showing one 2-colouring.

for deassignment and reassignment, because they are qualified to teach similar kinds of classes, and they share a split assignment (occupying times $W1$ and $W2$), whose removal is desirable. Deassign all their work and build its clash graph, in which each deassigned meeting is a node and two nodes are joined by an edge whenever their meetings share at least one time (Fig. 2). Now a clash-free reassignment is a 2-colouring of this graph. Each of the graph's K connected components can be coloured independently, and has two distinct 2-colourings; in various special cases there are fewer, making at most 2^K distinct colourings altogether. In practice, K is small enough to permit an exhaustive search for a colouring that does not exceed the teachers' workload limits. For safety, the author's implementation imposes a fixed upper limit on the number of colourings tried.

On real instances this method runs quickly, but its results are disappointing, averaging only about one improvement per instance. Still, it might be useful in other resource assignment problems, such as nurse rostering, where similarly qualified resources work together.

5 Bipartite matching

A *bipartite graph* is an undirected graph whose nodes may be divided into two sets, such that every edge connects a node of one set to a node of the other. A *matching* in an undirected graph (bipartite or otherwise) is a subset of the edges such that no two edges touch the same node. A *maximum matching* is a matching containing as many edges as possible (Fig. 3). The *bipartite matching problem* is the problem of finding a maximum matching in a bipartite graph. There is a standard polynomial-time algorithm for this problem, used in timetabling for more than forty years (Csima and Gotlieb 1964; Gotlieb 1962; de Werra 1971).

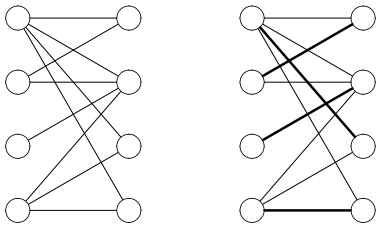


Fig. 3 A bipartite graph (left), and the same graph with a maximum matching, shown in bold (right).

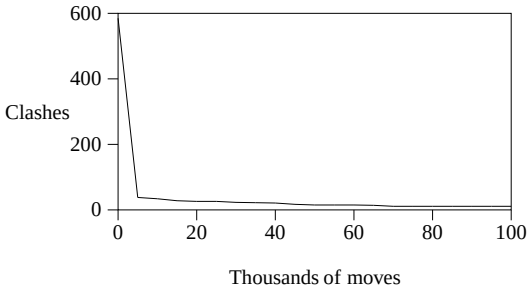


Fig. 4 Performance of tabu search on a bipartite matching problem with 3686 demand nodes and 5378 supply nodes, taken from a real instance of a high school timetabling problem, whose optimal solution is known (by applying the standard polynomial-time algorithm) to have 5 unmatched demand nodes, or equivalently (if every demand node is assigned) 5 clashes. After 100000 moves, starting from an initial greedy solution with 585 clashes, the best solution found had 11 clashes. The neighbourhood was all single moves of a demand node's assignment from one supply node in its domain to another. A move was tabu if it involved a demand node that had been moved recently; the tabu list length was 1500 demand nodes. About ten values for tabu list length were tried; results improved as the length was increased to 1500, and then worsened.

In timetabling, it is usual for one of the two sets of nodes to represent variables (or slots, meetings, etc.) demanding something to be assigned to them, while the other set represents entities (times, resources, etc.) which are available to supply these demands. Accordingly, these two sets will be referred to as the *demand nodes* and the *supply nodes*. A maximum matching assigns supply nodes to as many demand nodes as possible, under the restrictions that each demand node requires one supply node from the set of supply nodes it is connected to, and that each supply node may be assigned to at most one demand node.

One application of bipartite matching to timetabling is in the assignment of rooms to meetings after the meetings' times are fixed. At each time, build a bipartite graph with one demand node for each demand for a room at that time, and one supply node for each room which is available at that time, and connect each demand node to those supply nodes representing rooms which are qualified to satisfy the demand (rooms which are large enough and contain the appropriate facilities). Then a maximum matching gives an optimal assignment of rooms at that time.

Fig. 4 documents a case where a standard local search method (tabu search) could not find an optimal solution to a large, real instance of the bipartite matching problem, even when several settings of its parameters were tried and ample time was allowed. Thus, when instances of bipartite matching problems lie within timetabling problems, it may be advantageous to solve them directly using the polynomial-time algorithm, as was done, for example, by the winning entry in the First International Timetabling Competition (Kostuch 2005).

The performance of tabu search on this problem raises a question: if the problem is difficult, then how can the standard algorithm solve it to optimality so quickly? Can that

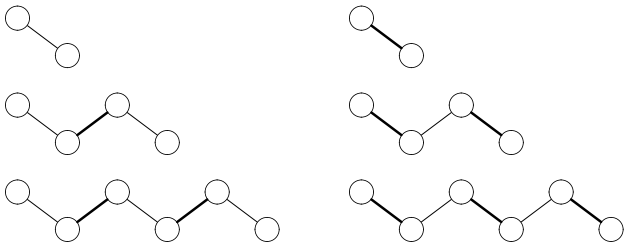


Fig. 5 Augmenting paths (at left) and the effect of applying them (at right). The first augmenting path carries out a simple assignment; the second carries out two assignments and one deassignment; and so on. In each case the size of the matching increases by one. No matching edge may initially touch the first or last node.

algorithm be applied to other problems, perhaps to NP-complete problems? To answer these questions it is necessary to examine the standard algorithm in detail.

The algorithm is called the *augmenting path method*. Starting at each unmatched demand node in turn, it searches the graph for a path from that node to a supply node, from there back to the demand node currently assigned that supply node, from there to a different supply node, and so on, ending at a currently unmatched supply node. Then making each non-matching edge on the path into a matching edge, and each matching edge on the path into a non-matching edge, increases the size of the matching by one (Fig. 5). A theorem guarantees that each node has to be searched through only once, so the cost of finding an augmenting path is bounded above by the total size of the graph; and another theorem guarantees that after each unmatched demand node has been taken as the starting point, the matching is maximum.

To dispel any idea that this algorithm is difficult to implement, here is the key procedure, for searching for an augmenting path out of a given demand node, and applying it if found:

```
bool Augment(DEMAND_NODE demand_node, int visit_num)
{
    int i;  SUPPLY_NODE supply_node;
    for( i = 0;  i < demand_node->domain_size;  i++ )
    {
        supply_node = demand_node->domain[i];
        if( supply_node->visit_num < visit_num )
        {
            supply_node->visit_num = visit_num;
            if( supply_node->supply_asst == NULL ||
                Augment(supply_node->supply_asst, visit_num) )
            {
                supply_node->supply_asst = demand_node;
                demand_node->demand_asst = supply_node;
                return true;
            }
        }
    }
    return false;
}
```

Other code is needed for initialization and trying each demand node in turn.

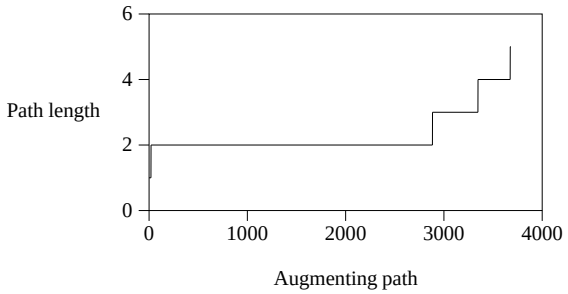


Fig. 6 This graph shows how the length of augmenting paths increases as the bipartite matching algorithm proceeds. The bipartite graph from Fig. 4 was used. For each value of k from $k = 1$ to $3686 - 5$, the length of the longest of the first k augmenting paths found is shown, defining length to be the number of demand nodes on the path. At first, the paths are short (at most 2 demand nodes), but by the end of the algorithm they have length 4 or 5. Breadth-first search was used to find these paths, so this shows that some steps near the end require *at least* 4 or 5 reassignments of demand nodes in order to improve solution quality.

The secret of the success of this algorithm is revealed in Fig. 6. At first, it finds very short augmenting paths, such as any local search algorithm could easily find. But towards the end, the paths become longer, until, on large examples such as the one used in the figure, at least 4 or 5 coordinated reassignments of demand nodes to supply nodes are required to improve the solution. This is difficult for local search algorithms based on simple moves and swaps: they become trapped in what seem to them to be large, featureless plateaus.

6 Ejection chains

It is not hard to see how to apply the augmenting path method of the previous section to assignment-type problems other than bipartite matching. Start at any point where an assignment is required but is currently missing. Mark all elements of the instance unvisited. Try to assign a valid value at the starting point. If that can be done directly, do it; otherwise, find all ways in which a valid value can be assigned, at the cost of one deassignment at some other point. For each of these ways, make the indicated deassignment and assignment, mark the elements involved as visited to ensure that they will not be touched again during the current search, and continue trying to reassign the deassigned element, using the same method recursively. If the search ever reaches an element that can be assigned directly, it does so and terminates, having completed a chain of assignments and deassignments which amount to an augmenting path. Repeat until no further progress occurs.

This idea was given the name *ejection chains* by Glover (the inventor of tabu search), who applied it successfully to the travelling salesman problem (Glover 1996). Similar ideas had probably been used earlier. For example, the widely used Kempe chain method from graph colouring, dating from the work of A. B. Kempe in 1879, could be described as an ejection chain method, although it differs in detail from the method presented here. An accessible account appears in Dowsland (1993).

In general, theorems which guarantee effectiveness (as in the bipartite matching case) will not be available; nevertheless, ejection chains preserve the other virtue of the augmenting path method, namely its ability to explore large plateaus.

A key restriction of the ejection chain method as formulated here is that only one deassignment is permitted for each assignment, ensuring that the structures searched are limited to paths. Although more complex augmenting structures, such as trees, could be permitted,

Fig. 7 Performance of tabu search on a typical teacher assignment problem (*bghs93* from Kingston 2008) with 305 teacher slots totalling 1275 times. The badness measure was the total number of clashes and hard workload limit overloads. After 4000 moves, starting from an initial greedy solution with badness 79, the best solution found had badness 23. The neighbourhood was all moves of one assignment from one qualified teacher to another. A move was tabu if its slot had been moved recently; the tabu list length was 50, which gave the best result of about ten values tried.

they complicate the implementation and seem less likely to pay off than paths. See Müller et al. (2005) for a simple method which can explore such structures.

This author has used ejection chains to improve the assignment of teachers to meetings in high school timetabling problems, after the meetings' times are assigned. Each meeting may contain several time blocks spread through the week, and may request a teacher of a particular kind. Vertex colouring may be embedded in this problem, making it NP-complete.

First, an initial assignment is made by taking each teacher in turn and applying a branch-and-bound tree search (with a fixed upper limit on the number of nodes searched, for safety) to pack as much workload as possible into the teacher, assigning only meetings for which the teacher is qualified, and avoiding clashes and hard workload limit overloads. Next, from each unassigned meeting the algorithm searches for ejection chains as described above. Finally, *split assignments* are introduced, in which the classes of unassigned meetings are split between two or more qualified teachers. Split assignments are undesirable, so it is important to minimize the number of unassigned meetings at the point they are resorted to. A full description has been given elsewhere (Kingston 2008).

It is interesting to compare the performance of this algorithm with a standard local search (tabu search). It is not clear how to do so fairly, however, since the author's algorithm never introduces a clash or a hard workload limit overload, preferring to leave a slot unassigned, whereas tabu search assigns every slot, at the cost of some clashes and hard workload limit overloads. There is no simple and fair means of interconversion as there was for the bipartite matching problem studied earlier.

When run on a typical instance, the author's algorithm produced a resource assignment in which there were 22 unassigned meetings after the initial assignment, and 15 after the ejection chain phase. This was a significant improvement, since it meant that 7 fewer meetings required split assignments. For comparison only, this solution was extended greedily to one in which every slot had an assignment, and that solution had 34 clashes and hard workload limit overloads.

The best run of tabu search (Fig. 7) on the same instance produced 23 clashes and hard workload limit overloads. Again for comparison only, this solution was reduced greedily to one in which there were no clashes or hard workload limit overloads, and that solution had 22 unassigned meetings. Even if we call these results a draw, the ejection chain method still has considerable advantages: it runs much faster, and there are no parameters to tune.

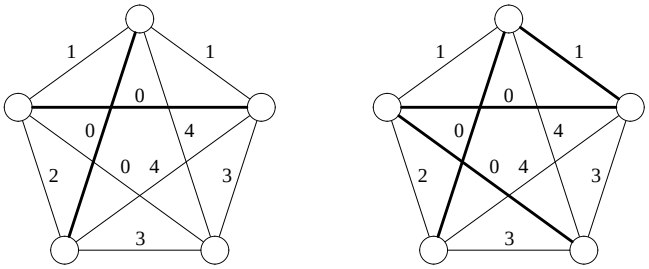


Fig. 8 A clash graph, showing a minimum matching (left), and a travelling salesman path (right).

7 NP-completeness analysis

Sometimes the outcome of problem analysis is not an idea for an efficient algorithm, but instead the discovery of a connection with another NP-complete problem. Even this apparently negative result may be useful, however, in suggesting that prior work on the other problem may be relevant, either directly or with some adaptations.

Two examples of this occur in examination timetabling, one well-known, the other less so. The well-known example is the connection with vertex colouring. Algorithms from the vertex colouring literature, such as the saturation degree heuristic and Kempe chains, have been adapted to examination timetabling in many papers.

To uncover the less well-known connection, it is necessary first to dispose of the 'no-clashes' constraint that points to vertex colouring. This may be done, for example, by a phased approach whose first phase clusters examinations so that there are as many clusters as time slots. For simplicity, this discussion will assume from now on that there are as many examinations as time slots, and that the aim is to assign one examination to each time slot.

With clashes out of the way, the remaining problem is to minimize cases of students having examinations too close together in time. This requirement can be formalized in several ways, two of which will be examined here.

Construct the familiar *clash graph*, in which each examination (or cluster of examinations) is represented by a node, and each pair of nodes is joined by an edge weighted by the number of students who attend the examinations of both nodes. One formulation is to have two examination time slots on each day, and aim to minimize the number of cases of students attending two examinations on the same day. This corresponds to finding a maximum matching of minimum total weight in the clash graph, which can be done in polynomial time. Another formulation does not consider the division of time slots into days, but merely their sequencing in time, and aims to minimize the number of cases of students having consecutive examinations. This corresponds to finding a travelling salesman path (like a travelling salesman tour, but with no requirement to end at the starting pont) in the clash graph. These constructions are illustrated in Fig. 8.

It is not suggested that these ideas provide an immediate solution to the examination timetabling problem. Rather, they make connections with other work that might bear fruit when suitably adapted. For example, to the author's knowledge, no attempt has been made to adapt the work of Glover (1996) on ejection chain neighbourhoods for the travelling salesman problem to examination timetabling.

8 Conclusion

This paper has highlighted the algorithms and complexity perspective on timetable construction, and shown by example that its use can be beneficial.

It is difficult to offer guidance in the application of the techniques advocated here, since they must be adapted to specific details of the problems under study. Familiarity with the list of standard algorithms and NP-complete problems is a prerequisite. In searching for algorithmic ideas, a focus on sets of related variables is often rewarding: the schedules of all sports teams in a given local area, the room requirements for all meetings at a given time, and so on. Indeed, where algorithms and complexity techniques have advantages, these seem to be due to their ability to handle sets of related variables together, rather than one by one as local search and integer programming solvers do. This point is illustrated repeatedly throughout this paper.

There is no practical barrier to combining algorithms and complexity techniques with other perspectives in timetabling research; the possibilities are limited only by our ingenuity. There is however one point of philosophical disagreement which should not be glossed over.

Metaheuristics are *general* approaches to optimization problems, easily applied to any problem. Integer programming, too, is a general approach. From the algorithms and complexity perspective, generalization is bad, not good, because it brings with it a corresponding weakening of the tools available to solve the problems. For example, the consequences of treating the bipartite matching problem as a general optimization problem were demonstrated in Sect. 5. Thus, the algorithms and complexity perspective will tend to lead towards specific details, while the other perspectives lead in the opposite direction.

Problem analysis is a very hit-and-miss process, but it does have the advantage of being open-ended. One can always hope to find a new algorithm, or a new connection with another problem. And when something does turn up, the payoff can be large. For these reasons, the algorithms and complexity perspective will continue to have a place in timetabling research.

References

Ahuja R, Ergun Ö, Orlin J, Punnen A (2002) A survey of very large-scale neighbourhood search techniques. Discrete Applied Mathematics, 123:75–102

Beasley JE (1993) Lagrangean relaxation. In Reeves CR (ed.), Modern Heuristic Techniques for Combinatorial Problems, Blackwell

Burke EK, Gendreau M (2008) Proceedings, PATAT2008 (Seventh international conference on the Practice and Theory of Automated Timetabling, Montreal)

Burke EK, Marecek J, Parkes AJ, Rudová H (2008) A branch-and-cut procedure for Udine course timetabling. In: Proceedings, PATAT2008 (Seventh international conference on the Practice and Theory of Automated Timetabling, Montreal)

Carter MW (2000) A comprehensive course timetabling and student scheduling system at the University of Waterloo. In Practice and Theory of Automated Timetabling III (Third International Conference, PATAT2000, Konstanz, Germany, Selected Papers), Springer Lecture Notes in Computer Science 2079:64–81

Csima J, Gotlieb CC (1964) Tests on a computer method for constructing school timetables. Communications of the ACM 7:160–163

Dowsland KA (1993) Simulated annealing. In Reeves CR (ed.), Modern Heuristic Techniques for Combinatorial Problems, Blackwell

Easton K, Nemhauser G, Trick M (2003) Solving the travelling tournament problem: a combined integer programming and constraint programming approach. In: Practice and Theory of Automated Timetabling IV (Fourth International Conference, PATAT2002, Gent, Belgium, August 2002, Selected Papers), Springer Lecture Notes in Computer Science 2740:100-109

Glover F (1996) Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65:223–253

Glover F, Laguna M (1998) Tabu Search, Kluwer

Gotlieb CC (1962) The construction of class-teacher timetables. Proc. IFIP Congress, 73–77

Kingston JH (2008) Resource assignment in high school timetabling. In: PATAT2008 (Seventh international conference on the Practice and Theory of Automated Timetabling, Montreal)

Kirkpatrick S, Gellat CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

Kostuch P (2005) The university course timetabling problem with a three-phase approach. In: Practice and Theory of Automated Timetabling V (5th International Conference, PATAT 2004, Pittsburgh, PA), Springer Lecture Notes in Computer Science 3616:109–125

Meyers C, Orlin JB (2007) Very large-scale neighbourhood search techniques in timetabling problems. In: Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Brno, Czech Republic), Springer Lecture Notes in Computer Science 3867:24–39

Müller T, Rudová H, Barták R (2005) Minimal perturbation problem in course timetabling. In: Practice and Theory of Automated Timetabling V (5th International Conference, PATAT 2004, Pittsburgh, PA), Springer Lecture Notes in Computer Science 3616:126–146

Murray K, Müller T, Rudová H (2007) Modeling and solution of a complex university course timetabling problem. In: Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Brno, Czech Republic), Springer Lecture Notes in Computer Science 3867:189–209

Ribeiro CC, and Urrutia S (2004) Heuristics for the mirrored travelling tournament problem. In: Proceedings, PATAT 2004 (5th International Conference on the Practice and Theory of Automated Timetabling, Pittsburgh, PA), 323–341

De Werra D (1971) Construction of school timetables by flow methods. INFOR – Canadian Journal of Operations Research and Information Processing 9:12–22

The Train Driver Recovery Problem – Solution Method and Decision Support System Framework

David M. Ryan
The Department of Engineering Science
The University of Auckland, New Zealand
e-mail: d.ryan@auckland.ac.nz

Natalia J. Rezanova
DTU Management Engineering
Technical University of Denmark, Denmark
e-mail: natalia.rezanova@gmail.com

Every railway operator experiences disruptions during the daily operations. Danish railway operator DSB S-tog A/S is no exception. DSB S-tog A/S operates on an urban train network with at least 6 trains per hour in each direction departing from every station of the network and up to 30 trains per hour in each direction departing from the Copenhagen central station. Minor train delays on the network are recovered by re-establishing the original plan using the slack time built into the timetable or delaying other trains. Major disruptions in the train schedule are recovered by re-routing or cancelling trains. A train is re-routed if it is turned back before reaching the end terminal station or driven through some stations without stopping. A cancellation is applied either to a single train task or to a whole train line, resulting in cancellations of all train tasks of a particular line for a certain period of time.

Disruptions in the train timetable affect the train driver schedule. When a train is delayed, re-routed or cancelled, a driver might be late for the next scheduled train task of the duty. If the driver is not available in due time for a train departure, the train task is assigned to another driver. If there is no available driver to cover the train task, the train is delayed or cancelled, causing a propagation of disruptions in the schedule. At the present time the operational re-scheduling process of disrupted train driver duties is conducted manually. If the disruption is severe and many train driver duties are disturbed, this is a very complicated task to carry out.

The interest of the passenger railway operator DSB S-tog A/S in introducing automated decision support for the train driver dispatchers is a key motivation for this project. The project has been a part of a Ph.D.-study [4] at Operations Research section of the Department of Management Engineering at the Technical University of Denmark in 2006 – 2009, and is now handed over to the Analysis Group of the Planning Department at DSB S-tog A/S.

The train driver re-scheduling has received a very limited attention by Operations Research practitioners. An integer programming approach to a simultaneous train timetable and crew roster recovery problem, tested on the New Zealands Wellington Metro line, is presented in [8]. The crew re-scheduling problem for train driver duties disrupted due to the maintenance work on train tracks is solved by [2] for the largest passenger railway operator in The Netherlands, while [3] present an algorithm for operational re-scheduling of the train drivers on the Dutch railway network. The review paper [1] describes the topic of crew recovery within the airline industry.

We propose an optimization-based solution method for solving the *Train Driver Recovery Problem* (TDRP) and a prototype for the decision support system for the train driver dispatchers. The optimization framework is based on solving restricted TDRP instances with a rolling time horizon, aiming to modify the original duty schedule as little as possible. For a particular disruption we identify a *disruption neighbourhood*, which is a part of the driver schedule characterized by a set of train tasks and a set of train drivers. The initial disruption neighbourhood is identified by a set of drivers, who's duties contain train tasks which are known to be disrupted within a certain recovery period, a time period within which a recovery solution is aimed to be found. A train task is disrupted if it is delayed, cancelled, re-routed or uncovered, i.e. assigned to an absent driver. All train tasks belonging to the initial set of drivers within the recovery period are included into the initial disruption neighbourhood. The Train Driver Recovery Problem (TDRP) aims at finding a set of feasible train driver recovery duties for drivers within the disruption neighbourhood with minimum modification from the original train driver schedule, such that all train tasks within the recovery period are covered and the driver duties outside the recovery period and duties of drivers not included in the disruption neighbourhood are unchanged. If a feasible recovery solution is not found withing a certain disruption neighbourhood, the disruption neighbouhood is expanded by either adding more train drivers or expanding the recovery period of the problem.

The TDRP is formulated as a set partitioning problem, where variables represent recovery duties of train drivers. The set of generalized upper-bound train driver constraints ensure that each train driver is assigned to exactly one recovery duty in the schedule. The train task constraints have a set partitioning structure and ensure that each train task in the recovery schedule is covered exactly once. It is observed in [7] that the linear programming relaxation of the set partitioning formulation of the crew rostering problem, which has a similar structure to the TDRP, possesses strong integer properties due to the existence of the generalized upper-bound crew constraints, which contribute to the perfect structure of the submatrix, corresponding to each crew member.

The solution method for solving the Train Driver Recovery Problem is based on solving the TDRP-LP and finding an integer solution with a constraint branching strategy. Since the cost of the recovery is not determined by a physical cost of the driver schedule (the drivers are already paid to be at work), but rather by the fictitious cost which expresses how attractive each recovery duty is, the optimality of the solution is not as important as the feasibility of the solution. The TDRP-LP is solved with a column generation method based on a limited subsequence strategy, where recovery duties with negative reduced costs are generated by limiting

the number of tasks (subsequences) a driver can perform after finishing any task in the duty. Starting with a small number of subsequences, it is gradually increased, allowing to consider less attractive subsequent tasks for recovery duties. When a feasible solution to the TDRP-LP is found, we consider the problem solved. If the initial number of drivers is not enough to cover all train tasks in the initial disruption neighbourhood, the disruption neighbourhood is expanded in two possible ways: either the number of drivers in the disruption neighbourhood is increased by adding available stand-by drivers or the recovery period is extended, including more train tasks from the involved drivers' duties. If the problem remains infeasible due to uncovered train tasks when there are no more available drivers to add to the disruption neighbourhood, the decision support system sends an infeasibility message to the dispatcher specifying which train tasks are uncovered and hence have to be delayed or cancelled.

If the solution to the TDRP-LP is fractional, a constraint branching strategy similar to the one described in [6] is applied in order to find an integer solution. Since every train driver submatrix in the set partitioning formulation of the problem is perfect, the fractions occur in the TDRP-LP only across train drivers' blocks of columns. It is therefore sensible on 1-branches of the Branch & Bound tree to force one driver r to cover a train task s, which also appears in another driver's optimal recovery duty while forbidding other drivers to include s in their recovery duties. On the 0-branch we forbid the driver r to cover the train task s. A depth-first search on 1-branches of the Branch & Bound tree is implemented and the branching procedure is terminated as soon as the first integer solution is found.

Real-life operational data is provided by DSB S-tog A/S in order to test the implemented solution method. Based on the computational experiments, we conclude that the proposed approach is indeed applicable for implementation in a decision support system for train driver dispatchers in practice. DSB S-tog A/S is working on using the research results obtained during this thesis and the programming code of the prototype to develop and implement the train driver decision support system in their operational environment.

References

[1] J. Clausen, A. Larsen, J. Larsen, N. J. Rezanova, *Disruption management in the airline industry–Concepts, models and methods*, Computers and Operations Research, 37 (2010), pp 809–821.

[2] D. Huisman, *A column generation approach for the rail crew re-scheduling problem*, European Journal of Operational Research, 180 (2007), pp 163–173.

[3] D. Potthoff, D. Huisman, G. Desaulniers, *Column Generation with Dynamic Duty Selection for Railway Crew Rescheduling.*, accepted for publication in Transportation Science, 2010.

[4] N. J. Rezanova, *The Train Driver Recovery Problem – Solution Method and Decision Support System Framework*, Ph.D.-thesis, Department of Management Engineering, Technical University of Denmark, 2009.

[5] N. J. Rezanova, D. M. Ryan, *The train driver recovery problem–A set partitioning based model and solution method*, Computers and Operations Research, 37 (2010), pp 845–856.

[6] D. M. Ryan, *The solution of massive generalized set partitioning problems in aircrew rostering*, The Journal of the Operational Research Society, 43 (1992), pp 459–467.

[7] D. M. Ryan, J. C. Falkner, *On the integer properties of scheduling set partitioning models*, European Journal of Operational Research, 35 (1988), pp 442–456.

[8] C. G. Walker, J. N. Snowdon, D. M. Ryan, *Simultaneous disruption recovery of a trian timetable and crew roster in real time*, Computers and Operations Research, 32 (2005), pp 2077–2094.

Papers

Curriculum-based Course Timetabling with SAT and MaxSAT

Roberto Asín Achá · Robert Nieuwenhuis

Abstract We introduce novel and strong techniques based on propositional satisfiability (SAT) solvers and optimizers (MaxSAT solvers) for handling the Curriculum-based Course Timetabling problem.

Out of 32 standard benchmark instances derived from the last International Timetabling Competition, our techniques improve the best known solutions for 10 of them (4 of these 10 being optimal), and for another 9 we match the best known solution (8 of them to optimality).

There is still much room for improvement in the encodings we use as well as in the underlying general-purpose SAT and MaxSAT solvers.

Keywords Timetabling · SAT · MaxSAT

1 Introduction

The problem of deciding the satisfiability of propositional formulas (SAT) does not only lie at the heart of the most important open problem in complexity theory (P vs. NP), it is also at the basis of many practical applications in such areas as Electronic Design Automation, Verification, Artificial Intelligence and Operations Research. Thanks to recent advances in SAT-solving technology, propositional solvers are becoming the tool of choice for attacking more and more practical problems by encoding them into SAT.

Example 1 The propositional clause set $\{ \neg x_1 \vee \neg x_2 \vee \neg x_3, \quad x_1, \quad x_2 \vee \neg x_3 \}$ is satisfiable: the assignment $\{x_1, x_2, \neg x_3\}$ is a model of it. If the clause $\neg x_1 \vee x_3$ is added, the set of clauses becomes *unsatisfiable*. A (complete) *SAT solver* is a tool that, given a set of clauses, either finds a model for it or reports its unsatisfiability. □

There exist several optimization versions of the SAT problem. In *MaxSAT* the aim is to find a model that maximizes the number of satisfied clauses. In *Partial MaxSAT* the input consists of two sets of clauses, the *hard* ones and *soft* ones, and the problem

Both authors address: Technical University of Catalonia, Barcelona, `www.lsi.upc.edu/~rasin` and `~roberto`. Both are partially supported by Spanish Min. of Educ. and Science through the LogicTools-2 project (TIN2007-68093-C02-01).

is to find a model for the hard clauses that maximizes the number of satisfied soft clauses. In *Weighted (Partial) MaxSAT* each soft clause has a *weight* and the aim is to minimize the sum of the weights of the falsified soft clauses.

Here we apply novel SAT, Partial MaxSAT, and Weighted (Partial) MaxSAT encodings for handling the Curriculum-based Course Timetabling problem. For this purpose we have used our own general-purpose *Barcelogic* SAT and Partial MaxSAT solvers, as well as several other solvers for (Weighted) Partial MaxSAT. We emphasize that these solvers are based on complete search, that is, they always terminate (when given sufficient resources), in SAT returning a model or an unsatisfiability answer, and in MaxSAT, they always find the optimal solution.

Out of 32 standard benchmark instances derived from the last International Timetabling Competition (Di Gaspero et al (2007)), our techniques improve the best known solutions for 10 of them (4 of these 10 being optimal), and for another 9 we match the best known solution (8 of them to optimality). These facts can be checked at the website `http://tabu.diegm.uniud.it/ctt`. There is still much room for improvement in the encodings we use as well as in the underlying general-purpose SAT and MaxSAT solvers.

This paper is structured as follows. In Section 2 we give basic definitions and background about SAT and SAT solvers, MaxSAT and MaxSAT solvers, about encoding cardinality constraints into (Max)SAT, and we define the Curriculum-based Course Timetabling problem.

In Section 3 we give a first encoding into SAT, where also the soft constraints of the timetabling problem are made hard. Therefore, for those timetabling instances where our Barcelogic SAT solver finds a model, this provides a zero-cost solution. Surprisingly it turns out that this is indeed the case in six of the 32 instances, and in less than 10 seconds.

In Section 4 we give MaxSAT encodings where different soft constraints of the timetabling problems are made soft, and we report on the corresponding experiments.

Section 5 summarizes all our results, and in Sections 6,7 and 8 we discuss related and future work and conclude.

2 Preliminaries

2.1 SAT and SAT Solvers

Let $\mathcal{X}$ be a fixed finite set of propositional variables. If $x \in \mathcal{X}$, then x and $\neg x$ are *literals* of $\mathcal{X}$. The *negation* of a literal l, written $\neg l$, denotes $\neg x$ if l is x, and x if l is $\neg x$. A *clause* is a disjunction of literals $l_1 \vee \ldots \vee l_n$. A (total truth) *assignment* A is a set of literals such that exactly one of $\{x, \neg x\}$ is in A for each x in $\mathcal{X}$. A literal l is *true* in A if $l \in A$ and is *false* in A if $\neg l \in A$. A clause C is true in A (or *satisfied* by A) if at least one of its literals is true in A. An assignment A is a *model* of a set of clauses S if it satisfies all clauses in S. The problem of finding out whether a given clause set S has any model (i.e., is satisfiable) is known as SAT. A (complete) *SAT solver* is a system that, given a clause set S, always terminates returning a model of S or (correctly) reports its unsatisfiability.

Most state-of-the-art SAT solvers (Moskewicz et al, 2001; Goldberg and Novikov, 2002; Een and Sorensson, 2003; Ryan, 2004; Biere, 2008) use *Conflict-driven Clause Learning*, and are originally based on the Davis-Putnam-Logemann-Loveland (DPLL)

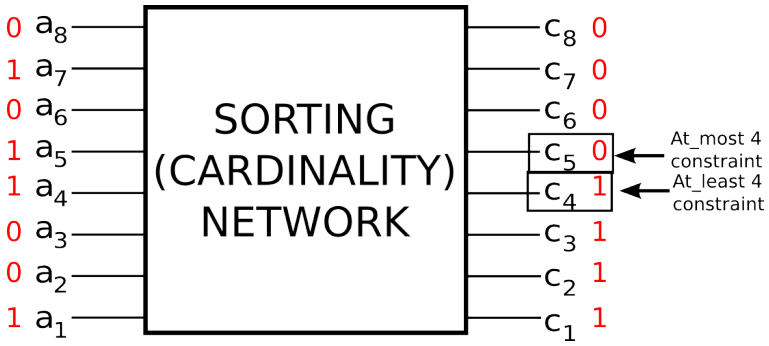


Fig. 1 Sorting network with input $\langle a_1 \dots a_8 \rangle$ and output $\langle c_1 \dots c_8 \rangle$

procedure (Davis and Putnam, 1960; Davis et al, 1962) (see, e.g., (Nieuwenhuis et al, 2006) for details and more references).

2.2 Encoding cardinality constraints into SAT

Although attempts have been made (see e.g., Cadoli and Schaerf (2005)) to define a problem-specification language for automatically generating a SAT encoding, this kind of experiences have shown to be limited in practice. For a given problem many different encodings into propositional clauses may exist, and SAT solvers can behave very differently on each one of them (see Hertel et al (2007)). Specialized encodings for a given problem may perform very well and even become a state-of-the-art technique for the problem, whereas generic ones may not even find a solution. Still, a lot of effort has been put in finding cheap and efficient ways of encoding constraints that appear in many real-world problems.

In particular, this is the case for *cardinality* constraints, saying that (at least, at most or exactly) k of a given set of n literals must be true. The natural straightforward encoding for this kind of constraints is exponential. For example, the following 10 clauses encode that *at most 2* of a set $\{x_1, x_2, x_3, x_4, x_5\}$ of variables are true:

$$\neg x_1 \vee \neg x_2 \vee \neg x_3 \quad \neg x_1 \vee \neg x_2 \vee \neg x_4 \quad \neg x_1 \vee \neg x_2 \vee \neg x_5 \quad \neg x_1 \vee \neg x_3 \vee \neg x_4$$
$$\neg x_1 \vee \neg x_3 \vee \neg x_5 \quad \neg x_1 \vee \neg x_4 \vee \neg x_5 \quad \neg x_2 \vee \neg x_3 \vee \neg x_4 \quad \neg x_2 \vee \neg x_3 \vee \neg x_5$$
$$\neg x_2 \vee \neg x_4 \vee \neg x_5 \quad \neg x_3 \vee \neg x_4 \vee \neg x_5$$

On the other hand, the following ones express *at least 2*:

$$x_1 \vee x_2 \vee x_3 \vee x_4 \quad x_1 \vee x_2 \vee x_3 \vee x_5 \quad x_1 \vee x_2 \vee x_4 \vee x_5$$
$$x_1 \vee x_3 \vee x_4 \vee x_5 \quad x_2 \vee x_3 \vee x_4 \vee x_5$$

The exponential blowup can be avoided by using auxiliary variables. For example, encodings inspired by BDDs , adder networks, and sorting networks have been proposed (Aloul et al, 2002; Bryant, 1986; Bailleux and Boufkhad, 2003).

For the encodings in this paper we have used *Cardinality Networks* (Asín et al, 2009), an improvement over sorting networks, that require $n \log^2 k$ extra-variables and $n \log^2 k$ clauses and have very good propagation properties. The basic idea is that the n input variables are seen as inputs of a circuit (encoded by clauses with auxiliary

variables) that *sorts* them into n output variables, i.e., all 1s of the input come first in the output, and then all 0s. In this way, to express that at least k input variables are true, it suffices to force the kth output variable to 1. Similarly, for "at most k", the $k+1$-th output variable is set to 0 (see figure 1; we refer to Asín et al (2009) and its references for all details, that cannot be included here).

2.3 MaxSAT and core-based MaxSAT solvers

The input to a *Weighted Partial MaxSAT* problem consists of a set of *soft* clauses, each one of them with a real number called its *weight*, and a set of *hard* clauses. The aim is to find a model of the hard clauses that minimizes the sum of the weights of the falsified soft clauses. Here the word "Weighted" is dropped if all soft clauses have the same weight, and the word "Partial" is dropped if there are no hard clauses (note that hard clauses can also be seen as soft clauses with infinite weight).

The state of the art in MaxSAT solvers is still seeing rapid development. The improvements in recent solvers are largely due to new algorithms based on *unsatisfiable cores*. The idea is (very roughly) the following. A (standard) SAT solver can be adapted to return, in case of unsatisfiabilty, a small (or even minimal, wrt. set inclusion) unsatisfiable subset of the initial set of clauses (Zhang and Malik, 2003). Such subsets, called (unsatisfiable) *cores*, are obviously useful (in applications like planning) for generating a small explanation why no feasible solution exists.

For solving (unweighted) MaxSAT, a SAT solver is run on the clause set. If a model is found, then a zero-cost solution has been found. Otherwise, a core is generated, an additional fresh *relaxation* variable is added to each soft clause in the core, and a cardinality constraint is added saying that at most one of the relaxation variables can be true, i.e., at most one of the clauses in the core is allowed to become false. Then the SAT solver is restarted on this extended clause set. This process is iterated until a model is found, which is then provably optimal. Many variants and improvements on this technique exist (that cannot be covered in this paper; see e.g., Fu and Malik (2006); Marques-Silva and Planes (2008); Ansótegui et al (2009); Manquinho et al (2009)). In particular, our new Barcelogic solver we use here for Partial MaxSAT uses a novel concept of *core clusters* to improve the pruning power of the cardinality constraints and never add more than one relaxation variable to any clause.

2.4 The Curriculum-based Course Timetabling Problem

The Curriculum-based Course Timetabling Problem was defined for track 3 of the 2nd International Timetabling Competition (ITC) held in 2007 and its complete description can be found in (Di Gaspero et al, 2007). Briefly, this problem deals with the following objects:

Courses: A *course* is composed of a defined number of lectures on some subject. Each course is associated to one teacher, a number of students and a minimum number of days over which its lectures may be spread. For example, there may exist a 50-student course c_1, taught by teacher t_1, consisting of 5 lectures to be spread over at least 3 distinct days of the week.

Curriculums: A *curriculum* is a set of courses that may share students. For example, curriculum k_1 may consist of four courses: c_1, c_2, c_3 and c_4.

Rooms: Rooms are the spaces in which courses take place. Each room has an associated capacity (number of students, typically from 20 to 1000).
Days and hours: The courses are taught weekly on some day and hour. Each day consists of a specified number of lecture hours in which a course can take place.

The problem is to find an assignment of courses to rooms and hours in such a way that the following *hard* (necessary) and *soft* (desirable) constraints are satisfied.

Hard Constraints:

Curriculum clashes: No two courses belonging to the same curriculum may be scheduled at the same time.
Teacher clashes (Hard): No two courses taught by the same teacher may occur at the same time.
Room clashes: No room must be used for more than one course at the same time.
Hour availability: Teachers may declare themselves as not available for certain hours.
Number of lectures: Exactly the specified number of lectures of every course must be scheduled.

Soft Constraints:

Room capacity: No course may be scheduled to a room with less capacity than the one needed by the course. Each violation of this constraint has a cost of 1 per student that does not fit into the room.
Min working days: The lectures of a given course must be spread over a given minimal number of days. Each day less than the minimum for a course has a cost of 5.
Isolated lectures: Lectures belonging to a curriculum should be adjacent to another lecture of the same curriculum. Each time a lecture belonging to one curriculum is isolated, this violation has cost 2.
Room stability: All lectures of a given course should be scheduled to the same room. Each extra room needed for a course has cost 1.

3 Our Basic SAT encoding

Here we give a first encoding into SAT. All other encodings in this paper are only very slight variants over this (two-page) basic encoding.

In this first encoding also the soft constraints of the timetabling problem are made hard. Therefore, for those timetabling instances where the SAT solver finds a model, this provides a zero-cost solution. Surprisingly it turns out that this is indeed the case in six of the 32 instances, and in less than 10 seconds. In this section we have used our Barcelogic SAT solver, that ranked third in the last SAT Race (2008, Guangzhou, China, see `baldur.iti.uka.de/sat-race-2008`), and first on unsatisfiable problems which is what matters most for optimisation applications.

We define the following propositional variables:

- $ch_{c,h}$: "course c occurs in hour h"
- $cd_{c,d}$: "course c occurs in day d"
- $cr_{c,r}$: "course c occurs in room r"
- $kh_{k,h}$: "curriculum k occurs in hour h"

Once these propositional variables are defined, for the encoding to be correct, that is, admit all correct solutions, and only these ones, we must carefully express every relation between the facts these variables represent.

Relation between ch and cd:

- If some course occurs in hour h, it also occurs in the day corresponding to h. So, for each course c and hour h, the following two-literal clause is needed:

$$\neg ch_{c,h} \vee cd_{c,day(h)}$$

- If some course occurs in a day d, it must also occur in some of the hours of d. So, for each course c and day d consisting of hours $h_1, h_2, \ldots h_n$, the following clause is needed:

$$\neg cd_{c,d} \vee ch_{c,h_1} \vee \ldots \vee ch_{c,h_n}$$

Relation between ch and kh:

- If some course c occurs in hour h, all the curricula $k_1, k_2, \ldots, k_{nk}$, to which c belongs occur in h. So, for each course c, hour h and curricula $k_1, k_2, \ldots, k_n$ that include c, the following clauses are needed:

$$\begin{array}{c} \neg ch_{c,h} \vee kh_{k_1,h} \\ \neg ch_{c,h} \vee kh_{k_2,h} \\ \vdots \\ \neg ch_{c,h} \vee kh_{k_n,h} \end{array}$$

- If some curriculum k occurs in hour h, then at least one of the courses belonging to k must also occur in h. So, for every hour h and curriculum k consisting of courses $c_1, c_2, \ldots, c_n$, the following clauses are needed:

$$\neg ct_{k,h} \vee ch_{c_1,h} \vee \ldots \vee ch_{c_n,h}$$

We now encode the constraints of the timetabling problem. Here we abstract away the concrete encoding of cardinality constraints by simply denoting them by $at\_most(k, S)$, $at\_least(k, S)$ and $exactly(k, S)$, where $k \in \mathbb{N}$ and S is a set of literals. Such an expression represents a set of clauses that are satisfied if, and only if at least (at most, or exactly) k of the nc variables of S are true.

Curriculum clashes: No two courses c and c' belonging to the same curriculum may be scheduled at the same hour. So, for each hour h and for each pair of distinct courses c, c' belonging to the same curriculum, we have:

$$\neg ch_{c,h} \vee \neg ch_{c',h}$$

Teacher clashes: No two courses c and c' with the same teacher may be scheduled to the same hour. So, for each hour h and for each pair of distinct courses c, c' such that $teacher(c) = teacher(c')$, we have:

$$\neg ch_{c,h} \vee \neg ch_{c',h}$$

Room clashes: No two courses c and c' may be scheduled to the same room at the same hour. So, for each room r, hour h, and pair of distinct courses c, c', we need:

$$\neg ch_{c,h} \vee \neg ch_{c',h} \vee \neg cr_{c,r} \vee \neg cr_{c',r}$$

Hour availability: For each course c with forbidden hours $h_1, h_2, \ldots, h_n$, we have the following one-literal clauses:

$$\begin{array}{l} \neg ch_{c,h_1} \\ \neg ch_{c,h_2} \\ \vdots \\ \neg ch_{c,h_n} \end{array}$$

Number of lectures: For each course c exactly $hours(c)$ of the set $ch_{c,h_1}, ch_{c,h_2}, \ldots, ch_{c,h_n}$ must be true:

$$exactly(hours(c), \{ch_{c,h_1}, ch_{c,h_2}, \ldots, ch_{c,h_n}\})$$

Room capacity: Each course must be scheduled to a room in which it fits. So, for each course c with number of students ns and for every room r with capacity cr such that $cr < ns$, we have:

$$\neg cr_{c,r}$$

Min working days: For each course c, at least $working\_days(c)$ literals of the set $cd_{c,d_1}, cd_{c,d_2}, \ldots, cd_{c,d_5}$ should be true:

$$at\_least(working\_days(c), \{cd_{c,d_1}, cd_{c,d_2}, \ldots, cd_{c,d_5}\})$$

Isolated lectures: If some curriculum k occurs in hour h, then k must also occur in an hour before or after in the same day. For each curriculum k:

- For each first hour of a day h:

$$\neg kt_{c,h} \vee kt_{k,h+1}$$

.

- For each last hour of a day h:

$$\neg kt_{c,h} \vee kt_{k,h-1}$$

- For each hour h that is not the first nor the last of a day:

$$\neg kt_{c,h} \vee kt_{k,h-1} \vee kt_{k,h+1}$$

Room stability: Each course must be scheduled to exactly one room. So, if there are rooms $r_1, \ldots, r_n$, for each course c we have[1]:

$$exactly(1, \{cr_{c,r_1}, \ldots, cr_{c,r_n}\})$$

[1] Note that, by including here only the rooms with sufficient capacity, the Room Capacity constraint would get subsumed. Here we have not done this becuase later on the latter constraint will become soft.

Table 1 Solving Times for basic encoding

dataset	vars	clauses	result	time
comp01	12886	62877	-	TO
comp02	110288	575890	UNSAT	2.0
comp03	72749	456172	UNSAT	0.8
comp04	92209	560109	UNSAT	0.6
comp05	34053	179043	UNSAT	0.2
comp06	105728	936770	UNSAT	0.6
comp07	214603	1803516	UNSAT	5.8
comp08	115353	708140	UNSAT	2.3
comp09	105952	603965	UNSAT	0.7
comp10	147673	1211016	UNSAT	3.9
comp11	12537	71919	SAT(=)	1.2
comp12	81659	470203	UNSAT	0.2
comp13	111401	626315	UNSAT	0.7
comp14	113180	675440	UNSAT	0.8
comp15	72749	456172	UNSAT	0.7
comp16	148258	1248224	UNSAT	0.8
comp17	144334	924138	UNSAT	0.6
comp18	40444	174369	UNSAT	0.2
comp19	109236	525817	UNSAT	0.3
comp20	116149	1184210	UNSAT	4.4
comp21	129565	926364	UNSAT	1.0
DDS1	900167	2588788	UNSAT	1.5
DDS2	137688	667470	SAT(=)	1.7
DDS3	60968	305601	SAT(=)	1.1
DDS4	1356276	12842169	UNSAT	25.7
DDS5	556569	3372803	SAT(=)	8.9
DDS6	125029	1001737	**SAT**	10.3
DDS7	124330	612475	SAT(=)	2.4
test1	19406	182328	UNSAT	0.4
test2	34748	213222	UNSAT	0.6
test3	63534	271811	UNSAT	0.2
test4	67539	293208	UNSAT	0.4

3.1 Experiments with the basic SAT encoding

We have tested this encoding over the full set of benchmarks presented in the ITC2007 track3 organizers' web site (`http://tabu.diegm.uniud.it/ctt`). All instances labeled with comp# are competition benchmarks, while the DDS# are bigger ones added after the competition. The test# examples were given for testing. They are small but hard. The web site allows researchers to compare on these benchmarks with the best known results and report new ones that are automatically checked for correctness and cost.

All experiments in this paper are on a 2100Mhz AMD-Opteron with 10000s timeout (TO) and 2GB memory-out (MO). We never include encoding time since it is negligible.

Table 1 lists our results on these benchmarks for the basic SAT encoding: number of variables and clauses, the output SAT/UNSAT, and the time used by our Barcelogic SAT solver. In six instances a model, and thus a zero-cost solution, is found (in 1.1, 1.2, 1.7, 2.4, 8.9, and 10.3 seconds respectively). For five of them, a solution of the same cost (in this case, zero) was already known. In all tables in this paper this is indicated with an = sign. In addition, in very little time (10.3 seconds), we also obtain a new zero-cost solution (DDS6) beating all the reported results over this benchmark.

Table 2 Results for relaxing "isolated lectures" as Partial-MaxSAT

dataset	vars	clauses	cost	barcelogic time	PM2 time
comp01	12886	62877	-	TO	TO
comp02	110288	575890	**24**	3719.0	TO
comp03	72749	456172	-	TO	TO
comp04	92209	560109	36	13.2	60.2
comp05	34053	179043	∞	131.3	89.5
comp06	105728	936770	**28**	540.5	270.7
comp07	214603	1803516	**6**	9625.0	MO
comp08	115353	708140	38	18.1	91.2
comp09	105952	603965	-	TO	TO
comp10	147673	1211016	**4**	145.8	226.6
comp11	12537	71919	0(=)	0.5	2.8
comp12	81659	470203	-	TO	TO
comp13	111401	626315	62	68.8	153.2
comp14	113180	675440	54	111.7	95.3
comp15	72749	456172	-	TO	TO
comp16	148258	1248224	**22**	24.7	84.6
comp17	144334	924138	**60**	700.8	7817.9
comp18	40444	174369	-	TO	TO
comp19	109236	525817	58	173.1	372.4
comp20	116149	1184210	**4**	2305.6	674.4
comp21	129565	926364	**86**	8874.2	TO
DDS1	900167	2588788	∞	1.4	5.3
DDS2	137688	667470	0(=)	0.8	3.0
DDS3	60968	305601	0(=)	0.4	1.9
DDS4	1356276	12842169	-	TO	MO
DDS5	556569	3372803	0(=)	10.4	17.9
DDS6	125029	1001737	0(=)	15.0	14.4
DDS7	124330	612475	0(=)	1.3	4.9
test1	19406	182328	-	TO	MO
test2	34748	213222	16(=)	123.4	59.3
test3	63534	271811	∞	0.3	0.6
test4	67539	293208	-	TO	MO

4 MaxSAT encodings

4.1 Relaxing "isolated lectures" as Partial-MaxSAT

Here we turn the "isolated lectures" constraint into a soft constraint. In the basic SAT encoding, each clause for this constraint encoded exactly one violation of the constraint. So here each one of these clauses that is not satisfied has cost 2: we use a Partial MaxSAT solver that considers these clauses as soft ones, and then each model found represents a solution with as cost twice the number of unsatisfied soft clauses.

Table 2 lists results using our own first partial-maxsat-solver prototype (Barcelogic) and also with PM2 (Ansótegui et al (2009)), the winner of the Industrial Partial MaxSAT track in the latest MaxSAT competition (`www.maxsat.udl.cat/09`). A cost of ∞ indicates that there is no solution if only the isolated lectures constraint is made soft. With respect to the SAT encoding where everything was hard, the number of instances for which we can give a solution grows from 6 to 20. Of the 14 new ones, 8 (indicated in bold) improve the previous best cost reported by the community.

Note that also the 6 zero-cost results of the previous section are obtained with this method and that we also match the best resported result (cost 16) for the test2 instance. In all tables, if the name of the instance is given in bold, this means that our solution is known to be optimal (the website also reports lower bounds).

Table 3 Results for relaxing "min working days" as Weighted-Partial-MaxSAT

dataset	vars	clauses	cost	WPM1 time	MSUNCORE time
comp01	13864	64189	-	TO	TO
comp02	111869	578209	-	MO	TO
comp03	74178	458243	-	MO	TO
comp04	93497	562036	35(=)	7518	TO
comp05	34902	180478	-	MO	MO
comp06	107670	939713	-	MO	TO
comp07	216804	1806892	6(=)	MO	6442
comp08	116948	710560	37(=)	4455	TO
comp09	107448	606143	-	MO	TO
comp10	149513	1213821	4(=)	62	57
comp11	13511	73276	0(=)	1	0.8
comp12	83492	473168	-	MO	TO
comp13	113012	628743	-	MO	TO
comp14	114680	677720	-	TO	TO
comp15	74178	458243	-	MO	TO
comp16	150222	1251187	**18**	615.3	391.6
comp17	146057	926721	-	MO	TO
comp18	41185	175639	-	TO	TO
comp19	110890	528176	-	MO	TO
comp20	118188	1187337	4(=)	325.8	193.3
comp21	131373	929034	-	MO	TO
DDS1	903884	2594511	**48**	645.0	1629.4
DDS2	137850	667646	0(=)	2.6	3.6
DDS3	62282	307644	0(=)	1.7	2.4
DDS4	1360646	12848872	-	MO	MO
DDS5	558714	3376303	0(=)	14.5	22.3
DDS6	126753	1004376	0(=)	22.4	14.5
DDS7	125311	614021	0(=)	4.2	5.0
test1	21022	184475	-	MO	MO
test2	36375	215426	16(=)	27.5	32.3
test3	65415	274319	-	TO	TO
test4	69420	295717	-	TO	TO

4.2 Relaxing "min working days" as Weighted-Partial-MaxSAT

Here we describe a very efficient way of relaxing, in addition to the "isolated lectures" constraint, also the "min working days" constraint. In the basic SAT encoding, we used cardinality networks to encode for each course c the min working days constraint $at\_least(working\_days(c), \{cd_{c,d_1}, cd_{c,d_2}, \ldots, cd_{c,d_5}\})$. For instance, if $working\_days(c)$ is 4, then we set the fourth output out_4 of the network to 1. In order to make this soft, and such that each day less than four has cost 5, we create one soft one-literal clause out_4 with weight 5, and two more weight-5 one-literal clauses for out_3 and out_2.

Table 3 shows results with two state-of-the-art Weighted-Partial MaxSat solvers that did very well in the last MaxSAT competition... *msuncore* (Manquinho et al, 2009) and *WPM1* (Ansótegui et al, 2009) (our Barcelogic solver cannot handle weighted MaxSAT yet). The winner of the industrial division *SAT4J* behaved much worse on these problems. With any of the two solvers, this encoding allows us to again improve the best known cost on two more instances (comp16 and DDS1). We also match the best solutions on two more (comp04 and 08). These solutions are moreover optimal for DDS1, comp04 and comp08.

Table 4 Results for Weighted-Partial-MaxSAT as Partial-MaxSAT

dataset	vars	clauses	cost	PM2 time
comp01	13864	65025	-	TO
comp02	111869	580927	-	TO
comp03	74178	460787	-	TO
comp04	93497	564285	35(=)	111.3
comp05	34902	186066	-	TO
comp06	107670	942715	**27**	4194.4
comp07	216804	1810289	-	MO
comp08	116948	713049	37(=)	158.7
comp09	107448	608874	-	TO
comp10	149513	1216744	4(=)	135.8
comp11	13511	74249	0(=)	1.3
comp12	83492	6479436	-	TO
comp13	113012	631309	59(=)	1867.9
comp14	114680	680244	51(=)	545.5
comp15	74178	460787	-	TO
comp16	150222	1254202	18(=)	143.7
comp17	146057	929587	-	TO
comp18	41185	178063	-	TO
comp19	110890	530718	-	TO
comp20	118188	1190643	4(=)	542.4
comp21	131373	932016	-	TO
DDS1	903884	2603332	48(=)	748.2
DDS2	137850	668420	0(=)	3.1
DDS3	62282	308551	0(=)	2.5
DDS4	1360646	12855538	-	MO
DDS5	558714	3380575	0(=)	16.9
DDS6	126753	1007166	0(=)	25.8
DDS7	125311	616673	0(=)	5.2
test1	21022	185639	-	MO
test2	36375	216742	16(=)	59.3
test3	65415	276191	-	MO
test4	69420	297864	-	TO

4.3 Weighted-Partial-MaxSAT as Partial-MaxSAT

Instead of using Weighted MaxSAT, one can also replicate each soft clause as many times as the cost related with it, and use a solver for unweighted MaxSAT. It turns out that this works very well on these problems because all weights are small. This can be observed in table 4: besides keeping the previous results, we beat one previously improved example's cost (comp06) and achieve another two best reported costs (comp13 and comp14). We did not use our Barcelogic solver because it has no support for replicated clauses.

Table 5 Results summary over the curriculum-based timetabling problem

dataset	Author previously best	Method	previous cost	our cost
comp01	Andrea Schaerf (+ others)	Various	5	-
comp02	Lu And Hao	Tabu Search	29	**24**
comp03	Tomas Muller	Local Search	66	-
comp04	Tomas Muller	Local Search	35	35(=)
comp05	Lu And Hao	Tabu Search	292	-
comp06	Tomas Muller	Local Search	37	**27**
comp07	Tomas Muller	Local Search	7	**6**
comp08	S. Abdullah, H. Turabieh	Other	37	37(=)
comp09	Lu And Hao	Tabu Search	96	-
comp10	Tomas Muller	Local Search	7	**4**
comp11	Andrea Schaerf (+ others)	Various	0	0(=)
comp12	Lu And Hao	Tabu Search	310	-
comp13	Lu And Hao	Tabu Search	59	62
comp14	Gerald Lach	Mathematical Progr.	51	51(=)
comp15	Andrea Schaerf (+ others)	Various	66	-
comp16	Lu And Hao	Tabu Search	23	**18**
comp17	Lu And Hao	Tabu Search	69	**60**
comp18	Lu And Hao	Tabu Search	65	-
comp19	Tomas Muller	Local Search	57	58
comp20	Gerald Lach	Mathematical Progr.	17	**4**
comp21	Tomas Muller	Local Search	89	**86**
DDS1	Gerald Lach	Mathematical Progr.	83	**48**
DDS2	Andrea Schaerf (+ others)	Various	0	0(=)
DDS3	Andrea Schaerf (+ others)	Various	0	0(=)
DDS4	S. Abdullah, H. Turabieh	Evolutionary Comp.	30	-
DDS5	Andrea Schaerf	Tabu Search	0	0(=)
DDS6	Gerald Lach	Mathematical Progr.	4	**0**
DDS7	Andrea Schaerf (+ others)	Various	0	0(=)
test1	Lu And Hao	Tabu Search	224	-
test2	Andrea Schaerf (+ others)	Various	16	16(=)
test3	S. Abdullah, H. Turabieh	Other	67	-
test4	Lu And Hao	Tabu Search	73	-

5 Results summary

Table 5 gives a summary of our achievements in encoding the curriculum-based course timetabling problem into different versions of SAT/MaxSAT. In the column "our cost" all figures in **bold** indicate costs where we have inproved the best known solution (10 of the 32 instances). In the column "previous cost" one can see the previously known best costs, which in some case we improve importantly: for example in DDS1 we improve from cost 83 to 48, which is moreover known to be optimal. For another 9 instances we match the best known solution (8 of them to optimality: again the names of the benchmarks are in bold if the best solution found is known to be optimal).

In the table We also indicate the author of the previously best known costs, the first one in obtaining the result (as it appears on the web site) and add the label (+others) to clarify that others also reached the same results. The column method indicates the technique used to obtain these previously best known results.

Altogher, out of 32 instances, we obtained 19 of the current best known results, 10 of which were improvements over the past known ones.

6 Related work

As it can be seen in Table 5 (see Lü and Hao (2010)) and in the algorithms specifications of the solvers that participated in the two timetabling competitions held until now (see, for example, Kostuch (2004) and Muller (2005)), the timetabling research field has been mostly dominated by local search techniques. In recent years, nevertheless, other techniques like Constraint Programming or Mathematical Programming have been presented with some success. On the other hand, in the specific case of SAT technologies applied to timetabling problems, not much work has been done. In fact, we are not aware of any work on timetabling using MaxSAT.

Using pure SAT some non-competitive (in expressivity and solving time) techniques are described in the two master thesis (Chin-A-Fat, September 2004; Hartog, 2007). The unpublished manuscript (Marić, 2008) deals with non-standard and hence hard-to-compare benchmarks, and uses more naive encodings (e.g., quadratic-size cardinality constraints).

7 Future Work

A lot of room for improvement exists in MaxSAT solvers for this kind of problems. We plan to work on our own solvers, improving our beta Partial MaxSAT and developing a new Weighted version of it. Also, to encourage the MaxSAT community to work on larger and more practically "industrially"-oriented benchmarks, we plan to contribute our encoded instances to the MaxSAT competition.

We also plan to work on new ways for efficiently encoding (and relaxing) some typical constraints in timetabling problems, like "Room allocation" and "Room stability". Furthermore, in instance "comp01" we found that inside timetabling problems, *pigeon-hole-like problems* can appear. As it is well known in the SAT community, these problems are very difficult to handle for SAT Solvers (which is the case of this particular instance). We also plan to search for ways of dealing with this type of constraints, possibly as a *theory* in the SAT Modulo Theories framework (Nieuwenhuis et al, 2006).

8 Conclusions

In this paper we have tackled the curriculum-based course timetabling problem. We have presented several encodings for the problem from pure SAT to Partial and Weighted-Partial MaxSAT. Each encoding has been tried on 32 instances corresponding to those shown in `http://tabu.diegm.uniud.it/ctt/` that belong to the 3rd track of the last International Timetabling Competition. We have tested several current-state-of-the-art SAT an MaxSAT solvers with good results, achieving 19 out of 32 of the current best known results, 10 of which were improvements over the past known ones.

This shows that using SAT and MaxSAT for timetabling is feasible and productive and encourages us to keep working in two main directions: i) to search for better suited MaxSAT solving techniques and ii) to find better encodings for this kind of problems.

Acknowledgements We want to thank Carlos Anstegui for his advise and help providing us access to his PM2 and WPM1 solvers.

References

Aloul FA, Ramani A, Markov IL, Sakallah KA (2002) Generic ilp versus specialized 0-1 ilp: an update. In: Pileggi LT, Kuehlmann A (eds) ICCAD, ACM, pp 450–457

Ansótegui C, Bonet ML, Levy J (2009) Solving (weighted) partial maxsat through satisfiability testing. In: Kullmann (2009), pp 427–440

Asín R, Nieuwenhuis R, Oliveras A, Rodríguez-Carbonell E (2009) Cardinality networks and their applications. In: Kullmann (2009), pp 167–180

Bailleux O, Boufkhad Y (2003) Efficient cnf encoding of boolean cardinality constraints. In: Rossi F (ed) CP, Springer, Lecture Notes in Computer Science, vol 2833, pp 108–122

Biere A (2008) PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation Submitted

Bryant RE (1986) Graph-based algorithms for boolean function manipulation. IEEE Trans Comput 35(8):677–691

Cadoli M, Schaerf A (2005) Compiling problem specifications into SAT. Artificial Intelligence 162(1-2):89–120

Chin-A-Fat K (September 2004) School timetabling using satisfiability solvers. Master's thesis, Technical University Delft, The Netherlands

Davis M, Putnam H (1960) A computing procedure for quantification theory. Journal of the ACM 7:201–215

Davis M, Logemann G, Loveland D (1962) A machine program for theorem-proving. Comm of the ACM 5(7):394–397

Di Gaspero L, McCollum B, Schaerf A (2007) The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Tech. rep., University of Udine

Een N, Sorensson N (2003) An extensible sat-solver. In: Proceedings of the Sixth International Conference on Theroy and Applications of Satisfiability Testing (SAT), pp 501–518

Fu Z, Malik S (2006) On solving the partial max-sat problem. In: Theory and Applications of Satisfiability Testing, SAT, vol LNCS 4121, pp 252–265

Goldberg E, Novikov Y (2002) BerkMin: A fast and robust SAT-solver. In: Design, Automation, and Test in Europe (DATE '02), pp 142–149

Hartog J (2007) Timetabling on dutch high schools: Satisfiability versus gp-untis. Master's thesis, Technical University Delft, The Netherlands

Hertel A, Hertel P, Urquhart A (2007) Formalizing dangerous sat encodings. In: Marques-Silva J, Sakallah KA (eds) SAT, Springer, Lecture Notes in Computer Science, vol 4501, pp 159–172

Kostuch P (2004) The university course timetabling problem with a three-phase approach. In: Burke EK, Trick MA (eds) PATAT, Springer, Lecture Notes in Computer Science, vol 3616, pp 109–125

Kullmann O (ed) (2009) Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings, Lecture Notes in Computer Science, vol 5584, Springer

Lü Z, Hao JK (2010) Adaptive tabu search for course timetabling. European Journal of Operational Research 200(1):235–244

Manquinho VM, Silva JPM, Planes J (2009) Algorithms for weighted boolean optimization. In: Kullmann (2009), pp 495–508

Marić F (2008) Timetabling based on sat encoding: a case study, faculty of Mathematics, University of Belgrade, Serbia

Marques-Silva J, Planes J (2008) Algorithms for maximum satisfiability using unsatisfiable cores. In: Proceedings of Design, Automation and Test in Europe (DATE 08), pp –

Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: Engineering an Efficient SAT Solver. In: Proc. 38th Design Automation Conference (DAC'01)

Muller T (2005) Constraint-based Timetabling. PhD thesis, PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2005

Nieuwenhuis R, Oliveras A, Tinelli C (2006) Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). Journal of the ACM 53(6):937–977

Ryan L (2004) Efficient Algorithms for Clause-Learning SAT Solvers. Master's thesis, School of Computing Science, Simon Fraser University

Zhang L, Malik S (2003) Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. In: 2003 Design, Automation and Test in Europe Conference (DATE 2003), IEEE Computer Society, pp 10,880–10,885

A Combination of Metaheuristic Components based on Harmony Search for The Uncapacitated Examination Timetabling

Mohammed Azmi Al-Betar · Ahamad Tajudin Khader · J. Joshua Thomas

Abstract In this paper, we investigate the effectiveness of combining the key components of the basic metaheuristic methods on the quality of solutions produced for Uncapacitated Examination Timetabling Problem (UETP). These components are *recombination*, *randomness*, and *neighbourhood structures*. The Harmony search Algorithm (HSA) is used to simulate different combinations of these components. It has three main components: Memory Consideration analogous to recombination, Random Consideration analogous to randomness and Pitch Adjustment analogous to neighbourhood structures. The combinations among metaheuristic components are evaluated using 17 different scenarios each of which reflects a combination of one, two or three components. The results show that the system that combines the three components (recombination, randomness, and neighbourhood structures) provides the best results. Furthermore, the best results obtained from the convergence scenarios were compared with 22 other methods that used a *de facto* dataset defined by Carter et al. (1996) for UETP. The results excel those produced by the previous methods in 2 out 12 datasets.

Keywords Examination Timetabling · Harmony Search Algorithm · Metaheuristic-based methods · Exploration · Exploitation

1 Introduction

Problem Background. Examination timetabling is a taxing administrative task that is often repeated in academic institutions every course session. It is the process of assigning a set of exams, each taken by a set of students, to a set of timeslots (and rooms) according to a set of constraints. Two classes of constraints appeared in the literature: *hard* and *soft*. The hard constraints must be satisfied to obtain a *feasible* solution while soft constraints are desired but not essential. Although soft constraints can be violated, the *quality* of solution is often evaluated against soft constraints ful-

Mohammed Azmi Al-Betar (✉) · Ahamad Tajudin Khader · J. Joshua Thomas
School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia
E-mail: mohbetar@cs.usm.my, tajudin@cs.usm.my, joshopever@yahoo.com

fillment. The basic objective is to obtain a feasible solution with the least number of soft constraint violations.

In computing terms, examination timetabling is a hard combinatorial optimisation problem which belongs to an NP-hard class for most of its variations. This problem normally has a huge and rugged search space with considerable local optimal solutions (Ochoa et al. 2009). This makes it hard to lend itself to be tackled using classical methods.

Previous Methods. An examination timetabling problem may be capacitated or uncapacitated (Pillay and Banzhaf 2009). Uncapacitated Examination Timetabling Problem (UETP) is addressed in this paper. Over the last five decades, the Artificial Intelligence and Operational Research communities have been developing a wide variety of *approximation* methods to tackle UETP. An extensive and exhaustive summary of these methods has been provided by Qu et al. (2009b). Earlier developments were based on graph coloring heuristic methods that assigned exams to timeslots, one by one, based on a difficulty level. A backtracking method is often used with these methods as a recovery approach to timetable with unscheduled exams. The main research of UETP was initiated by Carter et al. (1996), who employed several graph coloring heuristic methods to UETP. Other investigations employing graph coloring heuristic methods for UETP include Burke and Newall (2004); Asmuni et al. (2005, 2009).

One of the most notable achievements for solving UETP has been the emergence of metaheuristic-based methods. Metaheuristic-based methods are classified into local search-based and population-based methods (Blum and Roli 2003). Local search-based methods (e.g., Hill climbing, Simulated annealing, Tabu Search) consider one solution at a time. The solution iteratively undergoes changes guided by an objective function until a stationary point near the initial solution is reached. Several local search-based methods that are tailored to UETP had been reported (Di Gaspero and Schaerf 2002; Di Gaspero 2002; Paquete and Stutzle 2003; Burke and Newall 2003; Casey and Thompson 2003; Merlot et al. 2003; Burke et al. 2004; Yang and Petrovic 2005; Burke et al. 2006; Abdullah et al. 2007). On the other hand, population-based methods (e.g., Genetic Algorithm, Ant Colony Optimisation, Artificial Immune System, Harmony Search Algorithm) consider a population of random solutions at a time. The characteristics of the current population are iteratively recombined to generate a new one. Some population-based methods tailored to UETP has been reported (Cote et al. 2005; Eley 2007).

Local search-based methods are able to explore the search space and find a local optimal solution more structurally, precisely and quickly than population-based methods. However, they go through a trajectory without doing a wider scan of the entire search space (Blum and Roli 2003). On the other hand, population-based methods are able to explore several search space regions at the same time. However, they are unable to find a precise local optimal solution to which they can converge (Fesanghary et al. 2008).

Hyper-heuristic methods have also been proposed for UETP. Normally, they have a high-level heuristic to select from a set of low-level heuristics. Several applications of hyper-heuristic for UETP have been reported (Kendall and Hussin 2005; Burke et al. 2007; Qu and Burke 2009; Pillay and Banzhaf 2009; Qu et al. 2009a).

The best solutions obtained for UETP were provided by metaheuristic-based methods, more precisely, by the hybrid metaheuristics (see Qu et al. 2009b). However, in-depth investigation into the main components of these methods that lead to these

Table 1 Examples of intrinsic components of the basic metaheuristic-based methods

Metaheuristic method	Neighborhood structures	Recombination	Randomness
Genetic Algorithm	–	crossover	mutation
Memetic Algorithm	Hill-Climbing optimizer	crossover	mutation
Harmony Search Algorithm	pitch adjustment	memory consideration	random considration
Hill Climbing	move, swap, exchange	–	–
Simulated Annealing	move, swap, kempe-chain	–	cooling schedule
Tabu Search	shake, kicker, s-chain	–	short term memory & aspiration criterion
Large Neighbourhood Structure	move, swap, cyclic-exchange, etc.	–	–

successful outcomes is still lacking. In this paper, a preliminary investigation of the combinations of the basic metaheuristic components,which includes recombination, randomness, and neighbourhood structures are studied.

Metaheuristic Components. The common component among local search-based methods has been the *neighborhood structures* which are able to explore the search space using one or more local changes in the current solution. On the other hand, the common component among population-based methods has been the *recombination*. Recombination exploits the characteristics of the current population in the process of producing a new population. Both local search-based and population-based methods may have a *randomness* component to diversify the search when and if necessary. Some examples of the three metaheuristic components are provided in Table 1. More metaheuristic components are provided by (Blum and Roli 2003).

The key research issue in applying metaheuristic or hybrid metaheuristic method to any combinatorial optimisation problem is to strike a balance between *exploration* and *exploitation* during the search (Qu et al. 2009b). Note that, during the *exploration* stage, the search is encouraged to explore the not-yet-visited search space regions when necessary. While during the *exploitation* stage, the search concentrates on the the already-visited search space regions (Blum and Roli 2003). To put it simply, exploration comes from an unguided search while exploitation comes from a guided search carried out by an objective function of the current solution(s). However, in order to a establish balance between exploration and exploitation, parameter settings (tuning or adaptation) have to be studied. Naturally, the parameter values guide the components of metaheuristics and can be classified into 'exploration consideration' components and 'exploitation consideration' components:

– ***Exploration consideration***. The components that are concerned with exploration rather than exploitation during the search (for example, *mutation* component in Genetic Algorithm).

– ***Exploitation consideration***. The components that are concerned with exploitation rather than exploration during the search (for example, *neighbourhood structures* guided by objective function in Hill climbing and *crossover* guided by the objective functions of the current population in Genetic Algorithm).

To elaborate, we can classify the metaheuristic components based on the source of improvement into the following three types:

1. **Local improvements**: The improvements that result from the *local* changes on the current solution. The main components that are responsible for this type of improvement is *neighbourhood structures*.
2. **Global improvements**: The improvements that *globally* result from recombining the characteristics of the current solutions. The main component responsible for this type of improvement is *recombination*.
3. **Random improvements**: The improvements that *randomly* result from exploring the search space using an unguided strategy. The main component responsible for this type of improvement is *randomness*.

Harmony Search Algorithm. In this study, Harmony Search Algorithm (HSA) is tailored to investigate the effectiveness of combining the metaheuristic components. HSA is a new metaheuristic population-based method inspired by the musical improvisation process (Geem et al. 2001). It has been successfully applied to a wide variety of optimisation problems (Ingram and Zhang 2009) including University Course Timetabling Problem (UCTP) (Al-Betar et al. 2008; Al-Betar and Khader 2009; Al-Betar et al. 2010). HSA is an iterative improvement method initiated with a number of provisional solutions stored in the 'Harmony Memory (HM)'. At each iteration, a new solution called 'new harmony' is generated based on three components: (i) 'Memory Consideration' which makes use of the characteristics of the solutions in HM; (ii) 'Random Consideration' which is used to diversify the new harmony, and (iii) 'Pitch Adjustment', analogous to neighbourhood structures [1]. A new harmony is then evaluated using an objective function and it replaces the worst harmony stored in HM. This process is repeated until a stop criterion is met.

Paper Contributions. The objectives of this study are as follows: (i) tailoring HSA for the UETP, (ii) investigating the effectiveness of combining the meta-heuristic components for producing high quality solutions to UETP.

Results. The HSA results are compared with other results produced by 22 published methods using a *de facto* standard datasets defined by Carter et al. (1996). Although other datasets exist (See *de jure* standard datasets defined in ITC-2007 [2] (McCollum et al. 2009)), the Carter dataset provides a suitable wide range of comparative methods which the proposed method can be evaluated against.

Paper Organization. In order to present a self-explanatory paper, The UETP is described in section. 2 . The way of tailoring HSA toward UETP is proposed in section. 3. A comparative evaluation and empirical study of combining meta-heuristic components are presented in section. 4. The paper concludes with possible research directions described in section. 5.

[1] neighbourhood structures refer to the *move* operators in local-search based methods, such as, move one exam from timeslot to another, swap the timesolts of two exams, etc.

[2] Second International Timetabling Competition (http://www.cs.qub.ac.uk/itc2007/)

2 Problem description

2.1 Problem definition

The UETP variation is concerned with assigning a set of exams, each taken by a set of students, to a set of timeslots with respect to hard (H1) and soft constraint (S1).

- **H1: Exam clash.** No student can sit for two exams at the same time.
- **S1: Exams spread out.** The exams taken by the same student should be spread out across a timetable.

In UETP, the main objective is to minimise the proximity cost function of soft constraint violations in a feasible timetable. The proximity cost function divides the penalty of soft constraint violations by the total number of students. This function will be described formally in the next section.

2.2 Problem formulation

The notation for UETP formulation is given in Table 2. A timetable solution is represented by a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$ of exams, where x_i is timeslot, $t \in \mathcal{T}$, for exam, $i \in \mathcal{E}$.

The proximity cost function f($\boldsymbol{x}$) for The UETP is formulated in Eq.(1)

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

Where $c_{i,j}$ element contains the total number of students sharing exam i and exam j, $a_{i,j}$ element contains the penalty value made based on the distance between exam i and exam j. This provides the *quality* of a solution in terms of how well the exams are spread. Note that the hard constraint H1 must be satisfied in the timetable $\boldsymbol{x}$ such that

$$x_i \neq x_j \qquad \forall x_i, x_j \in \boldsymbol{x} \wedge c_{i,j} \geq 1$$

The value of the proximity cost function $f(x)$ is referred to as the Penalty Value (PV) of a feasible timetable.

3 Proposed Method

The concepts of Harmony Search Algorithm (HSA) are described within the context of creating a pleasing harmony within a musical context (Lee and Geem 2004, 2005; Lee et al. 2005). Table 3 shows the relationship between the UETP terms and optimisation terms in the musical context. In musical improvisation, a group of musicians improvise the pitches of their musical instruments. From repeated practice sessions, a pleasing harmony as decided by their own audio-aesthetic standard is sought. Similarly, in the optimisation context, a set of decision variables is assigned with values. From repeated iterations, an optimal solution as decided by an objective function is sought. In the timetabling process, a set of exams is scheduled with timeslots. From repeated

Table 2 Notations used to formalise the UETP

Symbol	Description
N	The number of exams.
P	The number of timeslots.
M	The number of students.
$\mathcal{E}$	Set of exams, $\mathcal{E} = \{1, 2, \ldots, N\}$.
$\mathcal{S}$	Set of students, $\mathcal{S} = \{1, 2, \ldots, M\}$.
$\mathcal{T}$	Set of timeslots, $\mathcal{T} = \{1, 2, \ldots, P\}$.
$\boldsymbol{x}$	A timetable is represented by $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$.
x_i	the timeslot of exam i.
$c_{i,j}$	Conflict matrix element: total number of students sharing exam i and exam j. $$c_{i,j} = \sum_{k=1}^{M} u_{k,i} \times u_{k,j} \quad \forall i, j \in \mathcal{E}$$
$a_{i,j}$	Proximity coefficient matrix element: whether the timetable $\boldsymbol{x}$ is penalized based on the distance between timeslot of exam i in timeslot of exam j. $$a_{i,j} = \begin{cases} 2^{5-\lvert x_i - x_j \rvert} & \text{if } 1 \leq \lvert x_i - x_j \rvert \leq 5. \\ 0 & \text{otherwise.} \end{cases}$$
$u_{i,j}$	Student-exam matrix element: whether student s_i is sitting for exam j $$u_{i,j} = \begin{cases} 1 & \text{if student } i \text{ sitting in exam } j \\ 0 & \text{otherwise.} \end{cases}$$

Table 3 The UETP and Optimisation terms in the musical context

Musical		Optimisation		UETP
Improvisation	$\leftrightarrow$	Generation	$\leftrightarrow$	Scheduling
Harmony	$\leftrightarrow$	Solution vector	$\leftrightarrow$	Timetabling solution
Musician	$\leftrightarrow$	Decision variable	$\leftrightarrow$	Exam
Pitch	$\leftrightarrow$	Value	$\leftrightarrow$	Timeslot
Pitch Range	$\leftrightarrow$	Value Range	$\leftrightarrow$	Feasible timeslots
Esthetic standard	$\leftrightarrow$	Objective function	$\leftrightarrow$	Proximity cost function
Practice	$\leftrightarrow$	Iteration	$\leftrightarrow$	Iteration
Pleasing harmony	$\leftrightarrow$	Optimal solution	$\leftrightarrow$	Feasible timetable with the least number of soft constraint violations

iterations, a feasible timetable with the least weight of soft constraint violations as decided by a proximity cost function is sought.

Algorithm 1 shows the pseudo-code of the HSA applied for UETP with five main steps that will be described below:

Step 1. Initialize the problem and HSA parameters.

The solution is represented by a vector, $\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_N)$, of exams. The value of each exam x_i is a timeslot. The possible range of each exam is the possible feasible timeslots. The proximity cost function is utilized in HSA as formalised in Eq.(1).

Algorithm 1 The basic harmony search algorithm

STEP1 Initialize the problem and HSA parameters

1: Input data instance of the UETP.
2: Utilize UETP pacific knowledge: objective function and solution representation.
3: Set the HSA parameters (HMCR, PAR, NI, HMS).

STEP2 Initialise the harmony memory

1: Construct feasible timetables based on Saturation Degree (SD) stored in harmony memory, $\mathbf{HM} = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^{\mathrm{HMS}}\}$
2: Recognise the worst vector in **HM**,
$\boldsymbol{x}^{\mathrm{worst}} \in \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^{\mathrm{HMS}}\}$ where $f(\boldsymbol{x}^{\mathrm{worst}}) \geq f(\boldsymbol{x}^j) \wedge j \in \{1, \ldots, \mathrm{HMS}\}$

STEP3 Improvise a new harmony

1: $\boldsymbol{x}' = \phi$ // new harmony vector
2: **for** $J = 1, \cdots, N$ **do**
3: $i \leftarrow$ Saturation-Degree ($\boldsymbol{x}'$)
4: **if** $(U(0,1) \leq \mathrm{HMCR})$ **then**
5: $x'_i \in \mathcal{Q}_i$ $\{\mathcal{Q}_i = \{x_i^j | t_{i,x_i^j} = 1 \wedge j \in 1, \ldots, \mathrm{HMS}\}\}$
6: **if** $(\mathcal{Q}_i = \phi)$ **then**
7: $x'_i \in \mathcal{X}_i$ $\{$ $\mathcal{X}_i = \{d | t_{i,d} = 1 \wedge d \in 1, \ldots, P\}$ $\}$
8: **if** $(\mathcal{X}_i = \phi)$ **then**
9: GOTO 1 { Restart }
10: **end if**
11: **end if**
12: $p \leftarrow U(0,1)$ $\{U(0,1)$ Uniform generator number between 0 and 1$\}$
13: **if** $(p \leq \mathrm{PAR1})$ **then**
14: Pitch adjustment Single-move (x'_i)
15: **else if** $(p \leq \mathrm{PAR2})$ **then**
16: Pitch adjustment Swap-timeslot (x'_i)
17: **else if** $(p \leq \mathrm{PAR3})$ **then**
18: Pitch adjustment Kempe-chain (x'_i)
19: **end if**
20: **else**
21: $x'_i \in \mathcal{X}_i$ $\{$ $\mathcal{X}_i = \{d | t_{i,d} = 1 \wedge d \in 1, \ldots, P\}$ $\}$
22: **if** $(\mathcal{X}_i = \phi)$ **then**
23: GOTO 1 { Restart }
24: **end if**
25: **end if**
26: **end for**

STEP4 Update the harmony memory

1: **if** $(f(\boldsymbol{x}') < f(\boldsymbol{x}^{\mathrm{worst}}))$ **then**
2: Include $\boldsymbol{x}'$ to the **HM**.
3: Exclude $\boldsymbol{x}^{\mathrm{worst}}$ from **HM**.
4: **end if**

STEP5 Check the stop criterion

1: **while** (not termination criterion is specified by NI) **do**
2: Repeat **STEP3** and **STEP4**
3: **end while**

The parameters of the HSA required to solve the UETP are also set in this step:

1. The Harmony Memory Consideration Rate (HMCR), used in the improvisation process to determine whether the value of a decision variable is to be selected from the solutions stored in the Harmony Memory (HM).
2. The Harmony Memory Size (HMS) is similar to the population size in Genetic Algorithm.

3. The Pitch Adjustment Rate (PAR), decides whether the decision variables are to be adjusted to a neighbouring value.
4. The Number of Improvisations (NI) corresponds to the number of iterations.

These parameters will be explained in more detail in the next steps.

Step 2. Initialize the harmony memory.

The harmony memory (HM) is an augmented matrix of size $N \times \text{HMS}$ which contains sets of solution vectors determined by HMS (see Eq.2). In this study, the feasible search space regions is only explored where the HM is initialized with random feasible timetables using Saturation Degree (SD) (Brélaz 1979). The SD was widely used to construct an initial solution for UETP. In SD, the exam that has the least number of valid timeslots in the partial timetable is timetabled first.

$$\text{HM} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{\text{HMS}} & x_2^{\text{HMS}} & \cdots & x_N^{\text{HMS}} \end{bmatrix} \tag{2}$$

Algorithm 2 provides a high level schematic pseudo-code of building HM solution using SD. Note that the objective function of each timetable in HM is calculated. The solutions in HM are sorted in ascending order in terms of their objective function values, such as, $f(\boldsymbol{x}^1) \leq f(\boldsymbol{x}^2) \leq \ldots \leq f(\boldsymbol{x}^{\text{HMS}})$.

Algorithm 2 Schematic pseudo-code of building HM solutions

```
for j = 1, ··· , HMS do
    x^j = φ
    k = 1
    while (k < N) do
        i= SaturationDegree(x^i)
        x_i^j = h {h ∈ 1 ... P ∧ h is feasible for x_i^j}
    end while
    calculate f(x^j)
    store x^j in the HM
end for
```

Step 3. Improvise a new harmony.

This is the main step in HSA for iterating towards an optimal solution in which this process called 'improvisation process'. In this step, the HSA will construct (or *improvise*) a *new harmony* vector (timetabling solution) from scratch, $\boldsymbol{x}' = (x'_1, x'_2, \cdots, x'_N)$, based on three operators (or components): (i) memory consideration, (ii) random consideration, and (iii) pitch adjustment.

In the timetabling domain, the process of constructing a feasible timetable (in our case *new harmony*), $\boldsymbol{x}' = (x'_1, x'_2, \cdots, x'_N)$, from scratch often requires an ordering

mechanism (Asmuni et al. 2009). As such, to preserve the feasibility during the improvisation process, the idea of Saturation Degree (SD) has been adopted for ordering the exams as follows: exam i that has the least feasible timeslots to be timetabled in the new harmony is selected first.

Formally let Trajectory matrix ($\mathbf{T}$) of size $N \times P$ contain binary elements, i.e., $(t_{i,j})$, which are assigned as follows:

$$t_{i,j} \leftarrow \begin{cases} 1 & \text{if exam } i \text{ can be feasibly assigned with timeslot } j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Let $\wp_k = \sum_{j=1}^{P} t_{k,j}$ be the total number of available timeslots for each exam k to be timetabled in $\boldsymbol{x}'$. SD iteratively selects exam i to be assigned in $\boldsymbol{x}'$ where:

$$i = \arg \min_{k=1 \ldots N} \wp_k \tag{4}$$

Definition 1 *Exam i can be feasibly assigned with timeslot j if and only if $x'_k \neq j \quad , \forall x'_k \in \boldsymbol{x}' \wedge c_{i,k} \neq 0 \wedge k \in \mathcal{E}$.*

Memory consideration. Every exam i, selected by SD to be assigned with a timeslot x'_i, selects a feasible timeslot from corresponding timeslots, $x'_i \in \{x_i^1, x_i^2, \ldots, x_i^{HMS}\}$, stored in HM vectors with probability (w.p.) HMCR where $0 \leq \text{HMCR} \leq 1$. The operation of this operator is similar to the recombination operator in other population-based methods and is a good source of exploitation (Yang 2009).

Formally, let exam i be selected by SD to be timetabled with timeslot x'_i, let set $\mathcal{Q}_i = \{x_i^j | t_{i,x_i^j} = 1 \wedge j \in 1, \ldots, \text{HMS}\}$ contain the feasible timeslots available for exam i in HM solutions. The timeslot x'_i of exam i will be randomly selected from $\mathcal{Q}_i$ with probability HMCR. In case the $\mathcal{Q}_i = \phi$ which means no feasible timeslot stored in the HM vectors for x'_i, the '**Exceptional random consideration (ERC)**' will run. In ERC, the random consideration operator described below attempts to assign the exam i with timeslot x'_i, as shown Algorithm 1, STEP 3, Line 7.

Random consideration. Exams that are not assigned with timeslots according to memory consideration are randomly assigned according to their available timeslots by random consideration with a probability of (1-HMCR). Formally, let exam i be selected by SD to be assigned with a timeslot x'_i, let set $\mathcal{X}_i = \{d | t_{i,d} = 1 \wedge d \in 1, \ldots, P\}$ contain all feasible timeslots for exam i. The timeslot x'_i of exam i will be randomly selected from $\mathcal{X}_i$ with probability (1-HMCR).

However, in case $\mathcal{X}_i = \phi$, the improvisation process will restart (Henceforth called **restart process**), see Algorithm 1, STEP 3, Lines 9 and 23. In summary see Eq.(5).

$$x'_i \leftarrow \begin{cases} x'_i \in \mathcal{Q}_i & \text{w.p.} \quad \text{HMCR} \\ x'_i \in \mathcal{X}_i & \text{w.p.} \quad \text{(1 - HMCR)} \end{cases} \tag{5}$$

Random consideration is functionally similar to the mutation operator in Genetic Algorithm which is a source of exploration in HSA (Yang 2009). The HMCR parameter is the probability of assigning one timeslot x'_i of exam i, based on historical timeslots stored in the HM solutions.

Pitch adjustment. Every exam i assigned with a timeslot x'_i in the new harmony vector, $\boldsymbol{x}' = (x'_1, x'_2, x'_3, \ldots, x'_N)$, from memory consideration is pitch adjusted with the probability of PAR ($0 \leq \text{PAR} \leq 1$) as follows:

$$\text{Pitch adjust for } x_i'? \leftarrow \begin{cases} \text{Yes} & \text{w.p.} \quad \text{PAR} \\ \text{No} & \text{w.p.} \quad \text{(1-PAR)} \end{cases} \tag{6}$$

If PAR = 0.10, this means that the HSA modifies the existing timeslot of each exam assigned by memory consideration with a probability of (PAR × HMCR), while the timeslot with probability (HMCR × (1 − PAR)) does not change.

For UETP, the pitch adjustment is a neighbourhood move based on 3 neighborhoods, each of which is selected with equal probability as follows:

$$\text{Adjust } x_i' \leftarrow \begin{cases} \text{Single-move} & 0 \leq p \leq \text{PAR1} \\ \text{Swap-timeslot} & \text{PAR1} < p \leq \text{PAR2} \\ \text{Kempe-chain} & \text{PAR2} < p \leq \text{PAR3} \\ \text{do nothing} & \text{otherwise.} \end{cases} \tag{7}$$

Where PAR1=PAR/3 and PAR1:PAR2:PAR3 is in the ratio of 1:2:3, and $p \leftarrow U(0,1)$ is a uniform distribution function which generates a random number between 0 and 1. For each exam i timetabled with x_i' based on memory consideration operator, x_i' is adjusted as follows:

- Pitch adjustment: **Single-move**. With probability range $[0, \text{PAR1}]$, replace the timeslot x_i' of exam i by another feasible timeslot. Although this process is similar to random consideration, it is guided by the objective function of the new harmony.
- Pitch adjustment: **Swap-timeslot**. With probability range of $(\text{PAR1}, \text{PAR2}]$,exam i and exam j swap their timeslots (x_i', x_j') while the feasibility is preserved.
- Pitch adjustment: **Kempe-chain**. With probability range $(\text{PAR2}, \text{PAR3}]$, x_i' is adjusted as follows: (i) Select the timeslot x_i' of exam i and randomly select another timeslot p'. (ii) All exams that have the same timeslot x_i' and conflict with one or more exams timetabled in p' are entered to Chain $\mathcal{G}$ where $\mathcal{G} = \{j | x_j' = x_i' \wedge t_{j,p'} = 0 \wedge \forall j \in \mathcal{E}\}$ (iii) All exams that have the same timeslot p' and conflict with one or more exams timetabled in x_i' are entered to Chain $\mathcal{G}'$ where $\mathcal{G}' = \{k | x_k' = p' \wedge t_{k,x_i'} = 0 \wedge \forall k \in \mathcal{E}\}$, (iv) simply assign the exams in $\mathcal{G}$ with p' and the exams in $\mathcal{G}'$ with x_i'.

In the original HSA proposed by Lee and Geem (2004), the pitch adjustment is unguided by the objective function which can be considered a good source of exploration (Yang 2009). For the purpose of this paper, this component has been modified to be guided by the objective function as follows: the adjustment performed by any pitch adjustment procedure in Eq (7) is accepted in the new harmony if the objective function is not negatively affected, i.e., $f(\boldsymbol{x}'') \leq f(\boldsymbol{x}')$ where $\boldsymbol{x}'$ is the new harmony before the pitch adjustment while $\boldsymbol{x}''$ is the new harmony after it.

Step 4. Update the harmony memory.

If the new harmony vector, $\boldsymbol{x}' = (x_1', x_2', \cdots, x_N')$, is better than the worst harmony vector in HM, the worst harmony vector is substituted with the new harmony vector .

Table 4 Characteristics of the Cater dataset

Datasets Key	Institution	timeslots	exams	students	Density
CAR-S-91	Carleton University, Ottawa	35	682	16925	0.13
CAR-F-92	Carleton University, Ottawa	32	543	18419	0.14
EAR-F-83	Earl Haig Collegiate Institute, Toronto	24	190	1125	0.27
HEC-S-92	Ecole des Hautes Etudes Commercials, Montral	18	81	2823	0.42
KFU-S-93	King Fahd University, Dharan	20	461	5349	0.06
LSE-F-91	London School of Economics	18	381	2726	0.06
RYE-S-93	Ryeson University, Toronto	23	481	11,483	40.07
STA-F-83	St.Andrew's Junior High School, Toronto	13	139	611	0.14
TRE-S-92	Trent University Peterborough, Ontario	23	261	4360	0.18
UTA-S-92	Faculty of Arts and Sciences, University of Toronto	35	622	21266	0.13
UTE-S-92	Faculty of Engineering, University of Toronto	10	184	2750	0.08
YOR-F-83	York Mills Collegiate Institute, Toronto	21	181	941	0.29

Step 5. Check the stop criterion.

Step 3 and step 4 of HSA are repeated until the termination criterion (maximum number of improvisations) is met. This is specified by the NI parameter.

4 Computational experiments

The proposed method is programmed in Microsoft Visual C++ version 6 under Windows XP. The experiments presented here ran on 16 heterogeneous computers with different CPU and RAM capability over 15 days. Note that the total number of experiments is 2040 (17 scenarios × 12 datasets × 10 runs). Thus the computational time has been neglected. As proposed by Qu et al. (2009b); Abdullah et al. (2007), time is not a major constraint as the timetabling problem does not require realtime solutions.

The proposed method is evaluated against *de facto* dataset released by (Carter et al. 1996) which were freely available[3]. Carter's dataset comprises 12 datasets which varies in size (number of exams, number of timeslots) and complexity . The characteristics of Carter dataset are shown in Table 4. The conflict matrix density in the last column refers to the ratio between the number of elements of value $c_{i,j} > 0$ and the total number of elements in the conflict matrix (Qu et al. 2009b). The main objective is to find a conflict-free timetable with the least number of soft constraint violations. The proximity cost function (see Eq.(1)) is used to calculate the Penalty Value (PV) for each timetable obtained. Note that there are two versions of some of Carter dataset (Qu et al. 2009b) and the annotation 'I' refers to which version is used (Pillay and Banzhaf 2009).

[3] http://www.asap.cs.nott.ac.uk/resources/data.shtml

Table 5 Convergence scenarios designed to simulate combinations among metaheuristic components

HMS	HMCR	PAR	Scenario No.	Source of improvement
50	100%	0%	1	GIM
		3%	2	GIM+LIM
		30%	3	GIM+LIM
	98%	0%	4	GIM+RIM
		3%	5	GIM+LIM+RIM
		30%	6	GIM+LIM+RIM
10	100%	3%	7	GIM+LIM
		30%	8	GIM+LIM
	98%	0%	9	GIM+RIM
		3%	10	GIM+LIM+RIM
		30%	11	GIM+LIM+RIM
1	100%	0%	12	No Improvement
		30%	13	LIM
	98%	0%	14	RIM
		30%	15	LIM+RIM
	75%	3%	16	RIM+LIM
		30%	17	RIM+LIM

4.1 Empirical study in Combination of Meta-heuristic components based on HSA

4.1.1 Experimental design

An empirical study of combining the metaheuristic components was conducted using 17 convergence scenarios, each of which varies in parameter settings as shown in Table 5. Each convergence scenario simulates one case of component combinations. The three components are: (i) *recombination* represented by memory consideration which is the source of Global IMprovement (GIM), (ii) the *randomness* is represented by random consideration which is the source of Random IMprovement (RIM), (iii) the *neighbourhood structures* are represented by pitch adjustment which is the source of Local IMprovement (LIM). Each scenario ran 10 times. Note that the NI= 100,000 is fixed for all experiments.

As shown in Table 5, the first 11 scenarios simulate the behaviour of population-based methods, i.e., HMS $\geq$ 1. The remaining scenarios (i.e., 12-17) simulate the behaviour of local search-based methods.

The value of HMCR determines whether the proposed method used only memory consideration component (i.e., GIM) when HMCR =100% or memory consideration (i.e., GIM) and random consideration (i.e., RIM) components when HMCR $<$ 100%. The value of PAR determines the rate of any gradient decent (LIM). when PAR=0, no LIM is obtained. Larger PAR refers to the rate of using pitch adjustment procedures. Note that PAR determines the values of PAR1, PAR2 and PAR3 (PAR1=PAR/3, PAR2= 2PAR/3, and PAR3=PAR).

The combination of metaheuristic components based on the type of improvements is shown in the last column of Table 5.

Analogies:

– Some scenarios combine GIM + RIM + LIM components (i.e., Scens. (5, 6, 10, 11)) in which the proposed method behaves similar to the Memetic Algorithm (MA) (Establish a good balance between exploration and exploitation).

– Some scenarios combine GIM + RIM components (i.e., Scens. (4, 9)) in which the proposed method behaves similar to Genetic Algorithm (GA) (Establish a balance between exploration and exploitation).
– Some scenarios combine GIM + LIM components (i.e., Scens. (2, 3, 7, 8)) in which the proposed method behaves similar to the MA without the mutation operator. (Easily getting stuck in local optimal solution since there is no RIM component)
– Some scenarios combine LIM + RIM components(i.e.,Scens. (15, 16, 17)) in which the proposed method behaves similar to the Simulated Annealing (SA) (Establish a good balance between exploration and exploitation).
– Scen. (1) has only GIM component in which the proposed method behaves similar to the GA without a mutation operator (This leads to a premature convergence problem).
– Scen. (13) has only LIM component in which the proposed method behaves similar to the Hill Climbing (Easily getting stuck in the local optimal solution).
– Scen. (14) has only RIM component which means the proposed method can be considered in this case a local search-based method (i.e.,HMS=1) but without neighbourhood structures. The proposed method is able to improve the solution by an iterative construction process.

Note that in Scen (12), the proposed method does not have any improvement component which means that the initial solution will remain the same during the search.

4.1.2 Experimental Results

Tables 6, 7 , and 8, summarise the results of the 17 scenarios on the penalty value of the solution by recording the best, average, worst and standard deviation (std.dev.) over 10 runs. Note that the three tables are separated with reference to various HMS. The best solution for each dataset is highlighted in bold. The results show that combining GIM + LIM + RIM components (e.g., Scens. (5, 6, 10, 11)) in the same method in general is promising.

4.1.3 Discussion

A closer look at Tables 6, 7 and 8, each scenario reflects the behaviour of the proposed method when one, two or three components are combined. It has to be noted that in HSA, the memory consideration is the source of GIM, random consideration is the source of RIM, and the three pitch adjustment procedures are the source of LIM.

Observations:

– The best solutions are obtained from the scenarios that combine GIM + RIM + LIM components. However, some scenarios that combine GIM + LIM components are able to yield few number of the best solutions (see Scen (2, 3) for CAR-S-91-I and UTA-S-92-I). Note that these scenarios might be affected with exceptional random consideration (ERC) which diversify the search (see Sect. 4.2).
– The overall best results are obtained when the HMS = 50, this suggests that larger HMS allow the HSA to explore multiple search space regions simultaneously which may produce better quality solutions.

– The HMCR affects the balance between exploration and exploitation which means that the larger HMCR leads to less exploration and the greater exploitation. For example, the proposed method in Scen (16, 17) is concerned with exploration rather than exploitation and thus the speed of convergence will be slow.
– The PAR is the rate of any gradient descent. Since it had the larger values, it produced the best results. In other words, the larger the PAR values are, the more rigorous is the fine-tuning of the search space region to which the HSA converges. It is also noted from the comparative evaluation as will be shown in Sect. 4.3 that the best cited results for Carter dataset come from local search-based methods. This suggests that the components that are concerned with exploitation are more useful than those concerned with exploration for UETP. In fact, this is one of the reasons why the most timetabling researchers have lately turned to use local search-based methods rather than population-based method in their timetabling problems.
– The HSA method produces the best results for some problem instances (see CAR-S-91-I, EAR-F-83-I, LSE-F-91, STA-F-83-I) when the value of PAR is 3% while the remaining best results are obtained when the value of PAR is 30%. Note that the PAR is the rate of any gradient decent which indicates that some problem instance can be efficiently tackled when the LIM is few in number while others can be efficiently tackled when the LIM is great in number. In general, some search space reigns of the timetabling problems are very rugged and need considerable local changes until the local optimal solution is obtained.
– The results obtained by Scen. (13, 14, 15) worth considering. Scen (13) simulates a local search-based method with only LIM component which similar to Hill Climbing with three neighbourhood structures (move, Swap, and Kempe Chain). In contrast, Scen. (14) simulates a local search-based method with only RIM component. The best results are obtained from Scen. (15) where the HSA combines LIM + RIM components. It is apparent that a local search-based method with an explorative strategy is able to yield better results than those with no explorative strategy.

4.2 Analysis of the Exceptional Random Consideration (ERC) and the restart process

Using the memory consideration, the proposed method might be unable to assign some exams with timeslots based on HM solutions (i.e., There are no feasible timeslots for some exams in the HM solutions). Therefore, the exceptional random consideration (ERC) attempts to assign these exams with timeslots from their available range (see Algorithm 1, STEP 3, Line 7).

However, if the ERC or random consideration were unable to assign any exam with a timeslot from their available range, the improvisation process would *restart* all over again with a different random seed (restart process). (see Algorithm 1, STEP 3, Lines 9 and 23).

Table 9 shows statistical information on the effect of ECR and the restart process on the behaviour of the proposed method with various HMS. Each number in C1 column is to the average number of exams that are assigned with timeslots based on ERC per 100,000 iterations calculated as follows:

$$C1 = \frac{\sum_{i=1}^{NI} \#\text{ exams assigned with timeslot using ERC}}{NI}$$

Table 6 The Penalty Values obtained by the proposed HSA in different convergence scenarios (Note that the HMS = 50)

Dataset		SCEN.1	SCEN.2	SCEN.3	SCEN.4	SCEN.5	SCEN.6
CAR-S-91-I	Best	5.91	**4.99**	5	5.25	5	5.04
	Average	6.06	5.08	5.14	5.36	5.08	5.13
	Worst	6.16	5.47	5.27	5.45	5.16	5.25
	Std.dev.	0.08	0.14	0.1	0.08	0.06	0.07
CAR-F-92-I	Best	5.04	4.35	4.29	4.56	4.31	5.93
	Average	5.22	4.43	4.44	4.76	4.38	6.1
	Worst	5.37	4.48	4.59	4.88	4.43	6.23
	Std.dev.	0.1	0.05	0.08	0.11	0.05	0.09
EAR-F-83-I	Best	38.77	36.61	34.8	35.63	**34.42**	34.91
	Average	39.91	37.36	35.34	36.56	35.51	35.33
	Worst	40.91	38.21	35.76	37.93	36.58	35.94
	Std.dev	0.851	0.584	0.337	0.825	0.674	0.459
HEC-S-92-I	Best	11.8	11.1	11	11.1	10.6	**10.4**
	Average	12.4	11.4	10.9	11.4	11	10.7
	Worst	12.7	11.8	11.1	11.6	11.4	11
	Std.dev.	0.33	0.24	0.14	0.2	0.24	0.19
KFU-S-93	Best	16.48	14.07	**13.5**	15.08	14	13.66
	Average	17.24	14.34	14.02	15.28	14.42	13.78
	Worst	18.13	14.58	14.46	15.68	14.82	13.94
	Std.dev.	0.584	0.216	0.374	0.203	0.269	0.099
LSE-F-91	Best	13.06	10.96	10.6	11.39	**10.48**	10.69
	Average	13.62	11.49	10.71	11.75	10.77	10.88
	Worst	14.03	11.86	10.9	12.11	11.06	11.19
	Std.dev.	0.265	0.32	0.113	0.248	0.214	0.163
RYE-S-93	Best	10.76	8.98	9.04	9.773	8.85	**8.79**
	Average	11.22	9.16	9.16	10.03	9.07	8.94
	Worst	11.51	9.31	9.27	10.2	9.46	9.17
	Std.dev.	0.244	0.12	0.08	0.13	0.18	0.14
STA-F-83-I	Best	157.92	157.16	157.2	157.19	**157.04**	157.12
	Average	158.6	157.35	157.96	157.49	157.16	157.21
	Worst	159.59	157.54	159.26	157.73	157.36	157.27
	Std.dev.	0.5233	0.1446	0.7619	0.1662	0.0916	0.0351
TRE-S-92	Best	9.58	8.36	8.2	8.41	8.26	**8.16**
	Average	9.65	8.52	8.32	8.75	8.35	8.32
	Worst	9.73	8.71	8.42	9.07	8.43	8.46
	Std.dev.	0.07	0.11	0.09	0.21	0.06	0.09
UTA-S-92-I	Best	4.05	3.51	**3.43**	3.59	3.46	3.49
	Average	4.11	3.55	3.51	3.73	3.52	3.55
	Worst	4.16	3.62	3.59	3.83	3.55	3.61
	Std.dev.	0.04	0.04	0.05	0.07	0.03	0.04
UTE-S-92	Best	27.1	25.75	25.6	25.66	25.3	**25.09**
	Average	28.95	26.28	25.95	26.22	25.85	25.45
	Worst	30.03	26.62	26.35	26.9	26.37	25.76
	Std.dev.	0.836	0.298	0.231	0.483	0.329	0.21
YOR-F-83-I	Best	39.79	37.35	36.13	37.73	36.32	**35.86**
	Average	40.85	37.88	37.04	38.4	37.3	36.56
	Worst	41.55	38.71	38.21	38.98	38.79	36.87
	Std.dev.	0.668	0.377	0.671	0.392	0.74	0.31

Table 7 The Penalty Values obtained by the proposed HSA in different convergence scenarios (Note that the HMS = 10)

Data set		SCEN.7	SCEN.8	SCEN.9	SCEN.10	SCEN.11
CAR-S-91-I	Best	5.02	5.09	5.2	5.19	5.17
	Average	5.15	5.65	5.22	5.42	5.26
	Worst	5.23	6.9	5.26	5.57	5.36
	Std.dev.	0.07	0.75	0.03	0.12	0.06
CAR-F-92-I	Best	4.52	5.91	4.69	4.47	4.75
	Average	4.62	6.22	4.81	4.58	5.53
	Worst	4.72	6.36	5.01	4.69	5.9
	Std.dev.	0.08	0.13	0.12	0.09	0.38
EAR-F-83-I	Best	36.1	37.21	35.72	34.91	34.93
	Average	38.11	38.04	38.69	36.26	35.81
	Worst	38.93	39.39	41.42	37.54	36.77
	Std.dev	0.88	0.809	1.69	1.059	0.747
HEC-S-92-I	Best	11.1	11	11.2	10.6	10.5
	Average	11.6	11.3	11.7	11.1	10.7
	Worst	11.8	11.8	12.2	11.5	10.8
	Std.dev.	0.24	0.26	0.33	0.29	0.12
KFU-S-93	Best	14.25	13.77	14.88	13.93	13.56
	Average	14.8	14.23	15.5	14.2	13.74
	Worst	15.68	14.64	16.41	14.56	13.97
	Std.dev.	0.489	0.302	0.423	0.188	0.153
LSE-F-91	Best	11.15	10.58	11.84	10.73	10.59
	Average	11.63	11.16	12.29	11.29	11.14
	Worst	11.98	11.59	12.72	12.02	11.94
	Std.dev.	0.231	0.334	0.325	0.409	0.373
RYE-S-93	Best	9.09	9.01	10.4	8.86	8.84
	Average	9.56	9.17	10.9	9.23	9.04
	Worst	9.78	9.54	11.6	9.57	9.17
	Std.dev.	0.21	0.15	0.37	0.26	0.15
STA-F-83-I	Best	157.1	157.14	157.31	157.07	157.17
	Average	157.23	157.28	157.66	157.21	157.28
	Worst	157.5	157.44	157.97	157.44	157.38
	Std.dev.	0.1449	0.106	0.2392	0.1481	0.0755
TRE-S-92	Best	8.68	8.55	9.09	8.43	8.27
	Average	8.9	8.65	9.32	8.54	8.32
	Worst	9.32	9.02	9.51	8.69	8.36
	Std.dev.	0.19	0.14	0.13	0.09	0.03
UTA-S-92-I	Best	3.89	3.56	3.68	3.62	3.54
	Average	4	3.64	3.71	3.68	3.66
	Worst	4.11	3.72	3.77	3.74	3.78
	Std.dev.	0.07	0.06	0.03	0.04	0.06
UTE-S-92	Best	26.44	26.09	25.91	25.51	25.37
	Average	27	26.49	26.94	25.91	25.76
	Worst	27.92	26.91	28.49	26.55	26.49
	Std.dev.	0.52	0.245	0.754	0.351	0.35
YOR-F-83-I	Best	37.83	37.54	38.25	36.98	36.17
	Average	39	38.01	40.3	37.65	37.3
	Worst	40.03	38.38	41.96	38.46	38.32
	Std.dev.	0.66	0.282	1.328	0.499	0.69

Table 8 The Penalty Values obtained by the proposed HSA in different convergence scenarios (Note that the HMS = 1)

Data set		SCEN.12	SCEN.13	SCEN.14	SCEN.15	SCEN.16	SCEN.17
CAR-S-91-I	Best	8.28	5.52	6.19	5.49	7.58	7.64
	Average	8.73	5.67	6.45	5.75	7.66	7.75
	Worst	9.05	5.86	6.72	5.98	7.88	7.86
	Std.dev.	0.24	0.11	0.17	0.17	0.1	0.08
CAR-F-92-I	Best	7.42	4.65	5.2	4.45	6.22	6.3
	Average	7.77	4.77	5.46	4.6	6.42	6.41
	Worst	8.05	4.88	5.63	4.74	6.6	6.49
	Std.dev.	0.21	0.08	0.14	0.09	0.12	0.08
EAR-F-83-I	Best	53.92	38.32	39.99	35.27	44.08	42.71
	Average	56.91	40.12	41.5	37.5	44.95	43.95
	Worst	58.89	41.48	43.83	38.93	46.01	46.08
	Std.dev	1.363	0.99	1.132	1.263	0.644	1.054
HEC-S-92-I	Best	17	11.2	11.2	10.7	11.9	11.5
	Average	19.1	11.6	11.8	11	12.4	11.9
	Worst	22.3	11.9	12.5	11.4	12.8	12.2
	Std.dev.	1.68	0.21	0.46	0.28	0.32	0.23
KFU-S-93	Best	23.99	14.49	15.05	14.24	19.09	16.66
	Average	27.13	14.89	15.91	14.47	19.87	17.03
	Worst	30.08	15.39	16.65	14.84	20.43	17.39
	Std.dev.	1.793	0.265	0.547	0.193	0.37	0.247
LSE-F-91	Best	19.29	11.36	11.55	11.29	14.78	15.35
	Average	21.27	11.83	12.01	11.7	15.47	15.87
	Worst	22.35	12.4	12.38	12.4	15.85	16.48
	Std.dev.	0.904	0.316	0.244	0.346	0.354	0.436
RYE-S-93	Best	19.4	9.51	11.1	10.2	14.7	14.4
	Average	20.9	9.99	11.4	10.5	15	14.8
	Worst	22.4	10.2	11.9	10.8	15.1	15.3
	Std.dev.	0.98	0.22	0.34	0.24	0.13	0.32
STA-F-83-I	Best	168.99	157.41	157.68	157.21	158.65	158.52
	Average	177.73	158.13	157.9	157.35	159.64	159.06
	Worst	185.87	158.39	158.3	157.44	160.59	159.77
	Std.dev.	5.0913	0.3182	0.235	0.0831	0.6015	0.4025
TRE-S-92	Best	12.3	9	9.48	8.63	10.4	10.4
	Average	13.3	9.26	9.75	8.97	10.7	10.7
	Worst	14	9.53	10.1	9.13	11	11
	Std.dev.	0.55	0.17	0.17	0.14	0.21	0.21
UTA-S-92-I	Best	5.71	3.71	4.13	3.93	4.92	4.82
	Average	6.16	3.8	4.32	3.99	5.02	4.91
	Worst	6.6	3.91	4.57	4.06	5.14	5.05
	Std.dev.	0.24	0.07	0.13	0.05	0.07	0.07
UTE-S-92	Best	38.88	27.54	26.47	25.7	29.55	29.91
	Average	41.94	28.51	27.16	26.1	30.68	30.38
	Worst	44.31	30.11	27.87	26.93	31.83	31.87
	Std.dev.	1.48	0.741	0.425	0.389	0.724	0.569
YOR-F-83-I	Best	50.42	39.46	40.47	38.59	43.71	43.96
	Average	53.08	40.84	42.86	39.36	45.03	45.07
	Worst	54.66	42.59	46.54	40.03	47.11	46.47
	Std.dev.	1.306	1.103	1.695	0.412	1.051	0.812

Table 9 The numbers of using ERC and restart process on Carter dataset per 100,000 iterations

Data set	HMS=1		HMS=10		HMS=50	
	C1	C2	C1	C2	C1	C2
CAR-S-91-I	141.89	4	48.77	4	14.54	7
CAR-F-92-I	77.83	81	34.1	14	11.15	160
EAR-F-83-I	13.16	256	12.95	70	4.59	295
HEC-S-92-I	4.11	2602	4.03	2867	3.11	5188
KFU-S-93	21.36	1170	16.77	1568	4.16	2663
LSE-F-91	18.09	637	12.58	636	2.51	861
RYE-S-93	27.64	958	13.66	319	3.57	1175
STA-F-83-I	4.32	0	4.01	0	2.09	0
TRE-S-92	20.73	2	30.11	186	4.46	31
UTA-S-92-I	95.55	25	48.87	13	9.57	38
UTE-S-92	4.79	985	3.88	489	1.84	2581
YOR-F-83-I	22.78	1202	18.71	377	4.55	1588

Each number in C2 column is the total number of iterations skipped per 100,000 (restart process).

The results in C1 suggest that increasing the HMS value in general reduces the number of exams assigned with timeslots using ERC per iteration. Although the results in C2 are not related to the HMS values, larger size and complexity of Carter dataset lead to a larger number of skipped iterations (restart process).

4.3 Comparative Evaluation and Analysis

The proposed Harmony Search Algorithm (HSA) is compared with some published methods using the Carter datasets, known and available for the authors. This includes a total of 22 comparator methods which comprise Local Search-based MetaHeuristic Methods (LSMHM), Population-based MetaHeuristic Methods (PMHM), Heuristic Methods (HM) and Hyper-Heuristic Methods (HHM) (See Table 10).

The results of the proposed method were compared with 22 comparative methods as shown in Tables 11, 12 and 13. The numbers in these tables refer to the Penalty Value (PV) calculated by Eq.(1). The indicator '-' shows where the method did not guarantee a feasible timetable (e.g, a hard constraint was not met) or the method did not test the corresponding dataset. The numbers in bold show the best solution obtained for that Carter dataset (lowest is best). The numbers in italic fonts indicate that a different dataset version was used. Note that the results obtained by the proposed method were collected from Tables 6, 7 and 8.

In the proposed method, the results outperformed those produced by heuristic and hyper-heuristic methods in 8 out of 12 Carter datasets shown in Table 11. Clearly, the heuristic and hyper-heuristic methods have not as yet measured up to the results obtained by the metaheuristic-based methods in terms of solution quality.

Table 12 shows the results of the proposed method compared with local-search based methods. The proposed method produces better overall results in 5 out of 11 comparative methods in all Carter dataset. Also the proposed method outperformed the local-search based methods in 2 out 12 Carter datasets. It has to be noted that these methods produced the best solutions cited above. Table 13 lists the results of the proposed method in comparison with the population-based methods. Once again in 9 out of 12 Carter dataset, the proposed method achieved better results

Table 10 Key to the comparator methods

#	Method	Class	Reference
0	Harmony Search Algorithm	PMHM	< *proposed method*>
1	Graph Coloring Heuristic Methods	HM	Carter et al. (1996)
2	Tabu Search Algorithm	LSMHM	Di Gaspero and Schaerf (2002)
3	Tabu Search Algorithm	LSMHM	Di Gaspero (2002)
4	Tabu Search Algorithm	LSMHM	Paquete and Stutzle (2003)
5	Local Search-based Methods	LSMHM	Burke and Newall (2003)
6	GRASP local Search-based Method	LSMHM	Casey and Thompson (2003)
7	Simulated Annealing and Hill Climbing	LSMHM	Merlot et al. (2003)
8	Time-Predefined Great Deluge	LSMHM	Burke et al. (2004)
9	Adaption of Heuristic Orderings	HM	Burke and Newall (2004)
10	Similarity Measure for Heuristic Selection	HM	Yang and Petrovic (2005)
11	Fuzzy Multiple Heuristic Orderings	HM	Asmuni et al. (2005)
12	Tabu Search Hyper-Heuristic Approach	HHM	Kendall and Hussin (2005)
13	Multi-Objective Evolutionary Algorithm	PMHM	Cote et al. (2005)
14	Hybrid Variable Neighbourhood	LSMHM	Burke et al. (2006)
15	Ant Colony Algorithm	PMHM	Eley (2007)
16	Graph-Based Hyper-Heuristic	HHM	Burke et al. (2007)
17	Ahuja-Orlin's Method	LSMHM	Abdullah et al. (2007)
18	Graph-Based Hyper-Heuristic	HHM	Qu and Burke (2009)
19	Local Search-based Methods	LSMHM	Caramia et al. (2008)
20	Graph-Based Hyper-Heuristic	HHM	Qu et al. (2009a)
21	Graph-Based Hyper-Heuristic	HHM	Pillay and Banzhaf (2009)
22	Fuzzy Multiple Heuristic Orderings	HM	Asmuni et al. (2009)

Finally, Table 14 shows the results of the proposed method in comparison with the best overall results obtained by the 22 comparative methods in Table 10. The differences between the results in the 3rd column indicate that the proposed method is able to produce respectable solutions that are very much near the best solutions. The last column in the Table 14 shows the ranking position of each result in a total of 22 comparative methods. For example, 5th position means that the result obtained by HSA ranks 5th out of 23 methods (including the proposed method).

Table 11: Comparison with heuristic and hyper-heuristic approaches

Data set	< *proposed method*>	Carter et al. (1996)	Burke and Newall (2004)	Asmuni et al. (2005)	Kendall and Hussin (2005)	Burke et al. (2007)	Qu et al. (2009a)	Qu and Burke (2009)	Pillay and Banzhaf (2009)	Asmuni et al. (2009)
CAR-S-91-I	4.99	7.1	5	5.19	5.37	5.36	5.11	5.16	**4.97**	5.29
CAR-F-92-I	4.29	6.2	4.3	4.51	4.67	4.93	4.32	**4.16**	4.84	4.54
EAR-F-83-I	**34.42**	36.4	36.2	36.64	40.18	37.92	35.56	35.86	36.86	37.02
HEC-S-92-I	**10.4**	10.8	11.6	11.6	11.86	12.25	11.62	11.94	11.85	11.78
KFU-S-93	**13.5**	14	15	15.34	15.84	15.2	15.18	14.79	14.62	15.8
LSE-F-91	**10.48**	10.5	11	11.35	-	11.33	11.32	11.15	11.14	12.09
RYE-S-93	8.79	**7.3**	-	10.05	-	-	-	-	9.65	10.38
STA-F-83-I	**157.04**	161.5	161.9	160.79	157.38	158.19	158.88	159	158.33	160.42
TRE-S-92	**8.16**	9.6	8.4	8.47	8.39	8.92	8.52	8.6	8.48	8.67
UTA-S-92-I	3.43	3.5	3.4	3.52	-	3.88	**3.21**	3.59	3.4	3.57
UTE-S-92	**25.09**	25.8	27.4	27.55	27.6	28.01	28	28.3	28.88	28.07
YOR-F-83-I	**35.86**	41.7	40.8	39.79	-	41.37	40.71	41.81	40.74	39.8

Table 12: Comparison with local based metaheuristic-based search approaches

Data set	< *proposed method*>	Caramia et al. (2008)	(Di Gaspero and Schaerf 2002)	Di Gaspero (2002)	Paquete and Stutzle (2003)	Burke and Newall (2003)	Casey and Thompson (2003)	Merlot et al. (2003)	Burke et al. (2004)	Yang and Petrovic (2005)	Burke et al. (2006)	Abdullah et al. (2007)
CAR-S-91-I	4.99	6.6	6.2	5.7	-	4.65	5.4	5.1	4.8	**4.5**	4.6	5.2
CAR-F-92-I	4.29	6	5.2	-	-	4.1	4.4	4.3	4.2	**3.93**	4	4.4
EAR-F-83-I	34.42	**29.3**	45.7	39.4	38.9	37.05	34.8	35.1	35.4	33.7	32.8	34.9
HEC-S-92-I	10.4	**9.2**	12.4	10.9	11.2	11.54	10.8	10.6	10.8	10.83	10	10.3
KFU-S-93	13.5	13.8	18	-	16.5	13.9	14.1	13.5	13.7	13.82	**13**	13.5
LSE-F-91	10.48	**9.6**	15.5	12.6	13.2	10.82	14.7	10.5	10.4	10.35	10	10.2
RYE-S-93	8.79	**6.8**	-	-	-	-	-	-	8.9	8.53	-	8.7
STA-F-83-I	**157.04**	158.2	160.8	157.4	168.3	168.73	*134.9*	157.3	159.1	158.35	159.9	159.2
TRE-S-92	8.16	9.4	10	-	9.3	8.35	8.7	8.4	8.3	7.92	**7.9**	8.4
UTA-S-92-I	3.43	3.5	4.2	4.1	-	3.2	-	3.5	3.4	**3.14**	3.2	3.6
UTE-S-92	25.09	**24.4**	27.8	-	29	25.83	25.4	25.1	25.7	25.39	24.8	26
YOR-F-83-I	**35.86**	36.2	41	39.7	38.9	37.28	37.5	37.4	36.7	36.35	37.28	36.2

Table 13 Comparison with population based metaheuristic approaches

Data set	< *proposed method*>	Cote et al. (2005)	Eley (2007)
CAR-S-91-I	**4.99**	5.4	5.2
CAR-F-92-I	4.29	**4.2**	4.3
EAR-F-83-I	34.42	**34.2**	36.8
HEC-S-92-I	**10.40**	**10.4**	11.1
KFU-S-93	**13.5**	14.3	14.5
LSE-F-91	**10.48**	11.3	11.3
RYE-S-93	**8.79**	8.8	9.8
STA-F-83-I	**157.04**	158.03	157.3
TRE-S-92	**8.16**	8.6	8.6
UTA-S-92-I	**3.43**	3.5	3.5
UTE-S-92	**25.09**	25.3	26.4
YOR-F-83-I	**35.86**	36.4	39.4

5 Conclusion and future work

In this paper, we have conducted a preliminary investigation of combining the key metaheuristic components that lead to the best solutions for Uncapacitated Examination Timetabling Problem (UETP). Harmony Search Algorithm (HSA), tailored for the purpose of this study, is iterated toward an optimal solution using three components: Memory Consideration, Random Consideration, and Pitch Adjustment. The proposed method has been evaluated against 12 datasets defined by Carter et al. (1996) and has been able to obtain two best overall results when compared with those obtained by 22 comparative methods available to the researcher.

Three metaheuristic components have been investigated: recombination, randomness and neighborhood structures, that have been classified according to the types of improvement provided: global improvement (GIM), random improvement (RIM), and local improvement (LIM) respectively. we experimented with 17 convergence scenarios each of which simulating a combination among those components. The results show that the method combining GIM + RIM + LIM components can obtain high quality solutions for for most test timetabling instances. This suggests that hybridization be-

Table 14 The comparison between the proposed method and the best cited results obtained from approaches in Table 7, 8 and 9 of Carter benchmarks

Data set	< *proposed method*>	Best result cited	differences	position – out of 23 methods
CAR-S-91-I	4.99	**4.5**	0.49	5th
CAR-F-92-I	4.29	**3.93**	0.36	7th
EAR-F-83-I	34.42	**29.3**	5.12	5th
HEC-S-92-I	10.4	**9.2**	1.2	4th
KFU-S-93	13.5	**13**	0.5	2nd
LSE-F-91	10.48	**9.6**	0.88	6th
RYE-S-93	8.79	**6.8**	1.99	5th
STA-F-83-I	**157.04**	157.2	-0.16	1st
TRE-S-92	8.16	**7.9**	0.26	3rd
UTA-S-92-I	3.43	**3.14**	0.29	8th
UTE-S-92	25.09	**24.4**	0.69	3rd
YOR-F-83-I	**35.86**	36.2	-0.34	1st
Total	316.45	305.17	11.28	

tween the key components of local search-based methods and those of population-based methods, is a promising research area which can produce good solutions for the most difficult UETP instances.

The convergence scenarios can also be seen as a parameter sensitivity analysis for the proposed HSA. Their results suggest that increasing HMS leads to increasing the ability of the proposed method to explore multiple search space regions simultaneously. Furthermore, the more the HCMR is, the less exploration and greater exploitation will be. In the timetabling domain, exploitation is more useful than exploration due to the structure of the search space. As such HMCR should be large enough to avoid the random search. PAR is the rate of any gradient descent. Larger PAR leads to rigorous fine-tuning in the search space region and more exploitation. In conclusion, larger HMCR, PAR and HMS, lead to better results.

The combination of metaheuristics components using HSA as an optimisation framework can inspire future researchers with food for thought. For example, the acceptance rule of Simulated Annealing can be combined with STEP 4 of the HSA.

References

Abdullah S, Ahmadi S, Burke EK, Dror M (2007) Investigating ahuja-orlin's large neighbourhood search approach for examination timetabling. OR Spectrum 29(2),351–372

Al-Betar MA, Khader AT (2009) A hybrid harmony search for university course timetabling. In: Blazewicz J, Drozdowski M, Kendall G, McCollum B (eds) Proceedings of the 4nd Multidisciplinary Conference on Scheduling: Theory and Applications (MISTA 2009), Dublin, Ireland, pp 157–179

Al-Betar MA, Khader AT, Gani TA (2008) A harmony search algorithm for university course timetabling. In: 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, Canada

Al-Betar MA, Khader AT, Liao IY (2010) A harmony search algorithm with multi-pitch adjusting rate for university course timetabling. In: Geem Z (ed) Recent Advances In Harmony Search Algorithm, Studies in Computational Intelligence (SCI), vol 270, Springer-Verlag, Berlin, Heidelberg, pp 147–162

Asmuni H, Burke EK, Garibaldi JM, McCollum B (2005) Fuzzy multiple heuristic orderings for examination timetabling. In: Proceedings of the 5th International Conference onPractice and Theory of Automated Timetabling (PATAT2004), LNCS, vol 3616, Berlin: Springer-Verlag, Pittsburgh, PA, USA

Asmuni H, Burke EK, Garibaldi JM, McCollum B, Parkes AJ (2009) An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. Computers & Operations Research 36(4),981–1001

Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surv 35(3),268–308

Brélaz D (1979) New methods to color the vertices of a graph. Commun ACM 22(4),251–256

Burke E, Newall J (2004) Solving examination timetabling problems through adaption of heuristic orderings. Annals of Operations Research 129(1),107–134

Burke EK, Newall JP (2003) Enhancing timetable solutions with local search methods. In: in Proceedings of the 4th International Conference on Practice and Theory of

Automated Timetabling (PATAT2002), LNCS, vol 2740, Berlin: Springer-Verlag, KaHo St.-Lieven, Gent, Belgium, pp 195–206

Burke EK, Bykov Y, Newall J, Petrovic S (2004) A time-predefined local search approach to exam timetabling problems. IIE Transactions 36(6),509–528

Burke EK, Eckersley AJ, McCollum B, Petrovic S, Qu R (2006) Hybrid variable neighbourhood approaches to university exam timetabling. Tech. Rep. Technical Report NOTTCS-TR-2006-2, School of Computer Science, University of Nottingham, UK

Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research 176(1),177–192

Caramia M, Dell'Olmo P, Italiano G (2008) Novel local-search-based approaches to university examination timetabling. Informs Journal on Computing 20(1),86–99

Carter MW, Laporte G, Lee SY (1996) Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society 74,373–383

Casey S, Thompson J (2003) Grasping the examination scheduling problem. In: in Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT2002), LNCS, vol 2740, Berlin: Springer-Verlag, KaHo St.-Lieven, Gent, Belgium, pp 232–244

Cote P, Wong T, Sabouri R (2005) Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In: Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT2001), LNCS, vol 3616, Berlin: Springer-Verlag, pp 151–168

Di Gaspero L (2002) Recolour, shake and kick: A recipe for the examination timetabling problem. In: in Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT2002), KaHo St.-Lieven, Gent, Belgium

Di Gaspero L, Schaerf A (2002) Tabu search techniques for examination timetabling. In: Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT2001), LNCS, vol 3616, Berlin: Springer-Verlag

Eley M (2007) Ant algorithms for the exam timetabling problem. In: Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT2001), LNCS, vol 3616, Berlin: Springer-Verlag, pp 364–382

Fesanghary M, Mahdavi M, Minary-Jolandan M, Alizadeh Y (2008) Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems. Computer Methods in Applied Mechanics and Engineering 197(33-40),3080–3091

Geem ZW, Kim JH, Loganathan GV (2001) A New Heuristic Optimization Algorithm: Harmony Search. Simulation 76(2),60–68

Ingram G, Zhang T (2009) Overview of applications and developments in the harmony search algorithm. In: Geem ZW (ed) Music-Inspired Harmony Search Algorithm, Springer, Berlin, Heidelberg, pp 15–37

Kendall G, Hussin N (2005) A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. In: Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT2001), LNCS, vol 3616, Berlin: Springer-Verlag, pp 270–293

Lee K, Geem ZW, Lee Sh, Bae Kw (2005) The harmony search heuristic algorithm for discrete structural optimization. Engineering Optimization 37(7),663–684

Lee KS, Geem ZW (2004) A new structural optimization method based on the harmony search algorithm. Computers and Structures 82(9-10),781 – 798

Lee KS, Geem ZW (2005) A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. Computer Methods in Applied Mechanics and Engineering 194(36-38),3902–3933

McCollum B, Schaerf A, Paechter B, McMullan P, Lewis R, Parkes A, Di Gaspero L, Qu R, Burke EK (2009) Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS JOURNAL ON COMPUTING DOI 10.1287/ijoc.1090.0320

Merlot LT, Boland N, Hughes BD, Stuckey PJ (2003) A hybrid algorithm for the examination timetabling problem. In: in Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT2002), LNCS, vol 2740, Berlin: Springer-Verlag, KaHo St.-Lieven, Gent, Belgium, pp 207–231

Ochoa G, Qu R, Burke EK (2009) Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In: GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, pp 341–348

Paquete L, Stutzle T (2003) Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem. In: in Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT2002), LNCS, vol 2740, KaHo St.-Lieven, Gent, Belgium, pp 413–420

Pillay N, Banzhaf W (2009) A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. European Journal of Operational Research 197(2),482–491

Qu R, Burke E (2009) Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. Journal of the Operational Research Society 60,1273–1285

Qu R, Burke EK, , McCollum B (2009a) Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. European Journal of Operational Research 198(2),392–404

Qu R, Burke EK, McCollum B, Merlot LTG, Lee SY (2009b) A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12(1),55–89

Yang XS (2009) Harmony search as a metaheuristic algorithm. In: Geem ZW (ed) Music-Inspired Harmony Search Algorithm, Springer, Berlin, Heidelberg, pp 1–14

Yang Y, Petrovic S (2005) A novel similarity measure for heuristic selection in examination timetabling. In: Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT2001), LNCS, vol 3616, Berlin: Springer-Verlag, pp 247–269

Bridging the gap between self schedules and feasible schedules in staff scheduling

Eyjólfur Ingi Ásgeirsson

Abstract Every company that has employees working on irregular schedules must deal with the difficult and time consuming problem of creating feasible schedules for the employees. We introduce an algorithm that takes a partial schedule created by requests from employees and creates feasible schedule where most of the employee's requests are unchanged, while still making sure that rules and regulations are not violated. The algorithm is based on independent modules, which can be executed in any order, and each module tries to emulate some action taken by a staff manager.

Our goal is to create a transparent and fair system that creates feasible schedules of high quality, but also a system where the employees can get an explanation and justification for every change that the algorithm makes to the employee requests. By emulating the actions of staff managers, the algorithm is easily understood by staff managers and, using detailed logs of any action, make any decision easy to explain to the employees.

We will present the algorithm and show results from four real world companies and institutions. The results show that a simple module based heuristic can get good results and create fair and feasible schedules that encourage employees to participate in the self-scheduling process.

Keywords Staff scheduling · Rostering · Heuristics · Local search

1 Introduction

Staff scheduling is a difficult and time consuming problem that every company or institution that has employees working on shifts or on irregular workdays must solve. A variant of the general staff scheduling problem focuses on scheduling the working hours for nurses in the health industry, so called nurse rostering or nurse scheduling

Work done in collaboration with Vaktabestun ehf.

E. I. Ásgeirsson
Reykjavik University
School of Science and Engineering.
Menntavegur 1, 101 Reykjavik, Iceland.
E-mail: eyjo@ru.is

[6]. The nurse rostering problem is well known and has been studied for over 45 years. The nurse rostering problem can include many types of constraints and covers a large set of staff scheduling problems, so even though our work is not specific for hospitals or nurses, most of the previous work focuses on nurse rostering.

There are three major approaches used in nurse rostering: cyclical scheduling [15], self scheduling and preference scheduling [3]. In cyclical scheduling, several sets of schedules are generated and then the nurses are assigned to a schedule that best fits their preferences so that collectively they satisfy the manpower requirements. The cyclical scheduling approach is rather rigid and therefore difficult to use in a flexible and changing environment.

In preference scheduling, each employee gives a list of preferences to the personnel manager who then creates a schedule that satisfies the demand for personnel and work restrictions while trying to fulfill as many preferences as possible. Some form of preference scheduling is widely used in real world environments and it has many benefits, such as flexibility, individual tailoring of the schedule and so forth. The major drawback to preference scheduling is the time required to create a high quality schedule that fulfills as many preferences as possible.

The self scheduling approach moves the responsibility of creating a schedule to the employees. The employees are given the required minimum and maximum number of employees that should be on duty at each time. The employees are then asked to sign up for the shifts that they want to work on, with the requirement that the resulting schedule must be a feasible schedule. Pure self-scheduling is difficult to implement fairly, in many instances it becomes easy for someone to manipulate the system, the sign-up order is important, new employees are likely to be at a disadvantage due to unfamiliarity with the system and, due to any number of reasons, some employees might not sign up for any shifts. There are however potentially many motivational benefits of self scheduling, such as improved co-operation, greater staff satisfaction and commitment, and reduced staff turnaround [16]. In real-life, self-scheduling can be implemented as a mixture of pure self-scheduling and preference scheduling. In this approach the employees sign up for shifts, creating a preliminary schedule and the staff manager then turns the preliminary schedule into an feasible staffing schedule, making sure that no rules have been violated and the staffing load fits the manpower need at each time. The final responsibility of creating a good schedule lies with the staff manager, but the employees see the manpower need and the current staff levels when signing up for shifts, so they share the responsibility of creating the schedule. This approach can include an incentive scheme, such as priority on popular shifts, to encourage the employees to create high quality preliminary schedules.

Mathematical programming techniques are in many cases too rigid to deal with the multiple and often changing, objectives and goals of staff scheduling. The research on staff scheduling is now mainly focused on more flexible metaheuristic approaches such as genetic algorithms [1, 2, 13, 18] and variable neighborhood search [7], with Tabu-Search [5, 12] and Simulated Annealing [4] being particularly successful [14]. Burke et. al. [8] introduced a multi-criteria metaheuristic approach based on Tabu-search, which is used in Belgian hospitals. Their system allows for user defined parameters, giving the users the opportunity to adjust the algorithm to their need and to the specifics of their problem.

There are two major problems with using black-box methods such as meta-heuristics and mathematical programming methods with the self-scheduling scheme introduced above. The first problem is that it is difficult to incorporate the experience and exper-

tise of the staff managers into such techniques [17]. Staff managers often have highly valuable knowledge, experience, and detailed understanding of their specific staffing problem, which will vary from company to company. The variation in the problem specifics between the different companies makes it also difficult to incorporate a single solution that fits all.

The second major problem is that the employees have spent time and effort into creating the preliminary schedule, so they have an emotional attachment to their selection. Since the staff manager has the final responsibility of the schedule, the staff manager is also responsible to the employees for any changes that are made to the preliminary schedule. The employees whose schedules are changed and whose wishes are ignored will demand to know why. Having a black-box method makes it impossible to see exactly why each choice is made, which makes it impossible to justify or explain individual decisions made by the algorithm.

2 Staff Scheduling

The staff scheduling problem we look at is a mixture of self scheduling and preference scheduling. The employees sign up for shifts and indicate periods where they cannot work. During the selection process, every employee has full information of how many employees are signed up for each shift and the required minimum and maximum staffing levels. The resulting plan, or preliminary schedule, is then used by the management as a foundation to create a feasible schedule. The quality of the preliminary schedule determines how close we are to either pure self scheduling or preference scheduling. If the preliminary schedule is close to feasible, then we have a self scheduling system, but on the other hand, if the preliminary schedule is far from being feasible, then the system is closer to preference scheduling, which entails the time consuming process of creating a feasible schedule from the preliminary schedule.

Our goal is to automate the time consuming process of creating a feasible schedule which is the drawback of preference scheduling systems. We will use a combination of local search methods and heuristics to emulate the manual process of creating a high quality schedule based on the preliminary schedule. The preliminary schedule is unlikely to satisfy the minimum and maximum required number of employees on duty, there might be some employees that have not signed up for any shifts or too few shifts and there might even be employees with too many scheduled hours.

In the real world instances that we've analyzed, the schedules that are created manually from the preliminary schedule are usually accepted as fair by the employees, the problem is how time consuming the process is. Our system is designed to preserve the perceived fairness of the current manual system, which we do by ensuring that the decisions of the algorithm are transparent and easily justifiable using the available data. There are no constrictions on the shifts that we use, each company or institution will have a set of allowed shifts and the shifts can have different length, they can overlap and different days of the week can have different set of allowed shifts.

We will present the algorithm and give results using real data to show how effectively the local search methods and heuristics can create feasible schedules with high quality, while still satisfying personnel preferences and preserving the trust that the employees have in the current system.

3 Constraints

Staff scheduling problems have a large number of constraints, such as minimum or maximum number of employee on duty at each time, the requests of employees, union rules and other regulations that must be satisfied. We partition the constraints into hard and soft constraints, where the hard constraints must always be satisfied while we allow the soft constraints to be violated if necessary.

3.1 Hard Constraints

The hard constraints, i.e. the constraints that must be satisfied at all times, are mostly based on union contracts and contracts with the employees. The system allows for having different constraints for different employees. In our examples the main constraints are usually the same for all the employees, with the exception of work limits. The hard constraints that we use are the following:

- **Restrictions on working hours and rest periods from employee contracts and union regulations.** Each employee can have restrictions on when they can work, such as employees that will never work nights or weekends. Additionally, there will be union regulations on rest periods, maximum lengths of continuous work, minimum length of continuous rest between shifts and other limits.
- **Vacation requests.** We treat requests for vacations as hard constraints, so the algorithm will never assign work duty to people on vacation.
- **Working weekends.** There can be limits on how many weekends particular employees are working. Common limits include working at most 2 or 3 out of every 4 consecutive weekends.
- **Requests for time off.** Each employee can have a some hours where they wish to be off-duty. We treat such wishes as hard constraints, so the algorithm will never violate such wishes. The number of such hours depends on the company and on the employee contract.
- **Special shifts, training sessions or meetings.** In many instances, employees have work related duties that are not flexible and are not necessarily included in the number of people on duty. Such instances include training sessions, meetings or other special functions. Since training sessions and meetings are often not flexible, we make sure that the algorithm will not make any changes to such functions.
- **Other limits on shifts or working hours, such as double shifts**. Double shifts are defined as two separate shifts in the same 24 hour period, where the interval between the shifts is less than the minimum resting period between shifts.

The set of constraints that are used is flexible and different from one company to the next. In one of our examples, the use of double shifts is not only allowed but actually encouraged while other companies might consider schedules with double shifts as infeasible.

To evaluate constraints such as working weekends or minimum rest, the actual schedule from the previous period must be included in the input.

One of the problems with the hard constraints is that the employees are allowed to be more flexible when they are creating their own schedule than staff managers or improvement algorithms. For example, an employee might sign up for a 10 hour shift, while a staff manager or an algorithm can only assign shifts of 8 hours or less to the

same employee. Since the employees have more flexibility, the preliminary schedules often contain individual employee schedules that would be considered infeasible. To make any improvements to such a schedule, the algorithm must accept the infeasibility of the current schedule, but make sure that the proposed changes to the schedule do not violate any hard constraints. We use a penalty score system to handle infeasible schedules and to check if proposed changes are feasible. Each time an individual employee schedule violates a constraint, the schedule receives a penalty. If the cumulative penalty is above a certain threshold then the schedule is considered infeasible. To handle the fact that the schedules can be infeasible to start with, the penalty for any schedule is calculated before and after a proposed change. If the penalty increase is above the threshold, then the change is not allowed. Using a penalty for each constraint violation and a threshold not only allows us to use requests that would be considered infeasible, but it is also flexible since it allows us to specify exactly the number of times a particular rule must be violated before the proposed schedule is considered infeasible.

3.2 Soft Constraints

The algorithm is designed so that the hard constraints are always satisfied. The soft constraints can be broken at any time, and represent the goals of the scheduling process. It is often impossible to satisfy all soft and hard constraints, so we must sometimes settle for satisfying as many soft constraints as possible. The soft constraints we use are the following:

- **Minimum and maximum staff levels.** An estimate for the demand for employees on duty at each time during the scheduling period is one of the prerequisites of the scheduling process. Some companies use minimum and maximum number of on-duty employees for each time slot in the scheduling period, while other companies simply state exactly how many employees should be on duty at each time. One of the goals is to have the number of on-duty employees within the minimum and maximum at all times, or as close to the exact number of employees that should be on duty. Companies and institutions often use some type of forecasting to estimate the required number of staff on duty, but the sophistication and the accuracy of the demand predictions can vary greatly from one company to the next.
- **Minimum and maximum number of on-duty hours for each employee.** Each employee is hired to work a specific number of hours per week, typically 40 hours per week for a full time employment. For each planning period, the number of hours that the employee should sign up for is calculated, based on the number of hours per week and the number of actual working hours in previous periods. Since the employees are often working irregular hours, there must be some flexibility in the system. For the scheduling period, each employee is assigned minimum and maximum duty hours, and one of the goals of the rostering process is to make sure that all employees are within the minimum and maximum duty hours.
- **Employee requests for shifts.** Before each scheduling period, the employees sign up for shifts. One of the main goals of this project is to encourage employees to create their own work schedules, so it is important to keep as much of the requests as possible.
- **Employees assigned to shifts on weekends adjacent to their vacations.** If an employee is starting his or hers vacation on a Monday, it is likely that the

employee wants the weekend free. We try to make sure that if an employee is on a vacation on a Monday or Friday, the adjacent weekend will be free, unless the employee has requested to work on that particular weekend.

Each company can have different priority rules for the soft constraints, which are reflected in the order in which various modules and functions of the algorithm are executed. A typical priority rule is that having all employees within minimum and maximum duty hours takes highest priority, then the staff levels and finally the requests of the employees.

4 The staff scheduling algorithm

The algorithm that we use is a collection of independent modules or functions, where each module takes the current schedule and tries to improve it. We can call the modules in any order, here we present the order that we usually use, i.e. we first execute Algorithm 1, then Algorithm 2 and so on. Some modules are executed more than once with different input parameters, which we will elaborate on when we give detailed description of each module. Most of these algorithms are simple and some of them have been introduced before, such as [10,11], but for completeness we will introduce and explain all the algorithms.

Algorithm 1 RepairShifts

```
Input: set of employees E
Input: set of allowed shifts A
for all e ∈ E do
  for all s ∈ e.shifts do
    s ← find closest shift to s from the shifts in A
  end for
end for
```

Algorithm 1 is used to make sure that all employees are only working on shifts that are allowed. The systems that the employees use to request shifts sometimes allow the employees to sign up for any hours. However, the collaborating companies and institutions restrict the employees to only work on certain shifts. The first step of the algorithm is to take the preliminary input and make sure that all employees are only signed up on shifts that are allowed. The variable $e.shifts$ denotes the set of shifts that employee e is signed up for. The distance between any two shifts s and t is $|s.start - t.start| + |s.end - t.end| + |s.length - t.length|$, where $s.start$ is the start time of shift s, $s.end$ is the end time of shift s and $s.length$ is the length of shift s, and similarly for shift t. The start and end times of the shifts are measured as the number of hours from the start of the scheduling period while the length of a shift is measured in hours. The length of a shift is included in the measure since we want to find an allowed shift that is both similar to the chosen shift in time and length.

The module shown in Algorithm 2 uses a priority rule to determine which employee is selected when the algorithm needs to remove an employee from an overstaffed shift. The priority rule can be different from one company to the next, in the preliminary version of the algorithm we use the number of working hours to determine which employee should be removed from an overstaffed shift.

Algorithm 2 Overstaffing: Remove shifts from employees

```
Input: set of overstaffed shifts S
while S ≠ ∅ do
    s ← select the maximum overstaffed shift from S
    E(s) ← {e ∈ E : s ∈ e.shifts}
    e ← select lowest priority employee from E(s)
    e.shifts ← e.shifts \ {s}
    Update s
    Update S
end while
```

Algorithm 3 Understaffing: Add shifts to employees

```
Input: Set of employees E
Input: Set of understaffed shifts S
while S ≠ ∅ do
    s ← select the shift with the largest total understaffing from S
    P(s) ← select all employees that can work on shift s.
    if P(s) ≠ ∅ then
        e ← select the employee from P(s) with fewest scheduled working hours.
        e.shifts ← e.shifts ∩ {s}
        Update S and E
    else
        S ← S \ {s}
    end if
end while
```

The function described in Algorithm 3 tries to decrease understaffing by locating understaffed shifts and then find employees that are available and can work on the shift. We order the employees in ascending order of scheduled working hours to improve the schedule of employees with too few scheduled hours.

Algorithm 4 Understaffing: Swap overlapping shifts

```
Input: Set of employees E
Input: Set of understaffed shifts S
for all e ∈ E do
    for all s ∈ e.shifts do
        O(s) ← {s' ∈ S : s' ∩ s ≠ ∅ ∧ e can work on shift s'}
        if O(s) ≠ ∅ then
            s* ← select the shift with the highest understaffing from O(s)
            e.shifts ← e.shifts \ s
            e.shifts ← e.shifts ∪ {s*}
        end if
    end for
end for
```

Algorithm 4 is used to decrease understaffing while making only small changes to the schedule. If we find an understaffed shift, we try to locate employees that are working on shifts that overlap with the understaffed shift. If the understaffed hours are not in the intersection of the two shifts, then moving the employee to the understaffed shift can decrease understaffing.

After we run Algorithms 3 and 4 to improve the understaffing, we use Algorithms 5 and 6 to improve the individual schedules of employees that are below the minimum

Algorithm 5 Staff below working hours: Add shifts

```
Input: Set of employees with scheduled working hours below minimum duty hours, E
Input: Set of shifts with staffing levels below maximum staffing, S
for all e ∈ E do
  for all s ∈ S do
    if e can work on shift s then
      e.shifts ← e.shifts ∪ {s}
      if Staffing level of s is at maximum staffing then
        S ← S \ {s}
      end if
    end if
  end for
end for
```

duty hours. Algorithm 5 selects an employee below duty hours and then tries to find a feasible shift with enough space for the employee.

Algorithm 6 Move shifts from employees above duty hours to employees below duty hours

```
Input: Set of employees with scheduled working hours above duty hours, E_above
Input: Set of employees with scheduled working hours below duty hours, E_below
for all Pairs (e_a, e_b) : e_a ∈ E_above, e_b ∈ E_below do
  for all s ∈ e_a.shifts do
    if e_b can work on shift s and e_a can be removed from shift s then
      e_a.shifts ← e_a.shifts \ {s}
      e_b.shifts ← e_b.shifts ∪ {s}
      if e_a is at or below duty hours then
        E_above ← E_above \ {e_a}
      end if
      if e_b is at or above duty hours then
        E_below ← E_below \ {e_b}
      end if
    end if
  end for
end for
```

Algorithm 6 is used to improve the balance of the employees duty hours. We create a set of employees above the duty hours and another set of employees that are below their duty hours. Then we look at all pairs of employees and try to find a shift that we can move from the above duty hours employee to the below duty hour employee. We can run Algorithm 6 either by allowing the algorithm to modify the requested shifts, or only allow modifications to shifts that were added by other modules of the algorithm.

If an employee is below duty hours, we can try to increase the number of working hours by swapping shifts. Algorithm 7 looks at employees below the minimum duty hours. For each employee, the module tries to swap an existing shift with a longer overlapping shift. Since the employee is already working on this particular day, such swaps are often feasible, assuming that the new shift has room for additional staff. Algorithm 7 can be allowed to change requested shifts, or we can focus only on shifts that have been added by other modules.

The experimental results in the next section are created by running Algorithms 1 to 7 in that order. Algorithms 6 and 7 were executed twice, first focusing only on shifts

Algorithm 7 Staff below working hours: Swap shifts to add working hours

```
Input: Set of employees with scheduled working hours below minimum duty hours, E
Input: Set of shifts with staffing levels below maximum staffing, S
for all e ∈ E do
  for all s ∈ e.shifts do
    O(s) ← {s' ∈ S : s' ∩ s ≠ ∅ ∧ e can work on shift s'}
    if O(s) ≠ ∅ then
      s* ← select the largest shift from O(s)
      if |s*| > |s| then
        e.shifts ← e.shifts \ {s}
        e.shifts ← e.shifts ∪ {s*}
        Update E
        Update S
      end if
    end if
  end for
end for
```

that had been added in previous steps, and then by allowing the modules to change requested shifts.

5 Experimental Results

To evaluate the performance of our algorithm, we use actual data from four companies and institutions. These companies and institutions include a nursing home, call centers and airport services. We will present the details of each problem instance and show examples of the preliminary schedule and the improved schedule. The scheduling period is usually 6 weeks, but we will plot the preliminary schedule and the improved schedule for only a single week for each problem instance. We tried to select a typical week for each instance. For each employee we measure the percentage of requested hours that are still in the improved schedule using the formula

$$\text{Requested hours granted} = \frac{\sum_{t=1}^{t=N} I_{prel}(t) \times I_{improved}(t)}{\sum_{t=1}^{t=N} I_{prel}(t)}$$

where N is the number of time slots in the scheduling period, $I_{prel}(t) = 1$ if the employee has requested to be working in time slot t, and 0 otherwise. Similarly the indicator function $I_{improved}(t)$ is equal to 1 if the employee is working in time slot t in the improved schedule and 0 otherwise. The percentage of requested hours is not defined for the employees that do not make any requests, so those employees are not included when we calculate the average requested hours granted.

We decided to use the number of hours instead of focusing on the shifts when measuring how close the final schedule is to the requested schedule from the employees. The reason is that we felt that if an employee has signed up for a $8 - 16$ shift, and we change it to $9 - 17$, then the employee is getting a shift that's very close to what was requested. Therefore, measuring hours is in our mind often more fair than focusing on shifts. However, this is not necessarily always the case, in some instances the measure should focus on the shifts instead of the hours. For the collaborating companies and institutions, the requested hours granted measure was considered both fair and appropriate.

	Preliminary schedule	Improved schedule
Scheduled hours	4669	5198
Man-hours overstaffed	478	220
Man-hours understaffed	777	21
Employees below minimum duty hours	3	0
Total unscheduled duty hours	905	545

Table 1 Results for the nursing home instance.

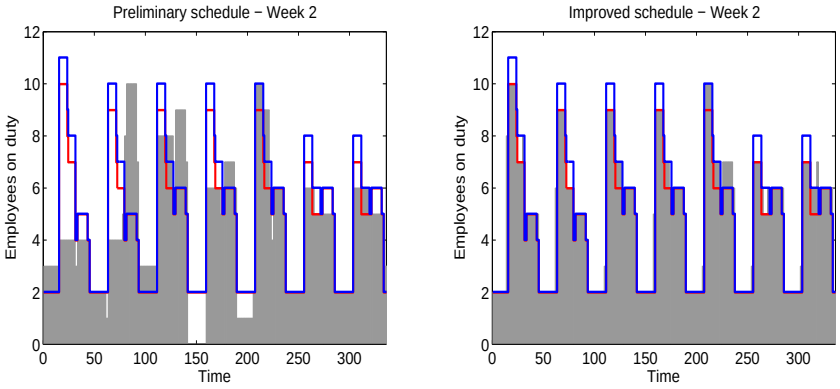


Fig. 1 Staffing levels for the nursing home problem instance. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

Using the notation introduced by De Causmaecker [9], we can describe the following problem instances as (AS|TVNO|PLGO).

5.1 Problem instance: Nursing home

The first problem instance comes from a nursing home with 55 employees. The scheduling period is 6 weeks with 30 minute intervals. There are 1428 different shifts that are allowed in the scheduling period. The length of each shift ranges from 4 hours up to 12 hours. If we sum up the total maximum working hours over the scheduling period, we get that the maximum number of hours that we can assign, without any overstaffing, is 5217 hours. However, the total duty hours of the employees is 5333 hours, so unless we violate the overstaffing constraint, we can never satisfy all duty hour requirements for the employees. The improved schedule has 5198 man-hours scheduled, not counting vacations and shifts that do not contribute to the staffing requirements. The results for the nursing home problem instance are shown in Table 1 while Figure 1 shows the preliminary schedule and the improved schedule for a typical week in the scheduling period.

The nursing home has the following hard constraints. An employee cannot work on more than 6 consecutive days, the maximum length of a shift is 9 hours while the minimum length of a shift is 4 hours. In any 24 hour period, each employee must get at least 8 consecutive hours of rest, while the maximum number of working hours in any 24 hour period is 9 hours.

	Preliminary schedule	Improved schedule
Scheduled hours	9424	11920
Man-hours overstaffed	390	791
Man-hours understaffed	1560	14
Employees below minimum duty hours	19	5
Total unscheduled duty hours	2108	234

Table 2 Results for call center A.

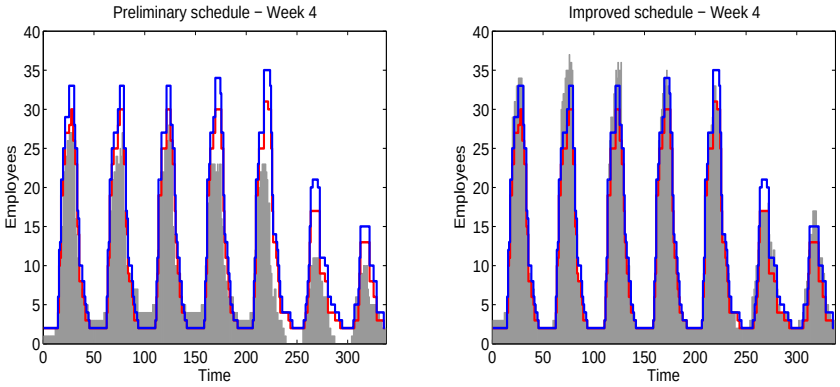


Fig. 2 Staffing levels for call center A. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

Figure 1 shows that the requirements for the minimum and maximum number of employees on duty have a very regular pattern. The nightshifts require 2 persons, while the mornings require between 8-10 people on duty. The number of on-duty employees decreases over the weekend, while Mondays require the highest number of on-duty employees. The preliminary schedule has both understaffing and overstaffing. However, in the improved schedule, the overstaffing has been reduced from a total of 478 hours to 220 hours, and the understaffing has gone from a total of 777 hours down to 21 hours. The average percentage of requested hours still in the improved schedule was 97.2%.

5.2 Problem instance: Call center A

The second problem instance is a call center. We have two call centers in our set of real world data so we will refer to them as call center A and call center B. Call center A has 92 employees and the scheduling period is 6 weeks in 30 minute intervals. The number of possible shifts in the scheduling period is 8863 and their lengths are from 4 hours up to 11 hours. The total maximum required on-duty employees is 11582, while the total duty hours for all employees is 12054, so it will be impossible to satisfy both the duty hour constraints and the overstaffing constraints. In the end, we actually schedule a total of 11920 hours, not including vacations and shifts that do not contribute to the staffing, so this instance has some overstaffing. Table 2 shows the data from the preliminary schedule and the results of the improved schedule. There are 5 employees

	Preliminary schedule	Improved schedule
Scheduled hours	6623	7554
Man-hours overstaffed	795	609
Man-hours understaffed	1306	189
Employees below minimum duty hours	15	7
Total unscheduled duty hours	1551	529

Table 3 Results for call center B.

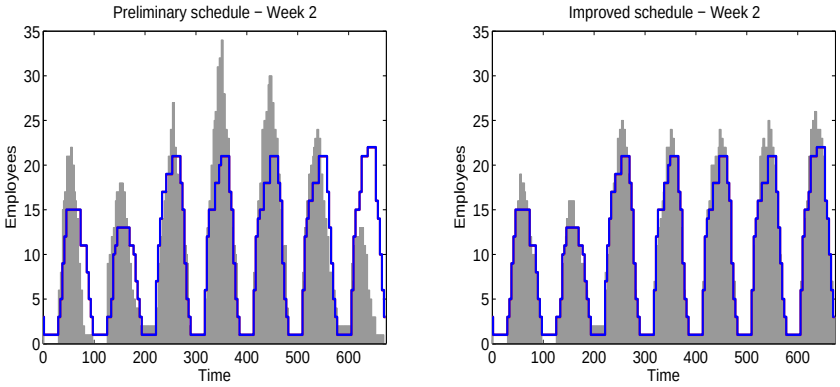


Fig. 3 Staffing levels for call center B. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

below the minimum duty hours in the improved schedule, but 3 of these employees are less than two hours below the minimum, while the remaining 2 employees are 6 hours and 22 hours below the minimum duty hours.

For call center A, the maximum number of consecutive working days is 6, the maximum number of working hours in each 24 hour period is 9 hours and the maximum length of a single shift is also 9 hours. In any 24 hour period, each employee must get at least 11 consecutive hours of rest.

Figure 2 shows a typical week in the scheduling period for call center A. The required number of on-duty employees peaks at around $30 - 35$ during the afternoon while the night shifts require only around $2 - 3$ on-duty employees. The preliminary schedule has both overstaffing and understaffing, while the improved schedule manages to almost eliminate the understaffing problem. However, the total number of overstaffed man-hours increases from 390 hours up to 791 hours. The increase in overstaffing is not surprising since having employees within the minimum and maximum duty hours has higher priority than overstaffing at this particular call center, and the call center has more staff than it needs to satisfy the maximum required on-duty personnel.

5.3 Problem instance: Call center B

Call center B does the planning for only 4 weeks in advance, but the schedule is created down to 15 minute intervals. This instance also does not use minimum and maximum number of employees that should be on duty in each interval, but specifies only the

	Preliminary schedule	Improved schedule
Scheduled hours	3997	6670
Man-hours overstaffed	192	641
Man-hours understaffed	2464	517
Employees below minimum duty hours	23	0
Total unscheduled duty hours	2087	0

Table 4 Results for airport ground services.

exact number of employees that should be on duty at each time. Since there is no flexibility in the required number of on-duty employees, the schedule is likely to have both overstaffing and understaffing as the algorithm tries to fit the number of employees to the exact number of required on-duty employees. The total number of employees at call center B is 62. The call center is overstaffed, the total available man-hours for the scheduling period is 8134 hours while the total required man-hours over the same period is 7134 hours. Table 3 shows the difference between the preliminary schedule and the improved schedule. In the improved schedule, there are still 609 man-hours of overstaffing and 529 unscheduled duty hours. The majority of this overstaffing and the unscheduled duty hours is due to the 1000 man-hour difference between the required man-hours and the available man-hours.

The hard constraints that must be satisfied for call center B are that employees cannot be working on more than 6 consecutive days, in every 24 hour period there must be at least 11 consecutive hours of rest and at most 11 hours of work. The maximum length of a shift is 11 hours while the length of a shift must be at least 4 hours.

Figure 3 shows a single week from the 4 week planning period. We see that there is still much overstaffing and understaffing in the improved schedule, but the overstaffing is more evenly distributed than in the preliminary schedule. One of the requests from the call center was that if overstaffing is necessary then it should be distributed as evenly as possible over the busiest periods. The improved schedule has problems with understaffing, the nightshifts for the first two nights in this particular week do not have anyone on duty, while the requirements call for at least one employee on duty at all time. The average percentage of requested hours still in the improved schedule is 86%. The reason for the low ratio of requested hours still in the improved schedule is mostly due to the fact that 12% of requested hours in the preliminary schedule were overstaffed and had to be changed.

5.4 Problem instance: Airport ground service.

The fourth problem instance is an airport ground service company. The scheduling period is six weeks in 30 minute intervals. The demand for on-duty employees depends on the flight schedules at the airport. In this particular instance, there are many flights that leave during the early morning, and then there is another concentration of flights in the afternoon. Since there are almost no flights scheduled at any time apart from the morning and afternoon busy periods, the requirements for employees peaks during the two busy periods but drops sharply during other times. The airport ground service has 53 employees. Due to the structure of the manpower requirements, the employees often work a short morning shift and then another short afternoon shift with a few hour break in-between. This problem instance is understaffed compared to the previous

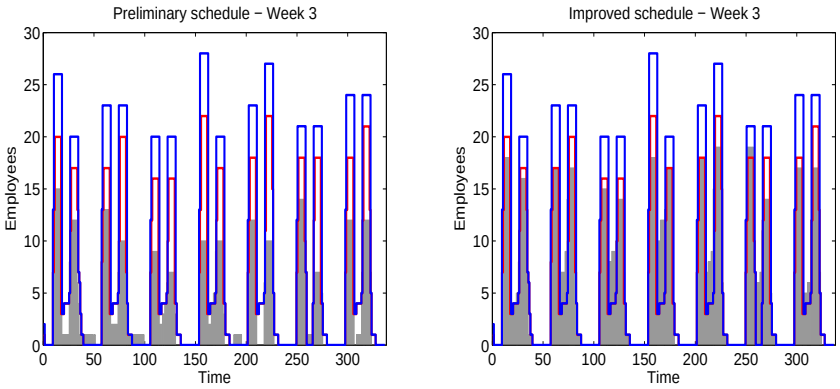


Fig. 4 Staffing levels for the airport ground services problem instance. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

examples, here the total maximum required man-hours is 8152 hours over the scheduling period while the available man-hours is only 6350. Table 4 shows the results of the improvements made to the preliminary schedule.

The hard constraints for the airport ground service are that there must be a minimum continuous rest of 11 hours in any 24 hour period, each employee can not work more than 5 consecutive days, employees cannot work more than 12 consecutive hours while the number of working hours in any 24 hour period is also 12 hours.

Table 4 shows that there are 641 hours of overstaffing in the improved schedule, even though the total available hours is much lower than the total maximum required hours. Figure 4, which shows the preliminary schedule and the improved schedule for a single week, gives an insight into why there is so much overstaffing. Due to the narrow peaks of busy periods in the morning and the afternoon, it's difficult to fit the employees exactly to the manpower requirements. The hours between the busy periods are often overstaffed due to employees working long shifts that cover both busy periods. The average percentage of requested hours that are in the improved schedule is 94%.

Figure 4 shows that the preliminary schedule is very understaffed. One of the reasons for the understaffing in the preliminary schedule is that out of 53 employees, 20 did not sign up for any shifts. However, the improved schedule does not have any employees under the minimum duty hours, so the algorithm managed to create feasible schedules for all the employees.

6 Conclusions

In this paper we have introduced an algorithm that is designed to emulate the behavior of staff managers when creating a high quality feasible staff schedule from a partial staff schedule based on requests from employees. Having a transparent system that the employees can trust is important since it encourages the employees to complete their own scheduling as well as possible, in the belief that the system will behave in a fair manner, while also completing the schedules for the employees that haven't completed their schedule.

Using real data from four different companies and institutions, we have shown that that the proposed algorithm does well in real world situations. The algorithm manages to decrease understaffing and make sure that the working hours of almost all employees are within the minimum and maximum bounds. Three of the examples that we presented have the problem of having more staff than the manpower requirements call for, so there is some overstaffing. The examples and the results demonstrate how difficult the staff scheduling problem is and highlight the challenge of maintaining a balance between overstaffing, understaffing, employee requests and the size of the staff versus the demand for employees. The presented algorithm is simple, transparent and easily understood, but still manages to perform well in our real world examples.

There is still more work that needs to be done, such as adding additional modules to the algorithm and analyzing the order in which they are executed. An ideal system based on this algorithm would have the option that each company would be able to define exactly in which order the modules are executed, in order to emulate exactly the behavior of the staff managers of the company. Being able to tailor-make the software for individual companies would allow for seamless implementation of such a staff scheduling system into companies that are currently spending time and effort into finding manual solutions to the staff scheduling problem.

References

1. J. Ahmad, M. Yamamoto, and A. Ohuchi. Evolutionary algorithms for nurse scheduling problem. *Proceedings of CEC00, San Diego*, pages 196–203, 2000.
2. U. Aickelin and K. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3(3):139–153, 2000.
3. J. F. Bard and H. W. Purnomo. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164, 2005.
4. M. J. Brusco and L. W. Jacobs. Cost analysis of alternative formulations for personnel scheduling in continuously operating organisations. *European Journal of Operational Research*, 86:249–261, 1995.
5. E. K. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. *SEAL98, LNCS 1585*, pages 187–194, 1999.
6. E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7:441–499, 2004.
7. E. K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Variable neighborhood search for nurse rostering problems. *Metaheuristics: computer decision-making*, pages 153–172, 2004.
8. E. K. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence special issue on Simulated Evolution and Learning, Springer*, 15(3):199–214, 2001.
9. P. De Causmaecker and G. Vanden Berghe. Towards a reference model for timetabling and rostering. *Annals of Operations Research*, 2010.
10. P. De Causmaecker, P. Demeester, and G. Vanden Berghe. Relaxation of coverage constraints in hospital personnel rostering. *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling*, pages 187–206, 2002.
11. P. De Causmaecker, P. Demeester, Y. Lu, and G. Vanden Berghe. Agent technology for timetabling. *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling*, pages 215–220, 2002.
12. K. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operations Research*, 106:393–407, 1998.
13. F. Easton and N. Mansour. A distributed genetic algorithm for employee staffing and scheduling problems. *Conference on Genetic Algorithms, San Mateo*, pages 360–367, 1993.
14. A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, February 2004.

15. J. P. Howell. Cyclical scheduling of nursing personnel. *Hospitals*, 40(2):77–85, 1966.
16. R. Hung. Improving productivity and quality through workforce scheduling. *Industrial Management*, 34(6), 1992.
17. S. Petrovic, G. Beddoe, and G Vanden Berghe. Storing and adapting repair experiences in employee rostering. *Selected Papers from PATAT, LNCS 2740. Springer-Verlag.*, pages 149–166, 2002.
18. J. Tanomaru. Staff scheduling by a genetic algorithm with heuristic operators. *Proceedings of CEC95*, pages 456–461, 1995.

An Evolutionary Algorithm in a Multistage Approach for an Employee Rostering Problem with a High Diversity of Shifts

Bäumelt Zdeněk · Šůcha Přemysl · Hanzálek Zdeněk

Abstract This work deals with the problem of rostering employees at an airport. There are about a hundred different shifts in order to handle the irregular coverage constraints. Together, with the strict constraints, given by the collective agreement, the problem becomes difficult to solve. Common one stage algorithms, applied to this problem, produce rosters containing too many isolated days-on and days-off which makes the roster unusable. This paper suggests a three stage approach for the employees rostering problem where a set of different shifts is needed to satisfy the coverage requirements. The solution is based on the problem transformation to a simpler problem, thereupon, an evolutionary algorithm is used to determine a rough position of the shifts in the roster. The maximal weighted matching in the bipartite graph is used as the inverse transformation of the problem and the final roster is obtained by the optimization based on a Tabu Search algorithm.

Keywords high diversity of shifts · multistage approach · evolutionary algorithm · employee rostering

1 Introduction

This paper deals with a problem from the traffic sphere belonging to the domain of employee timetabling/employee rostering/personnel scheduling problems (ETPs). The main difference to the classical Nurse Rostering Problem (NRP) [1], [14] is in the shift coverage demand. A typical NRP considers a small set of shifts, e.g. {day, night} [9] or {early, late, night} [13]. On the contrary, in

Z. Bäumelt · P. Šůcha · Z. Hanzálek
Department of Control Engineering, Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2, 166 27, Prague 6, Czech Republic
E-mail: baumezde@fel.cvut.cz, suchap@fel.cvut.cz, hanzalek@fel.cvut.cz

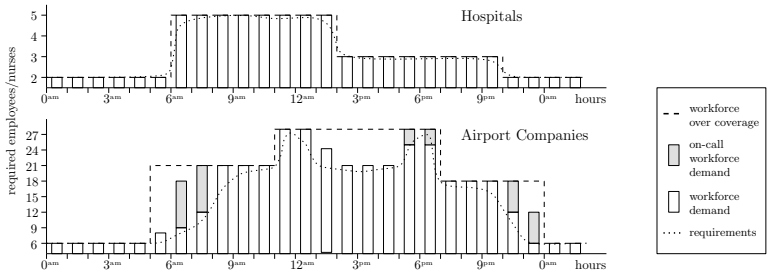


Fig. 1 The coverage function f_C examples

the ETP, motivated by real problems from the public transport (e.g. airport companies), the set of shifts can be quite large. This is caused by the fact that the coverage constraints are given by a so called *coverage function* f_C [6] determining the number of employees required each hour (see Fig.1). The f_C reflects the changes in the workforce demand during the day, that are caused by the traffic peaks and, in our case, it also fluctuates for different days and different seasons of the year. There are two basic possibilities how to cover peaks in the f_C. Either it can be fit using the 'classical' set of shifts, or it can be satisfied with an *extended set of shifts* with the size of dozens or hundreds of shifts. The extended set of shifts consists, not only, of shifts with different start and finish times, but also contains split shifts and on-call shifts. The split shifts facilitate coverage of the traffic peaks during the day, while the on-call shifts are used as an alternative for employees' sick leaves and other unanticipated causes. This approach makes the ETP more difficult, but it allows one to minimize the personnel expenses caused by the over coverage of the workforce. Two examples of f_C, corresponding to real data, are shown in Fig. 1. The first one is f_C typical for NRPs considering three shifts {early, late, night}. The second one is f_C typical for airport companies, where coverage requirements are depicted by a dotted curve. A dashed line corresponds to the over coverage of the workforce demand when a set of shifts with a small size is used. The gray blocks at the second f_C represent another feature – *on-call hours*. There are also *before shift on-calls*, when the employee is on the phone and comes to work one or two hours earlier if necessary. On the other hand, there are also *after shift on-calls* where the head of the ward decides whether the employee stays longer or leaves at the regular end of the shift. These before and after on-calls can resolve unpredictable changes in the f_C, e.g. traffic peak delays caused by bad weather conditions.

It is obvious that a high diversity of shifts is needed to cover the f_C in the second problem depicted in Fig. 1. We denote this problem as the Employee Timetabling Problem with a High Diversity of shifts (ETPHD).

The ETPHD is not only specific with a large variety of shifts but also through its set of constraints. The constraints that make this problem more complex are so called *block constraints*. These constraints are an extension

of the 'classical' constraints limiting the number of consecutive days and are described in detail in Section 2.1.

1.1 Related Works

Summaries of the approaches for solving problems from the timetabling/rostering domain are published in [1], [4]. The most reviewed part of the ETPs belongs to the health care branch [2], [3]. In the ideal case, i.e. a small set of shifts, a small set of employees and a simplified set of constraints, the problem can be solved by *Integer Linear Programming* (ILP) [9] leading to the optimal solution.

The ETPs can also be modeled as *Constraints Satisfaction Problems* (CSP), solved by constraint programming techniques [8]. A hybrid approach from the domain of the declarative programming was presented in [7] on a simplified NRP where the authors proposed an automatically implied constraint generation. Through this hybrid technique, the ratio of the solved NRPs can be increased. Furthermore, this approach allows one to discover non-solvable problems before search, for some instances.

It is impracticable to use the optimal approaches like ILP when more difficult ETPs are considered. In this case, heuristic approaches are applied or the solved ETP is separated into its subproblems. These two possibilities are sometimes joined together to attain suboptimal solutions.

One of the most applied metaheuristic approaches for ETPs is a *Tabu Search Algorithm* (TSA). A two stage approach is described in [10] where, in the first step, a feasible solution with respect to hard constraints is found and, in the second step, a TSA based optimization is used. Similar stage separation is described in [5] where the comparison of two approaches (TSA and *Memetic Algorithm* (MA)) for the optimization stage were introduced. In a general way, TSA is faster than MA, but its computation time considerably depends on the previous initialization stage.

The nearest problem to the problem described in this paper, from the coverage constraints point of view, is described in [6]. The coverage is expressed as a varying number of staff needed for each grade throughout the day. The presented method is a two stage approach, previously described in [11] where a MA is used in the initialization stage and a TSA is employed in the optimization stage. The time interval coverage constraints are met by different combinations of shifts applied in the second stage with TSA. However, the number of shifts (not their combinations) considered in this work for one grade is less than ten.

In terms of the NRP classification proposed in [12], the presented ETPHD can be categorized as ASBI|TVNO|PLGM.

1.2 Contribution and Outline

In this paper, we introduced a multistage approach for handling the ETPHD specified by the high diversity of shifts. The basic idea lies in a transformation of the extended set of shifts to a simpler one. The transformed timetable is initialized by an evolutionary algorithm (the first stage) and the problem instance is transformed back by an algorithm based on matching in the bipartite graph (the second stage). The objective of these stages is to determine the rough position of the blocks of shifts. The final roster is obtained during the optimization based on the TSA (the third stage). This stage uses our adaptation of the TSA suggested in [5]. The contributions of the paper are: a) a transformation allowing one to solve the ETPHD described in Sections 3 and 5, b) an ILP model presented in Section 3 and c) an algorithm for the first stage based on a evolutionary algorithm (EA) shown in Section 4.

The paper is organized as follows: Section 2 outlines the motivation problem at the airport. Section 3 explains the problem transformation to a problem with a reduced set of shifts and shows its ILP model. The transformed problem is solved in Section 4 by an EA. The inverse transformation is described in Section 5. Experiments and performance evaluation are summarized in Section 6 and the last section concludes the work.

2 Problem Statement

The problem solved in this paper is inspired by a real ETPHD from the traffic sphere. This problem is outstanding through its extended set of shifts where the shifts differ, not only, in the starting and finish times. There are also different split shifts and shifts prolonged by several before and after on-call hours.

The goal of the ETPHD is the same as in the NRP, to assign the requested shifts from the set of shifts to the employees with respect to the given constraints that are discussed below in detail.

2.1 Constraints

Constraints considered in the ETPHD are divided into two groups. The first group is stated as *hard constraints* that have to always be fulfilled. On the other hand, *soft constraints* can be violated, but their non-fulfillment is penalized in the objective function. Considering all mandatory rules of ETPHD, given by the labour code and the collective agreement, as hard constraints makes the problem over constrained. Therefore, the constraints separation is not firm in our approach, i.e. some hard constraints are considered as soft constraints with large penalties.

The hard constraints considered in this problem are:

(c_1) An employee cannot be assigned to more than one shift per day.

(c_2) Shifts requiring a certain grade has to be covered by employees with this grade.
(c_3) Over coverage of shifts is not allowed.
(c_4) Under coverage of shifts is not allowed.
(c_5) The minimal time gap of free between shifts must be kept.
(c_6) Personnel requests must be considered – like fixed shift assignment, day-off requests, partial day-off requests (e.g. an employee is able to work to 5pm).
(c_7) The maximum and minimum number of consecutive days-on and maximum hours have to be kept.
(c_8) The minimal block rest between the blocks have to be fulfilled.
(c_9) Valid blocks of shifts must be respected, e.g. no more than one split shift is allowed in the block.
(c_{10}) The minimal block rest after 2 consecutive night shifts must be kept, e.g. 70 hours free.

The constraint (c_5), which is considered differently than normal is remarkable for this restriction. It defines a *minimal time gap* between two shifts equal to 12 hours. This minimal time gap can be shortened down to 10 hours subject to a condition that the following minimal time gap will be prolonged by the time equal to the previous shortage. The hard constraint (c_6) keeps the *fixed shifts* in the roster from unacceptable assignments, e.g. a planned business trip or holidays must be respected.

What makes this ETPHD problem difficult are the hard constraints (c_7) and (c_8) dealing with the so called *block of shifts* (c_9). This block is defined as a sequence of consecutive shifts where block rest between each two shifts in the block does not exceed the defined *minimal block rest* covered by (c_8), e.g. 45 hours. The hard constraint (c_7) defines that the count of working shifts in each *block* is less than or equal to the maximal shift count. Likewise, the number of working hours in the block is limited. The last block constraint (c_9) limits the number of certain shifts in the block. These constraints make the situation more complex since the position of the blocks is crucial for the quality of the resulting schedule. Therefore, in our opinion, it rules out the majority of the single stage approaches since the rough position of the block should be determined in the first stage respecting the fixed shifts in the roster (e.g. planned holidays, planned business trips, etc.).

The soft constraints, considered in the ETPHD, are:

(c_{11}) The maximum number of shifts of a given type performed in the planning period should not be exceeded, e.g. a max. of 7 night shifts during 5 weeks.
(c_{12}) Overtime hours should be balanced according to an employee workload.
(c_{13}) Weekend and night working hours should be in balance according to an employee workload.
(c_{14}) Hours of a certain shift kind (early, late, night, split and on-call shifts) should be balanced.

In order to simplify the benchmarks, only the constraints (c_1), (c_3)–(c_9) and (c_{12}) are taken into account in the rest of the paper. The remaining constraints

(c_2), (c_{10}) and (c_{11}) can be easily incorporated into the mathematical model as well.

2.2 Problem Formalization

Let E be a *set of employees*, D represents a *set of days* from the whole planning period and S denotes the *extended set of shifts*. Consequently, the *roster* is represented by R, a binary matrix such that $\forall i \in E, \forall j \in D, \forall s \in S$

$$R_{ijs} = \begin{cases} 1, & \text{the shift } s \text{ is assigned to the employee } i \text{ on the day } j \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

When the roster contains a *fixed shift* s, defined due to (c_6), the corresponding $R_{ijs} = 1$ is a constant and another shift can not be assigned to this position.

The coverage constraints (c_3) and (c_4) from Section 2.1 are expressed by a binary matrix RS where $RS_{sj} = 1$ iff the shift $s \in S$ is required on the day $j \in D$. Subsequently, in relationship to constraint (c_5), we can define a binary matrix with *shift precedences* SP so that

$$SP_{s_1 s_2} = \begin{cases} 1, & \text{the shift } s_1 \text{ can be followed by } s_2 \text{ on the subsequent day} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $s_1, s_2 \in S$.

Even though there is a large number of different shifts, set S can still be joined into groups given by a mapping $\mathcal{M} : S \longmapsto K$ where $K = \{\mathcal{F}, \mathcal{H}, \mathcal{E}, \mathcal{L}, \mathcal{N}, \mathcal{S}, \mathcal{O}\}$ is a *set of shift kinds*. The set of shift kinds consists of {free $\mathcal{F}$, required free or holiday $\mathcal{H}$, early shifts $\mathcal{E}$, late shifts $\mathcal{L}$, night shifts $\mathcal{N}$, split shifts $\mathcal{S}$ and on-call shifts $\mathcal{O}$}. Let K_W, K_F and K_S be subsets of K defined as follows

$$\begin{aligned} K_W &= \{\mathcal{E}, \mathcal{L}, \mathcal{N}, \mathcal{S}, \mathcal{O}\} \\ K_F &= \{\mathcal{F}, \mathcal{H}\} \\ K_S &= \{\mathcal{S}, \mathcal{O}\}. \end{aligned} \tag{3}$$

In other words, K_W is a subset of *working shift kinds*, where K_F represents *free shift kinds*. The last subset K_S consists of *split shift kinds* and *on-call shift kinds*. Furthermore, for each $k \in K$, let L_k be an *average shifts length* of kind k so that $L_k = \text{avg}_{s \in S | \mathcal{M}(s) = k} |s|$ where $|s|$ is the length of the shift $s \in S$.

Finally, *workloads* of all employees E are defined by a non-negative vector W according to the length of planning period.

3 Mathematical Model of the Transformed Problem

The goal of the first stage of the algorithm is to design the rough position of the blocks where the shifts should be placed in an accordance to the given constraints. The *rough blocks* of shifts can be modeled as blocks of days-on separated by days-off. The output of the first stage, presented in this paper,

gives extra information in the form of which kind of shift $k \in K$ should be assigned on which day in the block. The first stage is described in this and the following section. This section presents a mathematical model based on a transformation used in the first stage.

3.1 Transformation $\mathcal{SK}$

Let $\mathcal{SK}$ be a transformation following from the mapping $\mathcal{M} : S \longmapsto K$. The $\mathcal{SK}$ transforms the ETPHD to ETPHD^K, specifically R to R^K where $R^K_{ijk} = 1$ iff the shift kind k is assigned to the employee i on the day j. In the same way, RS becomes RS^K where RS^K_{kj} is number of required shifts of kind k for the day j. Finally, SP becomes SP^K such that $SP^K_{k_1k_2}$ expresses, whether the shifts of kind k_1 can be followed by the shifts of kind k_2. The transformation is defined by equations (4), (5) and (6).

$$R^K_{ijk} = \begin{cases} 1, & \exists s \in S \mid R_{ijs} = 1 \wedge \mathcal{M}(s) = k \\ 0, & \text{otherwise} \end{cases}, \quad \forall i \in E, \forall j \in D, \forall k \in K \tag{4}$$

$$RS^K_{kj} = \sum_{s \in S \mid \mathcal{M}(s)=k} RS_{sj}, \quad \forall k \in K, \forall j \in D \tag{5}$$

$$SP^K_{k_1k_2} = \begin{cases} -1, & \forall s_1, s_2 \in S \;\middle|\; \begin{array}{l} SP_{s_1s_2} = 0 \\ \mathcal{M}(s_1) = k_1 \\ \mathcal{M}(s_2) = k_2 \end{array} \\ 0, & \forall s_1, s_2 \in S \;\middle|\; \begin{array}{l} SP_{s_1s_2} = 1 \\ \mathcal{M}(s_1) = k_1 \\ \mathcal{M}(s_2) = k_2 \end{array} \\ a, & \text{otherwise} \end{cases}, \quad \forall k_1, k_2 \in K \tag{6}$$

The positive penalty cost a of $SP^K_{k_1k_2}$ reflects the cases, when the precedence of the shift kinds $k_1, k_2 \in K$ is not obvious in general, but for most of the combinations of shifts $s_1, s_2 \in S \mid \mathcal{M}(s_1) = k_1 \wedge \mathcal{M}(s_2) = k_2$ is permitted, e.g. $SP^K_{\mathcal{L},\mathcal{E}}$.

3.2 Integer Linear Programming Model of ETPHD^K

The roster formed by rough blocks can be stated by an ILP model. The model uses a multicriteria objective function Z considering a linear combination of the constraints (c_3), (c_4) and (c_{12}) fulfillment, where $\alpha, \beta > 0$ are weights of the criterions. These criterions are evaluated by the piecewise linear functions (e.g. absolute value function) penRS and penW. The penRS function reflects

the over and under coverage of the assigned shift kinds, while the penW function corresponds to the coverage of the employees' workloads. These functions are represented in the ILP model by a set of auxiliary variables that are not incorporated into equations (7)–(13) in order to make the model more readable.

$$\min Z = \min \left\{ \alpha \cdot \sum_{k \in K_W} \sum_{j \in D} \text{penRS} \left(RS_{kj}^{K} - \sum_{i \in E} R_{ijk}^{K} \right) + \beta \cdot \sum_{i \in E} \text{penW} \left(W_i - \sum_{j \in D} \sum_{k \in K} L_k \cdot R_{ijk}^{K} \right) \right\} \tag{7}$$

subject to

$$\sum_{k \in K} R_{ijk}^{K} = 1, \quad \forall i \in E, \forall j \in D \tag{8}$$

$$R_{ijk_1}^{K} + R_{i,j+1,k_2}^{K} - SP_{k_1k_2}^{K} \leq 2, \quad \forall i \in E, \forall j = \langle 1, |D| - 1 \rangle, \forall k_1, k_2 \in K \tag{9}$$

$$\sum_{j=t}^{t+B_{maxL}} \sum_{k \in K_W} R_{ijk}^{K} \leq B_{maxL}, \quad \forall i \in E, \forall t = \langle 1, |D| - B_{maxL} \rangle \tag{10}$$

$$\sum_{k \in K_W} \left(R_{ijk}^{K} - R_{i,j+1,k}^{K} + R_{i,j+t,k}^{K} \right) \geq 0, \quad \forall i \in E, \forall t = \langle 2, B_{minL} \rangle, \forall j = \langle 1, |D| - t \rangle \tag{11}$$

$$\sum_{k \in K_W} \left(R_{ijk}^{K} - R_{i,j+t-1,k}^{K} + R_{i,j+t,k}^{K} \right) \leq 1, \quad \forall i \in E, \forall t = \langle 2, BR_{minL} \rangle, \forall j = \langle 1, |D| - t \rangle \tag{12}$$

$$\sum_{j=t}^{t+d} \sum_{k \in K_S} R_{ijk}^{K} \leq 1 + M \cdot \sum_{j=t}^{t+d} \sum_{k \in K_F} R_{ijk}^{K}, \quad \forall i \in E, \forall d = \langle B_{minL}, B_{maxL} \rangle, \forall t = \langle 1, |D| - d \rangle \tag{13}$$

The constraints of the ILP model are stated by equations (8) – (13). The first constraint equation (8) corresponds to the constraint (c_1), i.e. one shift per day is assigned. Similarly, equation (9) matches the constraint (c_5) represented by $SP_{k_1k_2}^{K}$.

The constraints (c_7), (c_8) are given by (10), (11), (12). A maximal block length B_{maxL} of the working shift kinds is constrained by (10), while the following equation (11) considers the minimal length of the blocks B_{minL}. The last inequality from the block constraints (12) defines the minimal block rest length BR_{minL} between the block of shifts. Since the ILP model of the first stage is formulated on shift kinds, constraints (10)–(12) consider the average shift

length L_k. Therefore, these equations limit the number of consecutive shift kinds instead of the sum of hours. Typical values for $(B_{maxL}, B_{minL}, BR_{minL})$ used in the solved ETPHD^K are $(5, 3, 2)$. In the last stage these constraints (c_7), (c_8) are reflected in the complete form.

The last equation (13) stands for (c_9) to avoid more shifts of the same kind in one block, e.g. it is not feasible to have more than one split shift, i.e. $k \in K_{\mathcal{S}}$, in one block. The term $M \cdot \sum_{j=t}^{t+d} \sum_{k \in K_F} R^K_{ijk}$ on the right side of (13) eliminates the equation in effect, when d consecutive days contain a free shift kind $k \in K_F$, i.e. it is not a block of the consecutive shifts. M is a big integer number.

4 Solution of the First Stage by an Evolutionary Algorithm

The transformation and the subsequent solution of the ILP model from Sectio 3.2 is the output of the algorithm's first stage. Due to enormous size of the ILP model it is not possible to find its feasible solution in a reasonable amount of time. Therefore, the solution of the first stage is found heuristically by the EA described below.

The roster is represented in the EA so that a couple of consecutive days are joined together. It reduces the overhead with the roster constraints' verification and the roster evaluation.

4.1 Modified Mathematical Model for the EA

The evolutionary algorithm, solving the first stage, uses the roster representation where the shift kinds assigned to the fixed number of consecutive days constitute a *gene*. The number of days representing the gene is called a *gene length* denoted as l. The consecutive genes are placed into *gene slots* GS indexed by p such that $p \in GS = \{\frac{j}{l} \mid j \in D \wedge (j \bmod l) = 0\}$. Thereafter, the gene $R^K[i,p]$ is a submatrix of R^K given by $R^K[i,p] = R^K_{ijk} \mid (p-1) \cdot l < j \leq p \cdot l$.

Both soft and hard constraints are taken into account as penalties in the objective function Z^E.

$$\begin{aligned} \min Z^E = \min \Bigg\{ & \alpha \cdot \sum_{k \in K_W} \sum_{j \in D} \mathrm{penRS}\left(RS^K_{kj} - \sum_{i \in E} R^K_{ijk} \right) + \\ & \beta \cdot \sum_{i \in E} \mathrm{penW}\left(W_i - \sum_{j \in D} \sum_{k \in K} L_k \cdot R^K_{ijk} \right) + \\ & \gamma \cdot \sum_{i \in E} \sum_{p \in GS} \mathrm{penUnsuit}\left(R^K[i,p] \right) + \\ & \delta \cdot \sum_{i \in E} \sum_{p \in \langle 1, |GS|-2 \rangle} \mathrm{penPrec}\left(R^K[i,p], R^K[i,p+1], R^K[i,p+2] \right) \Bigg\} \end{aligned} \tag{14}$$

The first two elements correspond to the objective function Z mentioned in (7), i.e. penalties of the under and over coverage of the required shift kinds and penalties of the unbalanced workload of the employees. The next two terms follow from the roster encoding. The third element of Z^E penalizes the suitability of the gene $R^K[i,p]$ given by the constraints expressed in (10)–(13).

The last element is focused on the gene precedences. It is necessary to take into account the borders of the genes after the recombination, i.e. the kind precedences SP^K and the block constraints related to the neighborhood genes have to be checked and updated. Furthermore, the other constraints, e.g. (c_{10}) can be easily appended through this criterion.

4.2 Evolutionary Algorithm

The `Preprocessing` function of the EA (shown in Alg. 1) contains the described transformation $\mathcal{SK}$. Consequently, in the `GeneratePopulation` function, all permutations with a repetition of shift kinds $k \in K$ of length l are generated and evaluated with respect to the considered constraints. Similarly, the precedences of genes are assessed with respect to (9)–(13). This static part of EA accelerates the evaluation of Z^E.

The roster R^K containing the genes $R^K[i,p]$ represents an individual $\mathcal{I}$ of a population $\mathcal{P}$. The initial population $\mathcal{P}_0$, containing $pSize_0$ individuals, is created randomly by the `GeneratePopulation` function in the following way. For each individual $\mathcal{I} \in \mathcal{P}_0$, the employees $i \in E$ are selected according to their count of the fixed shifts. The selection itself is similar to the rank selection of

Input : $ETPHD$ instance
Output: Roster R^K

0 $ETPHD^K \leftarrow$ `Preprocessing`$(ETPHD)$;
1 $\mathcal{P}_0 \leftarrow$ `GeneratePopulation`$(ETPHD^K, pSize_0)$;
2 **foreach** $\mathcal{I} \in \mathcal{P}_0$ **do** `Evaluate`$(\mathcal{I})$;
3 $\mathcal{P} \leftarrow \mathcal{P}_0$;
4 **while** *stop condition is not met* **do**
5 $\quad \mathcal{P} \leftarrow$ `Select`$(\mathcal{P}, pSize)$; `// select the` $pSize$ $\mathcal{I} \in \mathcal{P}$
6 $\quad \mathcal{P}_N \leftarrow \emptyset$; `// clear population` $\mathcal{P}_N$
7 $\quad$ **for** $i \leftarrow 1$ **to** $oCount$ **do** `// breed` $oCount$ `offsprings`
8 $\quad\quad [\mathcal{I}_1, \mathcal{I}_2] \leftarrow$ `ChooseParents`$(\mathcal{P})$;
9 $\quad\quad \mathcal{I}_{ofs} \leftarrow$ `Crossover`$(\mathcal{I}_1, \mathcal{I}_2)$;
10 $\quad\quad \mathcal{I}_{ofs} \leftarrow$ `Mutate`$(\mathcal{I}_{ofs})$ with probability p_M;
11 $\quad\quad \mathcal{P}_N \leftarrow \mathcal{P}_N \cup \mathcal{I}_{ofs}$; `// add offspring` $\mathcal{I}_{ofs}$
12 $\quad$ **end**
13 $\quad$ **foreach** $\mathcal{I} \in \mathcal{P}_N$ **do** `Evaluate`$(\mathcal{I})$; `// evaluate` $\mathcal{P}_N$
14 $\quad \mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_N$; `// merge populations`
15 **end**
16 $R^K \leftarrow \mathcal{I} \in \mathcal{P}$ with the lowest value of Z^E;
17 **return** R^K

Algorithm 1: An Evolutionary Algorithm pseudo-code

the EA. The top employees have the higher probability to be selected earlier than the others. When the employee is selected, it is necessary to choose the gene slot $p \in GS$ where the gene could be assigned. The gene slots selection is similar to the employees selection, i.e. according to the position of the fixed shifts in the slot $R^K[i,p]$.

The multicriteria objective function Z^E, given by (14), is used in the `Evaluate` function. The stop condition of the evolutionary algorithm is based on the ratio of the Z^E improvement to Z^E during the last 20 populations. If this ratio is under a given threshold, the algorithm stops.

The rank selection is used in the `Select` function to reduce the number of individuals of $\mathcal{P}$ to $pSize$. Furthermore, the applied selection keeps the population unique, i.e. $\forall \mathcal{I}_1, \mathcal{I}_2 \in \mathcal{P}$ holds $\mathcal{I}_1 \neq \mathcal{I}_2$. Finally, the elitism set is supplemented during the computation to keep the best $\mathcal{I} \in \mathcal{P}$ alive.

There are two basic possibilities, how to perform the `Crossover` function in the ETPHD^K. A uniform crossover in a horizontal dimension, i.e. with the certain probability p_{CH}, the better roster of employee i with respect to Z^E, is chosen from the individuals $\mathcal{I}_1, \mathcal{I}_2$. A typical value of $p_{CH} = 0.6$ makes the better roster from two individuals more favorable. An advantage is that there is no need to apply any repair operators for the horizontal crossover, because the whole rosters of the employees are copied to the offspring, i.e. all constraints related to precedences (kind precedence, gene precedence) are satisfied. The influence of the crossover to the 'vertical' constraints, e.g. shift kinds coverage, is incorporated to the objective function Z^E.

On the contrary, for a vertical dimension, it is better to apply the 1-point crossover instead of the uniform one. For each point of the crossover, it follows that the objective function Z^E increases rapidly due to precedence constraints violations. Then, the repair mechanism has to be applied to these violations at the point of the crossover. It is realized by the repeatedly applied mutation in order to fix all violated precedences of the genes. This repair mechanism is applied on each employee where the gene precedence is violated. Therefore the crossover in the vertical dimension is noticeably time consuming in a comparison to the rest of the evolutionary algorithm. For that reason, this type of crossover is suitable for the solution diversification only, when the algorithm is caught in a local optima.

The mutation operator is applied on the offspring $\mathcal{I}_{ofs}$ after the crossover a with probability p_M. The value of p_M is typically 0.2. The mutation is focused on randomly selected employees from the roster. For each employee, a few genes $R^K[i,p]$ are changed in a random way, too.

5 The Second Stage – Inverse Transformation $\mathcal{KS}$

The objective of the first stage is to determine the rough position of the blocks and to assign a shift kind $k \in K$ to each position of the block. The second stage transforms the roster back when the shift kinds K are substituted by the

required shifts $s \in S \mid RS_{sj} = 1$ for the day $j \in D$. This inverse transformation is based on the maximum weighted matching in a bipartite graph G_j where $j \in D$ is an index of the day.

For a day $j \in D$, let G_j be a bipartite graph with bipartition $V(G_j) = S(j) \cup E$, where $S(j)$ is a set of shifts required for the day j, so that $S(j) = \{s \in S \mid RS_{sj} = 1\}$. Furthermore, let $c : E(G_j) \to \mathbb{R}$ be the weights on the edges. There is an edge $(i, s) \in E(G_j)$ with $c\big((i, s)\big) = 1$ iff $\big(R^{K}_{ijk} = 1 \mid \mathcal{M}(s) = k \wedge k \in K_W\big) \wedge \big(SP_{s_{prev},s} = 1 \mid s_{prev} \in S(j-1) \wedge R_{i,j-1,s_{prev}} = 1\big)$ where s_{prev} is a shift assigned to $i \in E$ on the previous day. Moreover, there are also edges $(i, s) \in E(G_j)$ with lower weight $c\big((i, s)\big) = \epsilon(s, s_{prev}) < 1$ iff $\big(\exists k \in K_W \mid R^{K}_{ijk} = 1\big) \wedge \big(SP_{s_{prev},s} = 1 \mid s_{prev} \in S(j-1) \wedge R_{i,j-1,s_{prev}} = 1\big)$. These edges represent an assignment which is still possible but it is not preferred, i.e. $\mathcal{M}(s)$ is not the kind assigned to this position in the block. Thereafter, the weight $\epsilon(s, s_{prev})$ reflects how the shift s fits into the block when s_{prev} is placed on the day before.

The algorithm of the inverse transformation consecutively, for $j = 1$ to $j = |D|$, generates graphs G_j and looks for the maximal weighted matching M_W. When $(i, s) \in M_W$ the shift $s \in S(j)$ is assigned to the employee $i \in E$ on the day j, i.e. $R_{ijs} = 1$.

The result of the second stage (R) is optimized in the third stage which can be based on common techniques, e.g. a Tabu Search algorithm [5] or other heuristic approaches.

6 Experiments

The presented experiments show results obtained on real data from the airport. The three stage approach suggested in this paper is compared with a one stage algorithm presented in [5]. Our implementation of the algorithm described in [5] differs in the Tabu List implementation. In our realization the neighborhood of the swaps was excluded from the Tabu List representation since it does not provide better results on instances of ETPHD. The three stage approach uses the same algorithm in the third stage, but its execution is preceded by two initialization stages described in this paper. Weights in the objective functions were the same in both approaches. Both algorithms were implemented in C# and experiments were executed on a PC with Intel Core 2 at 2.4 GHz.

The mathematical model of the transformed problem contains $|E| \cdot |D| \cdot |K|$ binary and $|K| \cdot |D| + |E|$ continuous variables. Due to size of the model, the problem solution can be found in reasonable time for up to $|E| \leq 3$ using non-commercial ILP solver GLPK [15] and up to $|E| \leq 5$ using CPLEX [16]. Since the number of employees $|E|$ in our problem instance is more than one hundred the first stage cannot be directly solved by ILP.

The length of the gene used for our instances was tuned during the experiments and the best results were reached with the gene length $l = 4$. This

value correlates with the constants B_{maxL}, B_{minL} and BR_{minL} of the block constraints.

The computational results are summarized in Tab. 1. For each test period the table shows the period length $|D|$, the number of employees $|E|$, the number of shifts $|S|$ considered in the period and the ratio of the fixed shifts F. The approaches are compared on the number of unplaced shifts $|\overline{S}|$, the number of isolated days-on/days-off C_S and the number of two consecutive days-on C_D. The overall objective function improvement is presented in the last column ΔZ.

The last instance in Tab. 1 with the largest set of shifts and employees is remarkable. For this instance, a few of the shifts have not been assigned, but in the objective function and all other measures point of view, the obtained roster for this problematic planning period is much better. This period was made more difficult by the additional mandatory courses (reflected in the value of fixed assignments of shifts F) that all of the employees should have passed, if at all possible, during this period.

The CPU time of the three stage approach is higher due to the first stage of the algorithm where the EA determines the rough position of the shifts. The average CPU time of the first stage is around 5 minutes depending on the number of the fixed shifts. On the other hand, the extra 5 minutes given to the algorithm used in the one stage approach does not lead to a significant objective function improvement. The computation time of the second stage is negligible with respect to the first stage and it is smaller than 6 seconds.

7 Conclusion

In this paper, we introduced a three stage heuristic algorithm for the employee timetabling domain. The solved problem, motivated by a real employee rostering at the airport, is characterized in that the coverage function typically consists of two peaks and the level of the workforce demand differs from hour to hour by up to five employees. In order to satisfy the coverage requirements and to minimize the personnel expenses it is necessary to cover the requirements by an extended set of shifts. In our case, it is more than one hundred shifts. This fact together with the strict roster constraints, given by the collective agreement, makes the employee rostering very difficult. Therefore, the problem solution was decomposed into three stages, where the first one de-

Table 1 Experiments

period	$\|D\|$	$\|E\|$	$\|S\|$	F[%]	1 stage approach		3 stage approach		ΔZ[%]
					$\|\overline{S}\|$	C_S/C_D	$\|\overline{S}\|$	C_S/C_D	
nov09	28	90	79	33.71	16	37/25	2	9/8	15.5
dec09	28	90	82	32.16	13	35/21	0	8/6	20.3
jan10	35	90	89	32.97	12	39/22	0	9/6	17.9
mar09	35	94	103	41.64	24	43/28	5	11/9	13.8

signs a rough position of the shift kinds (i.e. early shifts, late shifts, etc.), the second stage assigns shifts into the roster and the final stage fine-tunes the final roster.

The experiments have shown that the common single stage approaches are not applicable on the ETPHD. The resulting roster suffers from the presence of isolated days-on and days-off, which makes the roster unusable.

On the contrary, the improvement of the objective function reaches about 15 percent with the proposed three stage approach. The rosters produced by our algorithm are more compact, the count of isolated days-on and days-off is suitable and all the required shifts are assigned to the roster.

Acknowledgements This work was supported by the Ministry of Education of the Czech Republic under the Research Programme MSM6840770038.

References

1. Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Landeghem, H.V.: The State of the Art of Nurse Rostering. Journal of Scheduling 7(6), 441-499 (2004)
2. Hung, R.: Hospital Nurse scheduling. Journal of Nursing Administration, 25(7/8), 21-23 (1995)
3. Cheang, B., Li H., Lim A. and Rodrigues B.: Nurse Rostering Problems – A Bibliographic Survey. European Journal of Operational Research, 151, 447-460 (2003)
4. Ernst, A.T., Jiang, H., Krishnamoorthy, M. and Sier, D.: Staff Scheduling and Rostering: A Review of Applications, Methods and Models. European Journal of Operational Research, 153(1), 3-27 (2004)
5. Berghe, G.V.: An Advanced Model and Novel Meta-heuristic Solution Methods to Personnel Scheduling in Healthcare, PhD Thesis, 276, University of Gent (2002)
6. Burke, E.K., De Causmaecker, P., Petrovic, S., Berghe, G.: Metaheuristics for Handling Time Interval Coverage Constraints in Nurse Scheduling. Applied Artificial Intelligence: An International Journal, 20(9), 743-766 (2006)
7. Wong, G.Y.C., Chun, A.H.W.: Nurse Rostering Using Constraint Programming and Meta-Level Reasoning. Engineering Applications of Artificial Intelligence, 17(6), 599-610 (2004)
8. Cheng, B.M.W., Lee, J.H.M. and Wu, J.C.K.: A Nurse Rostering System Using Constraint Programming and Redundant Modeling. IEEE Transactions on Information Technology in Biomedicine, 1, 44-54 (1997)
9. Azaiez, M.N., Al Sharif, S.S.: A 0-1 Goal Programming Model for Nurse Scheduling. Computers & Operations Research, 32(3), 491-507 (2005)
10. Burke, E.K., De Causmaecker, P. and Berghe G. V.: A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem. In: B. McKay et al., Editors, Simulated Evolution and Learning, Selected Papers from the 2nd Asia-Pacific Conference on Simulated Evolution and Learning, SEAL 98, Springer Lecture Notes in Artificial Intelligence, vol. 1585, Springer, 187-194 (1999)
11. Burke, E.K., Cowling, P., De Causmaecker, P. and Berghe, G.V.: A Memetic Approach to the Nurse Rostering Problem. Applied Intelligence 15, 199-214 (2001)
12. De Causmaecker, P.: Towards a Reference Model for Timetabling and Rostering. Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)
13. Valouxis, C. and Housos, E.: Hybrid Optimization Techniques for the Workshift and Rest Assignment of Nursing Personnel. Artificial Intelligence in Medicine, 20, 155-175 (2000)
14. Staff Rostering Benchmark Data Sets, http://www.cs.nott.ac.uk/∼tec/NRP/
15. GLPK (GNU Linear Programming Kit), http://www.gnu.org/software/glpk/
16. IBM ILOG CPLEX Optimizer, http://www.ilog.com/products/cplex/

List of Variables

f_C	coverage function
d, i, j, k, p, s, t	indices
$\alpha, \beta, \gamma, \delta$	positive weights of the objective function
Z	objective function
S	set of shifts
$S(j)$	set of shifts required on the day $j \in D$
D	set of days from the planning period
E	set of employees
K	set of shift kinds, i.e. early, late, night, etc.
L_k	average shift length of shift kind k
B_{minL}	minimal block length (in days)
B_{maxL}	maximal block length (in days)
BR_{minL}	minimal block rest length (in days)
M	big M – big integer number
K_W	set of working shift kinds
K_F	set of non-working shift kinds
K_S	set of split shift kinds
W	vector of employees workloads
R	binary matrix representing the roster, where $R_{ijs} = 1$ iff shift $s \in S$ is assigned to the employee $i \in E$ on day $j \in D$
RS	matrix of requested shifts, where $RS_{sj} = 1$ iff the shift $s \in S$ is requested on day $j \in D$
SP	matrix of the shift precedences, so that $SP_{s_1 s_2} = 1$ iff the shift $s_1 \in S$ can be followed by $s_2 \in S$ on the consecutive days
a	penalty cost of shift precedence $SP^K_{k_1 k_2}$
R^K	binary matrix representing the roster, where $R^K_{ijk} = 1$ iff shift of kind $k \in K$ is assigned to the employee $i \in E$ on day $j \in D$
RS^K	matrix of the requested shifts, where RS^K_{kj} is the number of the required shifts of kind $k \in K$ on day $j \in D$
SP^K	matrix representing the feasibility of two consecutive shifts precedence
$R^K\,[i, p]$	submatrix of R^K – a gene
l	gene length (in days)
$\mathcal{P}$	population in the evolutionary algorithm
$pSize, pSize_0$	size of the population, size of the initial population
$oCount$	count of the offsprings breed in each iteration of the EA
$\mathcal{I}$	individual of the population $\mathcal{P}$
p_{CH}	probability of the better roster acceptance in the horizontal crossover
p_M	probability of mutation

G_j	bipartite graph
$V(G_j)$	set of vertices of G_j
$E(G_j)$	set of edges of G_j
c, ϵ	weights of the edges $E(G_j)$
M_W	maximal weighted matching $M_W \subseteq E(G_j)$
C_S	count of isolated days-on/days-off
C_D	count of two consecutive days-on
F	ratio of the fixed shifts (count of fixed days divided by the count of all days in the roster)
$\overline{S}$	set of unplaced shifts

Network Flow Models for Intraday Personnel Scheduling Problems

Peter Brucker[1] and Rong Qu[2]

[1]Universität Osnabrück, Albrechtstr. 28a, 49069 Osnabrück, Germany, e-mail: pbrucker@uni-osnabrueck.de
[2]Automated Scheduling, Optimization and Planning (ASAP) Group, School of Computer Science, University of Nottingham, NG8 1BB, UK

ABSTRACT: Personnel scheduling problems can be decomposed into two stages. In the first stage for each employee the working days have to be fixed. In the second stage for each day of the planning period an intraday scheduling problem has to be solved. It consists of the assignment of shifts to the employees who have to work on the day and for each working period of an employee a task assignment such that the demand of all tasks for personnel is covered. Robinson et al. [3] formulated the intraday problem as a maximum flow problem under the following assumptions: employees are qualified for all tasks, their shifts are given, and they are allowed to change tasks during the day.

We show that the network flow model can be extended to cover the case in which employees are not qualified to perform all tasks. Further extensions allow to calculate shifts of employees for the given day under the assumption that an earliest starting time and a latest finishing time as well as a minimal working time are given. Also labour cost can be taken into account by solving a minimum cost network flow problem.

KEYWORDS: personnel scheduling, assignment problem, network flows

1 Introduction

A general personnel scheduling problem can be formulated as follows.

There is a planning horizon consisting of a number of consecutive days. Associated with each day is a set of periods in which certain tasks have to be performed. For each period of a day and task which has to be performed in this period employees are needed.

The planning horizon has to be divided into working days and rest days for each employee. A shift has to be assigned to each working day of an employee. *Shifts* consist of a set of *working periods* possibly interrupted by breaks and idle times which are part of the shift.

For each employee there is a set of tasks he can be assigned to.

A *working pattern* is defined by the set of working days and for each working day a shift. A working pattern is feasible for an employee if it satisfies a number of constraints.

One has to assign

- to each employee a feasible working pattern, and
- to each working period of this pattern a task to be performed by the employee.

This has to be done in such a way that

- all tasks can be performed (i.e. the demand of tasks for employees is satisfied), and
- corresponding costs are minimized.

The model has two levels which we denote by *days scheduling* and *intraday scheduling* level. At the days level one has to assign working days to employees while at the intraday level for each employee working on the day one has to assign a shift and to each working period of this shift a task for which the employee is qualified.

One can differentiate between preemptive and non-preemptive problems. In a preemptive problem employees may change the working place during a shift. This is not allowed in non-preemptive versions.

Robinson et al. ([3]) considered the personnel scheduling problem under the assumption that

- preemption is allowed, and
- each employee can perform each task.

They applied tabu search to find good working patterns for the employees, and given the working patterns they solved the problem of assigning tasks to the active periods of each employee by maximum flow algorithms.

A network flow model for a special non-preemptive personnel scheduling problem is discussed in [4].

This paper is organized as follows. The maximum flow model of Robinson et al. ([3]) is presented in Section 2, followed by the extended network flow model in Section 3. In Section 4 we present further extensions concerning demand and supply sides of the network model we build in Section 3. The last section contains concluding marks.

2 The maximum flow formulation of Robinson et al.

The intraday personnel scheduling problem of Robinson et al. ([3]) can be described as follows.

On each day a subset of employees is available. Each employee e working on a fixed day is available during some time window $[S_e, F_e[$. A shift of employee e is a time interval $[V_e, W_e[$ with $S_e \leq V_e \leq W_e \leq F_e$ and $W_e - V_e \geq m_e$ where m_e is a given minimal shift length. During each period within a shift the employee

performs a task, or has a (long or short) break, or is idle. There are maximal or minimal time distances between V_e, W_e ,the starting times, or finishing times of breaks. Breaks are non-preemptive.

There are n tasks $j = 1, \cdots, n$. Each task j has a duration p_j and must be processed by exactly one employee within a time window $[R_j, D_j[$ with $D_j - R_j \geq p_j$. Preemption is allowed, i.e. different employees may perform a task and an employee may perform different tasks on a day. Also interruption and later consideration of a task is possible. However, the total processing of task j must be equal to p_j.

Each employee can be assigned to any task.

One has to assign feasible shifts to the employees and for each shift to assign tasks to its active periods such that

- *the duration of each task is covered within its time window, and*
- *the total labor costs are minimal.*

The labor costs are defined as follows: meal breaks are unpaid. Short rest breaks are compensated. An overtime rate is paid for the time of a shift exceeding a given limit M. If an employee is not given at least two days off for a week then there is an additional pay.

Under the assumption that for each employee a shift has been fixed the problem can be formulated as a maximum flow problem with the following data.

Let T be the set of all R_j- and D_j- values, and all block starting and finishing times for all employees working on the day (blocks are maximal sets of consecutive working periods of a shift). Denote by $t_1 < t_2 < ... < t_s$ the ordered sequence of all elements in T.

The network (V, A) can be constructed as follows. The set V of nodes consists of

- task nodes $j = 1, \cdots n$,
- interval nodes $[t_i, t_{i+1}[$ $(i = 1, \cdots, s-1)$, and
- a source s and a sink t.

There are three different types of directed arcs:

- arcs (s, j) with upper capacity p_j,
- arcs $([t_i, t_{i+1}[, t)$ with upper capacity $(t_{i+1} - t_i)N_i$ where N_i is the number of employees available in time period $[t_i, t_{i+1}[$,
- there is an arc between a task node j and an interval node $[t_i, t_{i+1}[$ if and only if $[t_i, t_{i+1}[\subseteq [R_j, D_j[$. The upper capacity of this arc is $t_{i+1} - t_i$.

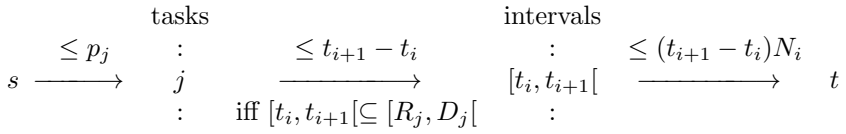


Figure 1: Network for the assignment of tasks to employees

The network is shown in Figure 1.

A flow in an arc $(j, [t_i, t_{i+1}[)$ may be interpreted as working time assigned to task j in the interval $[t_i, t_{i+1}[$. There exists a feasible task assignment if and only if the value of a maximal flow is equal to $\sum_{j=1}^{n} p_j$.

If there is a maximal flow with this property then in each task node j the processing time p_j is distributed to the time intervals $[t_i, t_{i+1}[$ in which j can be processed and the time j is processed in $[t_i, t_{i+1}[$ cannot exceed $t_{i+1} - t_i$. Furthermore, due to the flow-balance constraints in the interval nodes $[t_i, t_{i+1}[$ the sum of these processing times cannot exceed $(t_{i+1} - t_i)N_i$. It is well known (see e.g. [1] P. 108) that under these conditions it is possible to process the parts of tasks assigned to $[t_i, t_{i+1}[$ by N_i employees if preemption is allowed.

Robinson et al. describe a tabu search heuristic for calculating shifts for the employees for a time horizon of several days and corresponding assignments to tasks. The tabu search can be described as follows.

A working pattern of an employee consists of all shifts assigned to the employee within the time horizon. A solution consists of the working pattern of all employees. A solution is feasible if it allows to cover the demand of all tasks on every day. Feasibility can be checked and task assignments can be calculated by solving a maximum flow problem for each day. The search is performed within the set of all feasible solutions.

The assumption that each employee can be assigned to any task is not always realistic. Therefore the model will be extended in the next section.

3 An extended network flow model

In this and later sections the assumption that employee e can perform only tasks $j \in Q_e \subseteq \{1, \cdots, n\}$ is added. A network which takes care of these additional constraints can be described as follows.

Again $t_1 < t_2 < ... < t_s$ are the time instances where the data are changing. The set of nodes of the network consists of

- task nodes $j = 1, \cdots, n$,
- interval-task nodes $[t_i, t_{i+1}[_j$ for all intervals $[t_i, t_{i+1}[$ with $[t_i, t_{i+1}[\subseteq [R_j, D_j[$,
- interval-employee nodes $[t_i, t_{i+1}[_e$ for all working intervals $[t_i, t_{i+1}[$ of employee e, and
- a source s and a sink t.

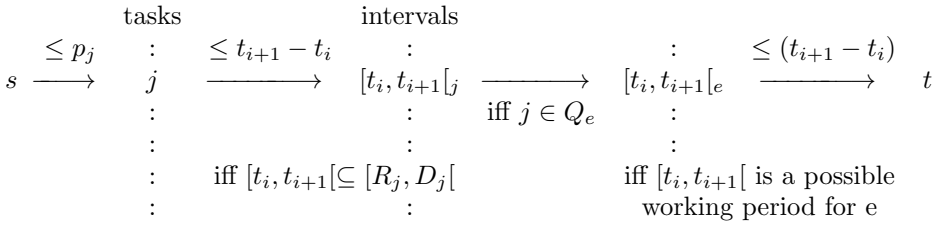


Figure 2: Extended nerwork

There are four different types of arcs:

- arcs (s, j) with upper capacity p_j,
- arcs $(j, [t_i, t_{i+1}[_j)$ with upper capacity $t_{i+1} - t_i$,
- arcs $([t_i, t_{i+1}[_j, [t_i, t_{i+1}[_e)$ for $j \in Q_e$, and
- arcs $([t_i, t_{i+1}[_e, t)$ with upper capacity $t_{i+1} - t_i$.

The network is shown in Figure 2. A flow in an arc $([t_i, t_{i+1}[_j\ ,\ [t_i, t_{i+1}[_e)$ may be interpreted as the number of time units employee e is assigned to task j within the time interval $[t_i, t_{i+1}[$. The flow conservation constraint for node $[t_i, t_{i+1}[_j$ distributes the time spent on task j in $[t_i, t_{i+1}[$ among employees which are qualified to do task j. The flow conservation constraint for node $[t_i, t_{i+1}[_e$ limits the workload of employee e in $[t_i, t_{i+1}[$ by $t_{i+1} - t_i$. There exists a feasible assignment of employees to tasks if and only if the maximum flow is equal to $\sum_{j=1}^{n} p_j$.

The procedure is illustrated by the following example with two employees and three tasks.

Example 1 Consider a problem with the following data. Notice that in the time interval $[2, 3[$ employee e_1 has a break.

task j	1	2	3
R_j	0	3	4
D_j	6	7	6
p_j	4	2	2

employee e_i	shift	Q_i
e_1	$[0, 2[\,, [3, 6[$	$\{1, 2\}$
e_2	$[3, 7[$	$\{2, 3\}$

The corresponding network with a solution is presented in Figure 3. Figure 4 shows the Gantt chart of the solution. The relevant t_i values are $0, 2, 3, 4, 6, 7$. Employee e_2 is idle in period $[3, 4[$.

4 Further extensions

The model introduced in the previous section can be extended at the demand side and/or the supply side. Possible extensions will be discussed in this section.

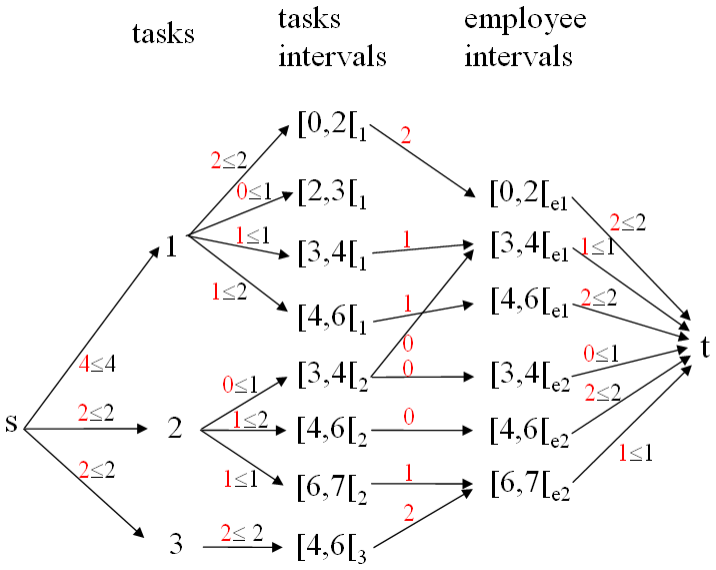


Figure 3: Network flow of Example 1

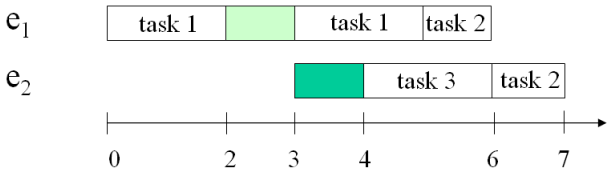


Figure 4: Gantt chart of the solution for Example 1

4.1 Extensions at the demand side

Instead of forcing the processing time of each task j to be equal to p_j by solving a corresponding maximum flow problem it is possible to enforce the constraint $LP_j \leq p_j \leq UP_j$ by the lower bound LP_j and the upper bound UP_j for the flow in the arc (s, j). In this case one has to find a feasible solution. If additionally costs are assigned to the arcs $([t_i, t_{i+1}[_e, t)$ one could minimize labour costs by solving a corresponding minimum cost network flow problem.

Another option is to replace

$$s \xrightarrow{\leq p_j} j \xrightarrow{\leq t_{i+1} - t_i} [t_i, t_{i+1}[_j$$

by

$$s \xrightarrow{\leq p_{ij}\,(t_{i+1} - t_i)} [t_i, t_{i+1}[_j$$

where p_{ij} is the number of employees needed for task j in the time interval $[t_i, t_{i+1}[$. Again one has to solve a maximum flow problem to cover the demand. Also by lower and upper bounds on the arcs $(s, [t_i, t_{i+1}[_j)$ the constraints $LD_{ij}\,(t_{i+1} - t_i) \leq p_{ij}\,(t_{i+1} - t_i) \leq UD_{ij}\,(t_{i+1} - t_i)$ can be enforced.

4.2 Extensions at the supply side

Instead of fixing the shift of employee e in advance one could fix only the availability interval $[S_e, F_e[$ and a minimal working time m_e for employee e. Then shifts for the employees which cover the demand of tasks can be calculated. To achieve this one has to replace

$$[t_i, t_{i+1}[_e \xrightarrow{\leq t_{i+1} - t_i} t$$

by

$$[t_i, t_{i+1}[_e \xrightarrow{\leq t_{i+1} - t_i} e \xrightarrow{\geq m_e} t$$

Due to node e and arc (e, t) the total working time of employee e cannot be smaller than m_e.

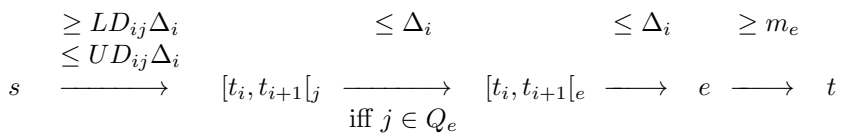


Figure 5: Combined extensions

4.3 Combined extensions

The extensions at the demand and supply side can be combined. A possible combination is shown in Figure 5 where $\Delta_i := t_{i+1} - t_i$. A feasible network flow solution corresponds to a feasible shift and task assignment. Also overtime costs can be taken into account by assigning these overtime costs to the arcs (e, t), zero costs to all other arcs, and by solving the corresponding minimum cost network flow problem.

5 Concluding remarks

In this note we have shown that the problem of assigning shifts to employees and employees to tasks to cover the demand can be efficiently solved by network flow algorithms if preemption is allowed, even if employees are not qualified for all tasks. This can be exploited in heuristics for personnel scheduling problems for a time horizon of several days.

However, a side effect is that employees have to switch between tasks (working places) during their shifts. These switches depend on the constraints under which shifts are calculated and may be unavoidable. In connection with this the following working place change minimization (WPCM-) problem is of interest: *Assume that shifts have been assigned to all employees working on a given day. Then we call a task assignment for these employees feasible if the demand of all tasks for employees is covered. Find a feasible assignment which minimizes the number of working place changes.*

In [2] it has been shown that the WPCM-problem is NP-hard if possible shifts for e have the form $[t, t + p_e[$ $(t = 0, \cdots, P - p_e)$ where P is the number of working periods of the day. The complexity of the WPCM-problem for other ways of shift assignments is unknown.

Based on the present on the network flow models, extended investigations will be carried out in our future work to develop heuristic algorithms which assign feasible shifts to employees and construct (directly) preemptive schedules taking care of working place changes (e.g. by constructing good shifts). Numerical results will be reported and analysed on solving real world problems.

References

[1] Brucker, P. (2007), Scheduling Algorithms, Springer, Berlin.

[2] Brucker, P., Qu, R., Burke, E. (2008). Personnel scheduling: models and complexity, Working Paper, Automated Scheduling, Optimization and Planning (ASAP) Group, School of Computer Science, University of Nottingham.

[3] Robinson, R., Sorli, R., Zinder, Y. (2005), Personnel scheduling with time windows and preemptive tasks, In: E. K. Burke and M. Trick (editors), Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, 18th August - 20th August 2004, Pittsburgh, PA USA: 561-566

[4] Segal, M. (1974), The operator scheduling problem: a network flow approach, Operations Research 22: 808-823.

Round-Robin Tournaments with homogenous rounds

Bregje Buiteveld · Erik van Holland · Gerhard Post[⋆] · Dirk Smit

Abstract We study round-robin tournaments for n teams, where in each round a fixed number (g) of teams is present and each team present plays a fixed number (m) of matches in this round. In the tournament each match between two teams is either played once or twice, in the latter case in different rounds. We give necessary combinatorial conditions on the triples (n, g, m) for which such round-robin tournaments can exist, and discuss three general construction methods that concern the cases $m = 1$, $m = 2$ and $m = g - 1$. For $n \leq 20$ these cases cover 149 of all 173 non-trivial cases that satisfy the necessary conditions. Of these 149 cases a tournament can be constructed in 147 cases. For the remaining 24 cases the tournament does not exist in 2 cases, and is constructed in all other cases. Finally we consider the spreading of rounds for teams, and give some examples where well-spreading is either possible or impossible.

1 Introduction

The most common form for sports competitions is the round-robin tournament. In such tournaments the number of matches between each pair of teams is the same: in a *single* round-robin tournament this number of matches is 1, while in a *double* round-robin tournament it is 2. If the competition contains n teams, there are $\frac{1}{2}n(n-1)$ matches in a single round-robin, and $n(n-1)$ matches in a double round-robin. Often the location of the match is important. In this case a match is *home* for one team, while it is *away* for the other team.

If the matches are divided in *rounds*, the rounds are ordered in time, and a team plays at most in one match per round, we have the notion of break. We say that a team

⋆ This research has been supported by the Netherlands Organisation for Scientific Research, grant 639.033.403, and by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

Bregje Buiteveld · Erik van Holland · Dirk Smit
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Gerhard Post
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands and
ORTEC, Groningenweg 6k, 2803 PV Gouda, The Netherlands

has a *break* between two of its consecutive matches, if it plays in both matches home, or in both matches away. It is well-known ([8,5,6]) that if n is even, and there are $n-1$ rounds with each team playing a match, then there are at least $n-2$ breaks. For n odd, the usual round-robin tournament contains n rounds with $n-1$ teams present, playing one match. In this case we can find a round-robin tournament without breaks, and even more remarkable this schedule is essentially unique (see [7]).

In some competitions the round-robin tournament has a different structure for the rounds: in a round only a (small) subset of all teams play one or more matches. The reason for this format can be two-fold: the match is played in a sports hall or court, which is rented for a (part of the) day by the league (see also [13,9]) or the teams prefer to play twice or more on the same day; this preference is encountered regularly for youth teams. While often in round-robin tournaments breaks are an important issue, this is obviously not the case in this situation.

Our main motivation was provided by the Dutch Inline Skater Hockey League, where in each round 3 or 4 teams play once against each other, leading to 3 or 6 matches in a round. Note that scheduling a competition in this form gives the competition organizer the possibility to cope with problems due to unavailabilities of teams, an aspect important for the Dutch Inline Skater Hockey League, where teams can indicate what dates they are not available. Hence the construction of the round-robin tournaments in this case is in parallel to the construction of basic match schedules (see [12]) in the usual round-robin tournaments: we make a schedule for dummy teams which are matched to the real teams in a second phase leading to the fixture list.[1]

This is the setting we will study: the round-robin tournament for n teams is divided in rounds which always contain a subset of teams of the same size (g), and all teams play an equal number (m) of matches in this round. The schedule in this form we will call an (n, g, m)-tournament. If necessary we will attach 'single' or 'double'; for example a double $(5, 4, 2)$-tournament.

In Section 2 we will derive some necessary conditions for the existence of an (n, g, m)-tournament. In the Section 3 and 4 we will give our results for the single round-robin tournaments, while Section 5 will discuss the double round-robin tournaments. In Section 6 we will give some case studies on the well-spreading of (n, g, m)-tournaments.

In this work we mainly investigate the existence of (n, g, m)-tournaments by using different 'standard' techniques. The techniques used apply to any number (n) of teams. We tried to be exhaustive for 'small' n, namely $n \leq 20$: it is seldom that one encounters competitions with more than 20 teams. Our standard techniques settle the existence of most (n, g, m)-tournaments for $n \leq 20$. Most remaining cases were solved by hand, though we made an ILP formulation as well, and used CPLEX 11.2 to establish the existence of some double round-robin tournaments. The solutions constructed are usually far from unique; we noted that the solutions constructed by hand are usually much more regular than those found by CPLEX; examples of this can be found in Section 4.

[1] The dates represent the times that a round can be played. For most competitions of the Dutch Inline Skater Hockey League there are more dates than rounds, leading to even more possibilities for the competition organizer.

2 Conditions for the existence of (n, g, m)-tournaments

We consider round-robin tournaments for n teams, where in each round a subset of g teams play m matches. There are two cases that we call trivial, and which we skip in all our further considerations:

1. The case $g = 2$ and $m = 1$. In this case each match represents a round.
2. The case $g = n$ and $m = n-1$. In this case a round is a complete single round-robin tournament.

For the remaining cases we can state integrality conditions for the existence of an (n, g, m)-tournament. This is formulated in the following theorem.

Theorem 1 *Let S be an (n, g, m)-tournament for a round-robin tournament. Let $d = 1$ if S is a single round-robin tournament, and $d = 2$ if S is a double round-robin tournament. Then the parameters n, g, m and d have the following relations.*

(a) g is even or m is even.
(b) $dn(n-1)$ is a multiple of gm.
(c) $d(n-1)$ is a multiple of m.

Proof.

1. The number of matches in a round is $\frac{1}{2}gm$. Hence either g or m must be even.
2. The number of rounds in tournament S is $\frac{\frac{1}{2}dn(n-1)}{\frac{1}{2}gm}$, i.e. the total number of matches divided by the number of matches per round. Since this fraction is integer, we have that $dn(n-1)$ is multiple of gm.
3. The number of rounds for a team is $\frac{d(n-1)}{m}$. Hence $d(n-1)$ needs to be a multiple of m. □

Theorem 1 restricts the number of possible combinations considerably. Table 2 gives all possible combinations (g, m) for $n \leq 20$ in the single round-robin case, 80 cases in total. For example, for $n = 7$ there exist at most 3 single $(7, g, m)$-tournaments, namely for $(g, m) = (3, 2)$, $(g, m) = (6, 1)$, and $(g, m) = (7, 2)$. A similar table for double round-robin tournaments can be found in Section 5.

At this point it is appropriate to discuss a graph theoretical setting for the single round-robin tournaments. The existence of a single (n, g, m)-tournament is equivalent to a decomposition of the complete graph K_n in m-regular subgraphs on g vertices, such that all these subgraphs are edge-disjoint. There are some special cases of interest:

1. If $g = n$ and $m = 1$ this decomposition is called a *1-factorization*. Necessarily n is even. This case corresponds to the regular round-robin tournament for n teams.
2. If $m = 2$ we have a decomposition of K_n in cycles. In our case the vertices in the components add up to g. If we require that all cycles have the same length g the decomposition is called a *cycle decomposition*.
3. If $m = g - 1$ we have that the teams present in a round all play against each other. Consequently the $(n, g, g-1)$-tournament is a collection of subsets of $\{1, 2, \ldots, n\}$, all of size g, such that each pair of teams is member of exactly 1 subset. Such collection of subsets is an $(n, g, 1)$-*(block) design*, see for example [3].

In the next section we will describe the results based on these 'standard constructions', and some variations. Section 4 is devoted to the cases for which no standard construction exists.

n	(g,m)								
4	(4,1)								
5	(4,1)	(5,2)							
6	(6,1)								
7	(3,2)	(6,1)	(7,2)						
8	(4,1)	(8,1)							
9	(3,2)	(4,1)	(4,2)	(6,1)	(6,2)	(6,4)	(8,1)	(9,2)	(9,4)
10	(6,1)	(6,3)	(10,1)	(10,3)					
11	(5,2)	(10,1)	(11,2)						
12	(4,1)	(6,1)	(12,1)						
13	(3,2)	(4,1)	(4,3)	(6,1)	(6,2)	(12,1)	(13,2)	(13,4)	(13,6)
14	(14,1)								
15	(3,2)	(5,2)	(6,1)	(7,2)	(10,1)	(10,7)	(14,1)	(15,2)	
16	(4,1)	(4,3)	(6,1)	(6,5)	(8,1)	(8,3)	(8,5)	(10,1)	(10,3)
	(12,1)	(12,5)	(16,1)	(16,3)	(16,5)				
17	(4,1)	(4,2)	(8,1)	(8,2)	(16,1)	(17,2)	(17,4)	(17,8)	
18	(6,1)	(18,1)							
19	(3,2)	(6,1)	(6,3)	(9,2)	(18,1)	(19,2)	(19,6)		
20	(4,1)	(10,1)	(20,1)						

Table 2. Parameters (g,m) for which a single (n,g,m)-tournament can exist.

3 Standard constructions for single round-robin tournaments

In the previous section we described three special cases, which we will call 'round-robin' construction, 'cycle' construction, and 'block-design' construction. We will describe the results of each of these combinatorial constructions in a separate subsection. Before turning to these constructions, we formulate a general construction, the merging of rounds. Note that this theorem is valid for both single and double round-robin tournaments.

Theorem 2 *Suppose an (n,g,m)-tournament exists with R rounds, where $g=n$, and m' is a multiple of m, as well as a divisor of mR. Then an (n,g,m')-tournament exists as well.*

Proof
Starting with an (n,n,m)-tournament, we can unite groups of m'/m rounds to obtain the (n,n,m')-tournament. □

3.1 Round-robin constructions

The constructions for $(n,n,1)$-tournaments (n even) and $(n,n-1,1)$-tournaments are well-known ([8,6]). We will call these tournaments the *regular* round-robin tournaments (single or double) for the given number of teams. Based on a regular tournament we can construct other tournaments as well. The next theorem is valid for single and double round-robin tournaments.

Theorem 3 *Let M be the number of matches in the round-robin tournament. An $(n,g,1)$-tournament exists in the following cases:*

(a) For n even, g even, and g a divisor of n.
(b) For n odd, g even, and g is divisor of $n-1$.

(c) For n even, g even, $\frac{1}{2}g$ a divisor of M, and $g \leq \frac{1}{2}n + 1$.
(d) For n odd, g even, $\frac{1}{2}g$ a divisor of M, and $g \leq \frac{1}{2}(n + 1)$.

Proof

(a) Starting with a regular $(n, n, 1)$-tournament, we can split the $\frac{1}{2}n$ matches of a round in groups of $\frac{1}{2}g$ matches each. All the matches in a round contain different teams, hence we end up with an $(n, g, 1)$-tournament.
(b) The proof is similar to case (a), starting with a regular $(n, n-1, 1)$-tournament.
(c) The construction resembles case (a), but we have to do a little more work. Consider a regular $(n, n, 1)$-tournament, and order the matches in this schedule according to the rounds. What we need to do is to refine this ordering such that picking the next $\frac{1}{2}g$ matches each time, results in an $(n, g, 1)$-tournament. (Note that the number of matches M is required to be a multiple of $\frac{1}{2}g$.) To construct the next round in the $(n, g, 1)$-tournament, we pick the next $\frac{1}{2}g$ consecutive matches: these fall within round (say) r and round $r + 1$. What could happen is that the matches in round $r + 1$ contain the same teams as those selected from round r. We show that the order of the matches can be adjusted to avoid this. Suppose k matches are in round r: these k matches contain $2k$ teams. Hence there are at most $2k$ matches in round $r + 1$ with those teams. Consequently there remain at least $\frac{1}{2}n - 2k$ matches in round $r + 1$ that can be used in the $(n, g, 1)$-tournament. The number of matches in round $r + 1$ is $\frac{1}{2}g - k$. Hence we need $\frac{1}{2}g - k \leq \frac{1}{2}n - 2k$, or $\frac{1}{2}g + k \leq \frac{1}{2}n$. Since $k < \frac{1}{2}g$, and $g \leq \frac{1}{2}n + 1$, this condition is satisfied.
(d) The analysis is similar to case (d), starting with a regular $(n, n-1, 1)$-tournament. □

Theorem 3 settles the existence of 34 parameter combinations in Table 2, namely all cases with $m = 1$ except the cases $(9, 6, 1)$, $(15, 10, 1)$, $(16, 10, 1)$, and $(16, 12, 1)$. Applying Theorem 2 yields the cases $(10, 10, 3)$, $(16, 16, 3)$, and $(16, 16, 5)$.

3.2 Cycle constructions

The decomposition of graphs by cycles of fixed length has been studied extensively. This research lead to the end result for complete graphs [1,11], stating that the obvious necessary conditions are also sufficient, see Theorem 4.

Theorem 4 *Suppose n is odd and $3 \leq g \leq n$, such that g divides $\frac{1}{2}n(n-1)$. Then the complete graph K_n can be decomposed in edge-disjoint cycles of length g.* □

Comparing this sufficient condition with the necessary conditions in Theorem 1, we see that for the existence of $(n, g, 2)$-tournaments the necessary conditions are sufficient. Hence all 22 $(n, g, 2)$-tournaments in Table 2 exist. Using Theorem 2 we get additionally the existence of all 6 (n, n, m)-tournaments for n odd and $m > 2$ even. Note that the results in [11,1] use cycles of fixed length. This can be used to obtain the following construction.

Theorem 5 *Suppose n is odd, g is even and $\frac{1}{2}g$ divides n. Then a single $(n, g, 1)$-tournament exists.*

Proof
According to Theorem 4 the complete graph K_n can be decomposed in cycles of length n. Choose in each of these cycles groups of $\frac{1}{2}g$ non-adjacent edges. Letting these edges

correspond to the matches of a round between the corresponding nodes, we obtain the required $(n, g, 1)$-tournament. □

This construction yields several of the $(n, g, 1)$-tournaments constructed before. In addition it yields the existence of a single $(9, 6, 1)$-tournament and a single $(15, 10, 1)$-tournament.

3.3 Block design constructions

As explained in Section 2 we can use block designs for the construction of $(n, g, g-1)$-tournaments. We formulate this equivalence in the following theorem.

Theorem 6 *A single $(n, g, g-1)$-tournament exists if and only if an $(n, g, 1)$-design exists.* □

Consequently, we can use the extensive tables (see for instance §II.1 in [3]) on block designs to settle the existence of $(n, g, g-1)$-tournaments. For $n \leq 20$ it yields the following results.

- A $(13, 4, 3)$-tournament and a $(16, 4, 3)$-tournament exist.
- A $(16, 6, 5)$-tournament does not exist.

Note that we have the first negative result. This implies that the necessary conditions in Theorem 1 are in general not sufficient. Another remark is on the $(16, 4, 3)$-tournament. Such a tournament was used in [10] to obtain a regular round-robin tournament for 16 teams with at least 40 breaks. This block design is a so-called *resolvable* block design (see also Definition 2): there exist 5 groups of 4 rounds each, where these 4 rounds contain exactly all 16 teams. Splitting this resolution into 2 parts again, we obtain a $(16, 8, 3)$-tournament.

Concluding we settled the existence of 4 additional tournaments, leaving only 9 cases from Table 2 unaccounted for; these cases we will study in the next section.

4 Ad hoc constructions of single round-robin tournaments

In this section we study the 9 cases in Table 2 that remained unsolved. We devote a separate subsection to most of these cases.

4.1 The $(9, 6, 4)$-tournament

This tournament does not exist, which we can prove as follows. The 36 matches have to be divided over 3 rounds, and each teams appears in 2 rounds only. If we look at round 3, there are three teams not present, say the teams 1, 2, and 3. These all have to appear in the rounds 1 and 2, while all other teams (4 to 9) appear in exactly one of the rounds 1 and 2, say the teams 4, 5, 6 in round 1, and the teams 7, 8, 9 in round 2. Consequently the teams 1, 2, and 3 all have to play the teams 4, 5, and 6 in round one, and the teams 7, 8, and 9 in round 2. The last matches for the teams 1, 2, 3 are the 3 mutual matches, to be played in 2 rounds. This is impossible, because the $(3, 3, 1)$-tournament does not exist. Hence the claim follows.

4.2 The $(10, 6, 3)$-tournament

The tournament has 5 rounds. Group the 10 teams in pairs, and require that in round r the pairs $r-1, r$ and $r+1$ (cyclically) are present. With this basis a tournament can be constructed. The tournament in Table 4.2 has the rounds permuted, to improve the spreading.

Round 1	**Round 2**	**Round 3**	**Round 4**	**Round 5**
1-5	1-7	3-6	1-2	5-9
2-6	2-8	4-8	4-10	6-10
3-4	9-10	5-7	3-9	7-8
1-6	1-8	3-8	1-4	5-10
3-5	7-9	4-7	2-9	8-9
2-4	2-10	5-6	3-10	6-7
1-3	1-9	3-7	4-9	5-8
2-5	2-7	6-8	1-10	7-10
4-6	8-10	4-5	2-3	6-9

Table 4.2. A single $(10, 6, 3)$-tournament

4.3 The $(15, 10, 7)$-tournament

This tournament has 3 rounds, and each team plays in 2 of these rounds. We put the teams 1-5 in group 1, 6-10 in group 2 and 11-15 in group 3. We can require that in round 1 the groups 1 and 2 play, in round 2 the groups 1 and 3, and in round 3 the groups 2 and 3. In a round all matches between teams of different groups have to be played. Note that the teams within a group play according to a single $(5, 5, 2)$-tournament, which exists. Hence we can construct a schedule, which is given in Table 4.3.

Round 1	**Round 2**	**Round 3**
group 1 - group 2	group 1 - group 3	group 2 - group 3
1-2, 3-4, 5-1, 2-3, 4-5	1-3, 5-2, 4-1, 3-5, 2-4	6-8, 10-7, 9-6, 8-10,7-9
6-7, 8-9, 10-6,	11-12, 13-14, 15-11,	11-13, 15-12, 14-11,
7-8, 9-10	12-13, 14-15	13-15, 12-14

Table 4.3. A single (15,10,7)-tournament

4.4 The $(16, 8, 5)$-tournament

This tournament has 6 rounds, and each team appears in 3 rounds. This suggests to divide the teams in 4 groups of size 4 (1-4, 5-8, 9-12, and 13-16), and let 2 groups to play in a round. Since there are 6 pairs of groups, this exactly fits. In a round the teams of one group have to play against all other teams in the other one, and one team of the same group. This leads to the tournament in Table 4.4.

Round 1	**Round 2**	**Round 3**
group 1 - group 2 1-2, 3-4, 5-6, 7-8	group 3 - group 4 9-10, 11-12, 13-14, 15-16	group 1 - group 3 1-3, 2-4, 9-11, 10-12
Round 4	**Round 5**	**Round 6**
group 2 - group 4 5-7, 6-8, 13-15, 14-16	group 1 - group 4 1-4, 2-3, 13-16, 14-15	group 2 - group 3 5-8, 6-7, 9-12, 10-11

Table 4.4. A single (16,8,5)-tournament

4.5 The $(16, 10, 1)$-tournament and the $(16, 12, 1)$-tournament

These were constructed ad hoc from a $(16, 16, 1)$-tournament. Hence they both exist, but the (extensive) schedules are not presented here.

4.6 The $(16, 10, 3)$-tournament

There are 8 rounds, and each team plays in 5 rounds. We require in round r the teams $10r - 9$ to $10r$, where the team number is taken modulo 16 from $\{1, 2, \ldots, 16\}$. So in the first round appear the teams 1 to 10, in round 2 the teams 11 to 16 and 1 to 4, and round 3 the teams 5 to 14, etcetera. With some care the schedule in Table 4.6 can be constructed.

R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
1-7	11-1	5-11	15-5	9-15	3-10	13-4	7-14
2-8	12-2	6-12	16-6	10-16	4-9	14-3	8-13
1-9	11-3	5-13	15-7	9-2	3-12	13-6	7-16
2-10	12-4	6-14	16-8	10-1	4-11	14-5	8-15
3-9	13-3	7-13	1-8	11-2	5-12	15-6	9-16
4-10	14-4	8-14	2-7	12-1	6-11	16-5	10-15
1-5	11-15	5-9	15-3	9-13	3-8	13-2	7-12
2-6	12-16	6-10	16-4	10-14	4-7	14-1	8-11
3-7	13-1	7-11	1-6	11-16	5-10	15-4	9-14
4-8	14-2	8-12	2-5	12-15	6-9	16-3	10-13
3-5	13-15	7-9	1-3	11-13	5-8	15-2	9-12
4-6	14-16	8-10	2-4	12-14	6-7	16-1	10-11
5-7	15-1	9-11	3-6	13-16	7-10	1-4	11-14
6-8	16-2	10-12	4-5	14-15	8-9	2-3	12-13
9-10	3-4	13-14	7-8	1-2	11-12	5-6	15-16

Table 4.6. A single $(16, 10, 3)$-tournament

4.7 The $(16, 12, 5)$-tournament

This tournament contains 4 rounds, and each team appears in 3 rounds. This suggests to divide the teams in 4 groups of 4, and per round to leave out one of these groups. Since each pair of groups appears twice, we let each team play against 2 teams from a different group, and one team from the own group. This leads to the schedule in Table 4.7. Here (a, b)-(c, d) denotes the matches a-c, b-d, a-d, and b-c.

Round 1	Round 2	Round 3	Round 4
(1,2)-(5,6)	(1,2)-(7,8)	(1,2)-(11,12)	(5,6)-(11,12)
(3,4)-(7,8)	(3,4)-(5,6)	(3,4)-(9,10)	(7,8)-(9,10)
(5,6)-(9,10)	(5,6)-(13,14)	(9,10)-(13,14)	(9,10)-(15,16)
(7,8)-(11,12)	(7,8)-(15,16)	(11,12)-(15,16)	(11,12)-(13,14)
(9,10)-(1,2)	(13,14)-(1,2)	(13,14)-(3,4)	(13,14)-(7,8)
(11,12)-(3,4)	(15,16)-(3,4)	(15,16)-(1,2)	(15,16)-(5,6)
1-2, 3-4	1-3, 2-4	1-4, 2-3	5-8, 6-7
5-6, 7-8	5-7, 6-8	9-11, 10-12	9-12, 10-11
9-10, 11-12	13-14, 15-16	13-15, 14-16	13-16, 14-15

Table 4.7. A single $(16, 12, 5)$-tournament

4.8 The $(19, 6, 3)$-tournament

The $(19, 6, 3)$-tournament does not exist. This was established by formulating the construction problem as an IP problem, and let CPLEX run to establish infeasibility.

5 Existence of double round-robin tournaments

The construction of double round-robin tournaments follows along the lines of the single round-robin tournaments, but in addition there are two general constructions from the single round-robin case. These we formulate in the following theorem.

Theorem 7 *Suppose a single (n, g, m)-tournament exists. Then*

(a) A double (n, g, m)-tournament exists.
(b) A double $(n, g, 2m)$-tournament exists.

Proof

(a) To construct the double round-robin tournament, we can repeat the single tournament.
(b) To construct the double round-robin tournament, we can repeat the matches in one round. □

Case (b) is a construction that is usually not preferred as the two matches between two teams are played in the same round. More generally, we define a *proper* round-robin tournament as a tournament with only different matches in a round. We will study exclusively proper round-robin tournaments from now on. There exist parameters (n, g, m) for which the improper tournament exists, although a proper (n, g, m)-tournament does not exist. This is for example the case for the double $(15, 5, 4)$-tournament: since the single $(15, 5, 2)$-tournament exists, we can construct a double $(15, 5, 4)$-tournament. However, a proper double $(15, 5, 4)$-tournament corresponds to a $(15, 5, 2)$-design, which does not exist; see also Subsection 5.3.

The construction in case (a) is often preferred, or even required. However sometimes it is better to use two different single round-robin tournaments, instead of one, see Section 6.

In Table 5 we list all combinations (n, g, m) that satisfy the necessary conditions in Theorem 1 for the double round-robin case. Because of Theorem 7(a) we omit the triples for which the single tournament exists; hence we omit all triples from Table 2,

n	(g,m)								
4	(3,2)	(4,2)							
5	(4,2)								
6	(3,2)	(4,1)	(5,2)	(6,2)					
7	(4,1)	(4,3)	(6,2)	(7,4)					
8	(4,2)	(7,2)	(8,2)						
9	(6,4)	(8,2)							
10	(3,2)	(4,1)	(4,3)	(5,2)	(6,2)	(9,2)	(10,2)	(10,6)	
11	(4,1)	(5,4)	(10,2)	(11,4)					
12	(3,2)	(4,2)	(6,2)	(8,1)	(11,2)	(12,2)			
13	(4,2)	(6,4)	(8,1)	(8,3)	(12,2)	(13,8)			
14	(4,1)	(7,2)	(13,2)	(14,2)					
15	(4,1)	(5,4)	(6,2)	(7,4)	(10,2)	(12,1)	(12,7)	(14,2)	(15,4)
16	(3,2)	(4,2)	(5,2)	(6,2)	(6,5)	(8,2)	(8,6)	(10,2)	(10,6)
	(12,2)	(12,10)	(15,2)	(16,2)	(16,6)	(16,10)			
17	(8,4)	(16,2)							
18	(3,2)	(4,1)	(6,2)	(9,2)	(12,1)	(17,2)	(18,2)		
19	(4,1)	(4,3)	(6,2)	(6,3)	(9,4)	(12,1)	(12,3)	(18,2)	(19,4)
	(19,12)								
20	(4,2)	(5,2)	(8,1)	(10,2)	(19,2)	(20,2)			

Table 5. Parameters for which only a double (n, g, m)-tournament can exist.

except the cases $(9, 6, 4)$, $(16, 6, 5)$, and $(19, 6, 3)$. This leads to 93 cases to investigate for $n \leq 20$.

We will discuss the basic construction methods in this case, and establish the existence of double (n, g, m)-tournaments for $n \leq 20$.

5.1 Round-robin constructions

We can use Theorem 3 to construct tournaments from the regular double round-robin tournament. This leads to 9 $(n, g, 1)$-tournaments: only the cases $(12, 8, 1)$, $(13, 8, 1)$, $(15, 12, 1)$, $(18, 12, 1)$, and $(19, 12, 1)$ remain open. Using Theorem 2 leads to the existence of 12 (n, n, m)-tournaments with n even and $m > 1$.

5.2 Cycle constructions

For cycle constructions for the double round-robin tournament we need the directed version of Theorem 4, which can be found in [2].

Theorem 8 *Suppose D_n is the complete digraph on n vertices and $M = n(n-1)$ arcs, and $3 \leq g \leq n$, such that g divides M. Then D_n can be decomposed into edge-disjoint directed cycles of length g.* □

As in the single tournament case, this theorem leads to the conclusion that all double $(n, g, 2)$-tournaments, satisfying the necessary conditions in Theorem 1 exist. Hence in Table 5 we get the existence of the 43 remaining cases with $m = 2$. Using Theorem 2 we get the existence of the 6 (n, n, m)-tournaments, with n odd and $m > 2$ even.

By using Theorem 8 we can prove the 'even' version of Theorem 5:

Theorem 9 *Suppose n is even, g is even and $\frac{1}{2}g$ divides n. Then a double $(n, g, 1)$-tournament exists.* □

From this theorem it follows that the $(12, 8, 1)$-tournament and the $(18, 12, 1)$-tournament exist.

5.3 Block design constructions

A theorem similar to Theorem 6 also exists for double round-robin tournaments.

Theorem 10 *A proper double $(n, g, g-1)$-tournament exists if and only if a $(n, g, 2)$-design exists.* □

Hence we can settle these cases by using the tables for block designs.

- The $(7, 4, 3)$-tournament, the $(10, 4, 3)$-tournament, the $(11, 5, 4)$-tournament, the $(16, 6, 5)$-tournament, and the $(19, 4, 3)$-tournament exist.
- A (proper!) $(15, 5, 4)$-tournament does not exist. As noted before there exists an improper $(15, 5, 4)$-tournament.

5.4 The remaining cases

We have 15 cases left, 8 cases with m even and 7 cases with m odd. The cases with m even are: $(9, 6, 4)$, $(13, 6, 4)$, $(15, 7, 4)$, $(16, 8, 6)$, $(16, 10, 6)$, $(16, 12, 10)$, $(17, 8, 4)$, and $(19, 9, 4)$. Note that in all these cases an improper tournament exists. The proper cases we solved with CPLEX: they all exist. The 7 remaining cases with m odd remain are: $(13, 8, 1)$, $(13, 8, 3)$, $(15, 12, 1)$, $(15, 12, 7)$, $(19, 6, 3)$, $(19, 12, 1)$, and $(19, 12, 3)$. These were all constructed manually, but are not presented here.

There is an interesting series among these remaining cases, already starting with the single $(7, 3, 2)$-tournament and the double $(11, 5, 4)$-tournament, and continuing with two cases in the list above: the $(15, 7, 4)$-tournament and the $(19, 9, 4)$-tournament. These tournaments are related to $(4n-1, 2n-1, n-1)$-designs, where in our case $n = 2, 3, 4, 5$. The blocks of the block design are used to determine the teams in the round[1]. For the $(7, 3, 2)$-tournament and the double $(11, 5, 4)$-tournament this determines the tournament as well (all teams in a round play against each other). For $n = 4, 5$ each pair of teams appears in $n-1$ rounds. To compose the tournament we have to choose which teams play against each other in which round. This can be done for $n = 4, 5$.

The $(4n-1, 2n-1, n-1)$-design is a symmetric block-design, also called a *Hadamard design* of order n. For small n the Hadamard designs are constructed by using different techniques, but for bigger values of n the existence is unknown. For one of the latest results on Hadamard designs we refer to [4].

6 Spreading of rounds for teams

Till now we discussed parameters (n, g, m) for which (n, g, m)-tournaments might exist. If such tournament exists, there are usually (but not always) many solutions. In this

[1] Note that this is an extra restriction on the construction of the tournament.

section we discuss an additional property that is important in practice, namely the spreading of rounds for a team. This is especially important in cases that a team appears in a few rounds only, few compared to the total number of rounds. In such cases we would prefer that the rounds in which a team appears are 'spread' over all rounds. Since we require this for all individual teams, it is not obvious that we can achieve that. Before continuing it is wise to give a definition of what we mean by 'well-spread'.

Definition 1 *We call an* (n, g, m)*-tournament* well-spread *if after any number of rounds, the number of rounds played by the teams differ at most 1.*

In the regular $(n, n, 1)$-tournaments (n even) the number of rounds played by the teams is the same after any number of rounds. For all other (n, g, m)-tournaments this is not the case. For this reason we allowed a difference of 1 in Definition 1. The regular round-robin tournaments are well-spread. The tournaments constructed by using Theorem 3(a) and (c) are well-spread as well. This is not the case for the tournaments constructed by Theorem 3(b) and (d).

There is another class of tournaments for which the rounds can be ordered such that the tournament is well-spread: the resolvable tournaments. This term is borrowed from the theory of block designs.

Definition 2 *We call a* (n, g, m)*-tournament* resolvable *if it is possible to construct an* (n, n, m)*-tournament from it, by only merging rounds of the* (n, g, m)*-tournament to a new round in the* (n, n, m)*-tournament.*

For a resolvable (n, g, m)-tournament g is a divisor of n. The regular $(n, n, 1)$-tournaments (n even) are (trivially) resolvable, as well as the tournament constructed from it by Theorem 3(a). Also in other cases resolvable tournaments can exist. We mentioned already the $(16, 4, 3)$-tournament in Subsection 3.3. The cases with $m = 2$ can be constructed from resolvable g-cycle systems (see Section V1.12 in [3]). Such 3-cycle system exists in K_9, leading to a resolvable single $(9, 3, 2)$-tournament. (By accident this tournament also corresponds to a resolvable $(9, 3, 1)$-design.) For K_{15} there exist 3-cycle, 5-cycle, and 7-cycle systems, leading to well-spread $(15, 3, 2)$-, $(15, 5, 2)$-, and $(15, 7, 2)$-tournaments.

There are parameters (n, g, m) for which no well-spread tournament exists. A beautiful example of this situation is the single $(7, 3, 2)$-tournament. It corresponds to a $(7, 3, 1)$-design, which is unique (up to permutations of the teams). This tournament is given in Table 6 if one only selects the first 3 matches in each round.

Note that any 2 rounds have exactly 1 team in common; this reflects the fact that this design corresponds to a projective plane (Fano plane). Hence after 2 rounds, there is one team that played 2 rounds (team 2), there are 4 teams that played 1 round (the teams 1,3,4,5) and 2 teams that didn't play at all (the teams 6 and 7). So the tournament is not well-spread. However for the double $(7, 3, 2)$-tournament we can find a well-spread tournament, by weaving two permuted single $(7, 3, 2)$-tournaments. Such tournament is given in Table 6 as a double $(7, 6, 2)$-tournament: by separating the first 3 and second 3 matches in a round to consecutive rounds, we obtain the well-spread double $(7, 3, 2)$-tournament.

Another example is the double $(6, 3, 2)$-tournament, which corresponds to a $(6, 3, 2)$-design. This design is unique and has the property that any two subsets have one team in common. Hence the $(6, 3, 2)$-tournament is not well-spread.

R 1	R 2	R 3	R 4	R 5	R 6	R 7
1-2	2-3	3-4	4-5	5-6	6-7	7-1
1-4	2-5	3-6	4-7	5-1	6-2	7-3
2-4	3-5	4-6	5-7	6-1	7-2	1-3
3-5	4-6	5-7	6-1	7-2	1-3	2-4
3-6	4-7	5-1	6-2	7-3	1-4	2-5
5-6	6-7	7-1	1-2	2-3	3-4	4-5

Table 6. A double $(7, 6, 2)$-tournament

7 Conclusions

We studied an interesting extension of the regular single and double round-robin tournaments for n teams, which we called the (n, g, m)-tournaments. We derived necessary conditions on the parameters (n, g, m) for the existence of a (n, g, m)-tournament. For $n \leq 20$ there are 173 combinations of (n, g, m) satisfying the necessary conditions; only in 4 cases the (proper) (n, g, m)-tournament does not exist.

The (n, g, m)-tournaments appear in practice, especially for the case that 2 matches are played in a round for a subset of teams ($m = 2$). From the theory of graph decomposition we established that such tournaments exist, if the number of teams in a round is a divisor of the total number of matches in the tournament. The case that more matches are played in a round can be interesting as well. One could think of a competitions where many short matches are played in a round, for example in a bridge competition.

Our focus was on the existence of the tournaments. We only touched on one additional property, namely the spreading of rounds for a team. Another requirement could be the spreading of the matches of a team within a round, (see [9]).

Acknowledgment. We thank Nans Wijnstok (competition organizer of the Dutch Inline Skater Hockey League) for motivating this research, and his continuing interest.

References

1. B. Alspach and H. Gavlas, 'Cycle decompositions of K_n and $K_n - I$', *Journal of Combinatorial Theory, Series B* 81 (2001), pp. 77–99.
2. B. Alspach, H. Gavlas, M. Šajna and H. Verrall, 'Cycle decompositions IV: complete directed graphs and fixed length directed cycles', *Journal of Combinatorial Theory, Series A* 103 (2003), pp. 165–208.
3. C. Colbourn and J. Dinitz, *The CRC handbook of Combinatorial Designs*, CRC Press, Inc, Florida, 2007.
4. D. Doković, 'Hadamard matrices of order 764 exist', *Combinatorica* 28 (2008), pp. 487–489.
5. D. de Werra, 'Scheduling in sports', *Annals of Discrete Mathematics* 11 (1981), pp. 381–395.
6. D. de Werra, 'Some models of graphs for scheduling sports competitions', *Discrete Applied Mathematics* 21 (1988), pp. 47–65.
7. D. Fronček and M. Meszka, 'Round robin tournaments with one bye and no breaks in home-away patterns are unique', in: *Proceedings of the 1st International Conference on Multidisciplinary Scheduling: Theory and Applications*, Nottingham, UK (2003), pp. 331-340.
8. T. P. Kirkman, 'On a problem in combinations', *Cambridge and Dublin Mathematics Journal* 2 (1847), pp. 191–204.

9. S. Knust, 'Scheduling sports tournaments on a single court minimizing waiting times', *Operations Research Letters* 36 (2008), pp. 471–476.
10. G. Post and G. Woeginger, 'Sports tournaments, home-away assignments, and the break minimization problem', *Discrete Optimization* 3 (2006), 165-173.
11. M. Šajna, 'Cycle decompositions III: Complete graphs and fixed length cycles', *Journal of Combinatorial Designs* 10 (2002), pp. 27–77.
12. J. A. M. Schreuder, 'Combinatorial aspects of construction of competition Dutch professional football leagues', Discrete Applied Mathematics **35**, (1992), pp. 301–312.
13. A. van Weert and J.A.M. Schreuder, 'Construction of basic match schedules for sports competitions by using graph theory', in: PATAT 1997, Lecture Notes in Computer Science 1408, E. Burke and M. Carter (eds.), Springer (1998), pp. 201–210.

Adaptive Selection of Heuristics for Improving Constructed Exam Timetables

Edmund K. Burke, Rong Qu, and Amr Soghier

School of Computer Science, University of Nottingham
Nottingham, NG8 1BB, UK
{ekb,rxq,azs}@cs.nott.ac.uk

Abstract. This paper presents a hyper-heuristic approach which hybridises low-level heuristics to improve constructed timetables. The constructed timetable is analysed and the exams causing a soft-constraint violation are identified. It is observed that both the type of move performed and the order in which exams are rescheduled in the timetable affect the quality of the solution produced. After testing different combinations in a hybrid approach, the Kempe chain move heuristic and swapping timeslots proved to be the best heuristics to use in a hybridisation. Similarly, it was proved that ordering the exams using Saturation Degree and breaking any ties using Largest Weighted Degree produces the best results. Based on these observations, an iterative hybrid approach is developed to adaptively hybridise these two heuristics in two stages. In the first stage, random heuristic sequences are generated and applied to the problem. The heuristic sequences are automatically analysed. The heuristics repeated in the best sequences are fixed while the rest are randomly changed in an attempt to find the best heuristic sequence. The approach is tested on the Toronto benchmark and the exam timetabling track of the second International Timetabling Competition, to evaluate its ability to generalise. The hyper-heuristic with low-level improvement heuristics approach was found to generalise well over the two different datasets and performed comparably to the state of the art approaches.

1 Introduction

For more than 40 years exam timetabling has been one of the mostly studied domains in the AI and OR research communities. This is due to its importance in many academic institutions worldwide. However, much of the research was aimed at developing methodologies that produce the best quality timetables for a single problem [24]. A more recent direction in this field, namely, hyper-heuristics, aims to raise the level of generality of search methodologies to create algorithms that act well over a range of problems. A hyper-heuristic is seen as a heuristic to choose heuristics [7]. In this case the low-level heuristics represent the search space. The low-level heuristics can be categorised as heuristics which construct a timetable or heuristics which perform certain moves to improve a constructed timetable. This paper presents a random iterative hyper-heuristic approach which uses improvement low-level heuristics. This approach has been tested on the Toronto benchmark and the second International Timetabling Competition (ITC2007) exam timetabling problems. It proved to generalise well over

the two datasets. Furthermore, very competitive results have been produced against other approaches in the literature.

The following section presents a brief description of the benchmarks and an overview on different approaches, including hyper-heuristic approaches, developed in the exam timetabling domain. A random iterative hyper-heuristic to improve timetables is proposed in section 3. An adaptive methodology to select low-level heuristics and the results obtained are presented in section 4. The future extensions of this work are summarised in section 5.

2 Exam Timetabling

2.1 The Toronto Benchmark

An exam timetabling problem consists of a set of exams to be allocated to a given set of timeslots. The generated timetable must satisfy the hard constraints of the problem, which are the requirements that cannot be violated, e.g. no one student must be scheduled to sit two exams during the same period. A timetable which meets all the hard constraints given is called a feasible timetable. A timetabling problem can also have a set of soft constraints that can be violated. The violations of these constraints are usually used to determine the quality of the timetable generated, e.g. there must be a certain number of periods between two exams sat by the same student. Therefore, high quality timetables contain the least number of soft constraint violations. The Toronto benchmark problem is well known in the exam timetabling community since it was firstly introduced by Carter et al. [11] in 1996. Over the years, a slightly different version has been used to test some approaches in the literature. The characteristics of the two versions of this dataset are presented in table 1. The problem has one hard constraint where conflicting exams cannot be assigned to the same time slot. In addition, a soft constraint is present where conflicting exams should be spread throughout the timetable as far as possible from each other. The goal here is to minimise the sum of proximity costs given as follows:

$$\sum_{i=0}^{4}(w_i \text{ x } n)/S$$

where

- $w_i = 2^{4-i}$ is the cost of assigning two exams with *i* time slots apart. Only exams with common students and are four or less time slots apart are considered as violations, i.e. $i \in \{0,1,2,3,4\}$
- n is the number of students involved in the conflict
- S is the total number of students in the problem

Since then this problem has been used to test and compare many approaches in the literature. Recently, a more constrained set of benchmarks was made available as part of the International Timetabling Competition (ITC2007) [17]. The next section describes the ITC2007 dataset in detail.

Problem	Exams I/II	Students I/II	Enrolments I/II	Density	Time Slots
car91	682	16925	56877	0.13	35
car92	543	18419	55522	0.14	32
ear83 I	190	1125	8109	0.27	24
ear83 II	189	1108	8057	0.27	24
hec92 I	81	2823	10632	0.42	18
hec92 II	80	2823	10625	0.42	18
kfu93 I	461	5349	25113	0.06	20
lse91	381	2726	10918	0.06	18
sta83 I	139	611	5751	0.14	13
sta83 II	138	549	5417	0.19	35
tre92	261	4360	14901	0.18	23
uta92 I	622	21266	58979	0.13	35
uta92 II	638	21329	59144	0.12	35
ute92	184	2750	11793	0.08	10
yor83 I	181	941	6034	0.29	21
yor83 II	180	919	6002	0.3	21

Table 1. Characteristics of the two versions of the Toronto Benchmark datasets

2.2 The International Timetabling Competition (ITC2007) dataset

The ITC2007 exam timetabling track could be considered as a complex and a more practical dataset in comparison to the Toronto benchmark. This is due to the larger number of constraints it contains. A full description of the problem and the evaluation function can be found in [17]. In addition, the characteristics which define the instances are summarised in table 2. The problem consists of the following:

- A set of timeslots covering a specified length of time. The number of timeslots and their durations are provided.
- A set of exams which should be allocated to the timeslots.
- A list of the students enrolled in each exam.
- A set of rooms with different capacities.
- A set of additional hard constraints (e.g. exam X must be after exam Y or exam A must use Room R).
- A set of soft constraints and their associated penalties.

In comparison to the Toronto benchmark, the ITC2007 dataset has more than one hard constraint. The hard constraints are as follows:

- No student sits more than one exam at the same time.
- The capacity for each individual room should not be exceeded at a given period.
- Period lengths should not be violated.
- Additional hard constraints should be all satisfied.

The soft constraints violations are summarised as follows:

- **Two Exams in a Row** The number of occurrences where a student sits two exams in a row on the same day.

– **Two Exams in a Day** The number of occurrences where a student sits two exams on the same day. If the exams are back to back then this is considered as a Two Exams in a Row violation to avoid duplication.
– **Period Spread** The exams have to be spread a certain number of timeslots apart.
– **Mixed Durations** The number of occurrences where exams of different durations are assigned to the same room.
– **Larger Exams Constraint** The number of occurrences where the largest exams are scheduled near the end of the examination session. The number of the largest exams and the distance from the end of the exam session are specified in the problem description.
– **Room Penalty** The number of times where certain rooms, which have an associated penalty, are used.
– **Period Penalty** The number of times where certain timeslots, which have an associated penalty, are used.

Instance	Conflict Density	Exams	Students	Periods	Rooms	no. of Hard Constraints
exam 1	5.05	607	7891	54	7	12
exam 2	1.17	870	12743	40	49	14
exam 3	2.62	934	16439	36	48	185
exam 4	15.0	273	5045	21	1	40
exam 5	0.87	1018	9253	42	3	27
exam 6	6.16	242	7909	16	8	23
exam 7	1.93	1096	14676	80	15	28
exam 8	4.55	598	7718	80	8	21

Table 2. Characteristics of the ITC2007 dataset

2.3 Exam timetabling approaches for the ITC2007 dataset

A three phased approach was developed by Muller [18] to solve the problems in the ITC2007 exam timetabling track. The first phase consists of an Iterative Forward Search algorithm to find a feasible solution. Hill climbing is then used to find the local optima in the second phase. Finally, a Great Deluge Algorithm is applied to further explore the search space.

Gogos et al. [14] proposed a method which used a GRASP (Greedy Randomised Adaptive Search Procedure). In the construction phase, five orderings of exams based on various criteria are generated. Tournament selection is used to select exams until they are all scheduled. A backtracking strategy using a tabu list is employed as required. In the improvement phase, Simulated Annealing is used. Finally, room allocations are arranged using integer programming in the third phase.

Atsuta et al. [3] used a constraint satisfaction solver incorporating tabu search and iterated local search. The solver differentiates between the constraints and their corre-

sponding weights during computation to improve performance. De Smet [12] also incorporated local search techniques in a solver called Drools, an Open-Source Business Rule Management System (http://www.jboss.org/drools/).

Pillay [19] introduced a biological inspired approach which mimics cell behaviour. The exams are initially ordered using the saturation degree heuristic and scheduled sequentially in the available "cells" i.e. timeslots. If more than one timeslot is available, the slot which causes the least overall constraint violations is chosen. Rooms are chosen using the best fit heuristic. If a conflict occurs before all the exams are scheduled, the timetable is rearranged to reduce the soft constraints violation. This is described as cell division. If the overall soft constraint violation is not improved without breaking hard constraints, cell interaction occurs. The timeslots are swapped in this process to remove hard constraint violations. The process continues until a feasible solution is achieved. Finally, the contents of cells having equal durations are swapped to improve the solution. This is called cell migration.

McCollum et al. [16] proposed a two phased approach where an adaptive heuristic is used to achieve feasibility during the first phase. The second phase improves the solution through the employment of a variant of the Great Deluge Algorithm.

2.4 Exam timetabling approaches for the Toronto Benchmark

An approach which uses a sequential construction method, employed by Caramia et al. [10], to assign exams in the least number of timeslots was able to produce the best quality timetables for four of the Toronto benchmark instances. It uses a greedy scheduler to obtain a feasible solution. A penalty decreaser and trader are then applied to improve the quality of the constructed solution. Burke et al. [6] introduced an approach which combines a variable neighbourhood search with a genetic algorithm which produced the best quality solution for one of the Toronto instances. In addition Burke et. al [5] proposed a method where a hill-climber compares the candidate solution with a solution produced a couple of iterations back instead of the current solution. This was called the "late acceptance criteria" and it produced the best quality solutions for another four instances. Yang et. al [26] employed Case-Based Reasoning to choose graph-heuristics to construct initial solution which were improved using a Great Deluge algorithm. This approach produced the best quality solution for one of the instances. The results obtained by these approaches are presented in section 4.1.

2.5 Hyper-heuristics in exam timetabling

Recently, some new methods were investigated to automatically find the best heuristic to solve a set of instances. This has led to the introduction of Hyper-heuristics. A hyper-heuristic can be seen as a method to choose low-level heuristics depending on the problems in hand. Furthermore, it could be used to adapt or tune heuristics and meta-heuristics. Hyper-heuristics in exam timetabling can be categorised, according to the low-level heuristics they use, into two types as follows:

1. Hyper-heuristics with constructive low-level heuristics
2. Hyper-heuristics with improvement low-level heuristics

A Tabu search was developed by Burke et al. [8] to optimise a search space of heuristic sequences comprised of two or more low-level heuristics. This work was extended in later research by Qu et al. [23] to construct heuristic sequences which produce feasible timetables. The combinations are then analyzed to find distribution patterns of low-level heuristics, based on which the heuristic sequences are adaptively adjusted to construct better timetables. In addition, hybridisations of the graph based hyper-heuristic with local search methods was investigated in [22].

Asmuni et al. [1] used fuzzy logic to combine two out of three graph colouring heuristics. The idea was to combine the two heuristics into a single value which calculates the difficulty of allocating an exam to a timeslot. The exams are ordered using this value and are scheduled in order. Furthermore, the approach was extended to tune the fuzzy rules instead of keeping them fixed [2].

Ersoy et al. [13] developed an approach called the hyperhill-climber where a hyper-heuristic is embedded in a memetic algorithm. The aim of this hyper-heuristic was to select the best hill-climber to apply or decide the best order in which hill-climbers are executed. In addition, Pillay et al. [20] created another approach where genetic programming was used to evolve hyper-heuristics.

Biligan et al. [4] presented different heuristic selection methods and acceptance criteria for hyper-heuristics in exam timetabling. Finally, a different method of combining heuristics was presented by Pillay et al. [21]. The low-level heuristics are combined hierarchically and applied simultaneously instead of being applied sequentially.

3 A Hyper-heuristic with low-level improvement heuristics

Several low-level heuristics can be used to improve a timetable with varying quality. The different low-level heuristics used could be considered as different methods for escaping from local optima. However, the order in which exams are moved and the type of moves performed play an important role in finding the best quality solution. In our hyper-heuristic approach, an initial feasible solution is constructed using the Largest Degree heuristic where the exams in the ordering are assigned randomly to a timeslot causing the least penalty. Our objective is to analyse the performance of the different low-level heuristics used to minimise the penalty incurred from a constructed solution. In addition, we test the effect of using different orderings for the exams causing penalties in the solution. Finally we develop an adaptive approach which orders the exams causing violations and automatically selects the best heuristic to use for each exam to produce an improvement.

3.1 The low-level heuristics

In this paper we investigate the effect of using different low-level heuristics or neighbourhoods to improve timetables. A combination of two improvement low-level heuristics is used in our approach. The following is a list of the heuristics investigated:

1. Move Exam (ME): This heuristic selects an exam and reassigns it to the timeslot causing the least penalty.

2. Swap Exam (SE): This heuristic selects an exam and tries swapping it with a scheduled exam leading to the least penalty timetable.
3. Kempe Chain Move (KCM): This is similar to the SE heuristic but is more complex as it involves swapping a subset of conflicting exams in two distinct timeslots. This neighbourhood operator proved success when it was previously used in [6] and [25].
4. Swap Timeslot (ST) : This heuristic selects an exam and swaps all the exams in the same timeslot with another set of exams in a different timeslot. After testing all the timeslots, the swap producing the least penalty timetable is applied.

3.2 The random iterative hyper-heuristic

The study presented in this paper takes a similar approach to that presented in [23] where a random iterative hyper-heuristic generates heuristic sequences of different quality to solve the benchmark problem mentioned in section 2.1. Instead of using the heuristic sequences to construct solutions, the heuristic sequences are used here to improve constructed feasible solutions by rescheduling exams causing penalties. Figure 1 presents the pseudo-code of this random iterative hyper-heuristic. The process starts by constructing an initial feasible solution. Since the initial solution constructed affects the improvement process, a random largest degree graph colouring heuristic which orders exams according to the number of conflicts each exam has with others is used [6]. This allows us to compare our approach to other approaches in the literature which use a similar method in construction. At every iteration, the exams causing violations in the constructed solution are identified and a random sequence of moves is generated. A move is the application of one of the low-level heuristics described in section 3.1. The sequence of moves is then applied to the sequence of exams as they are unscheduled one by one. Only moves that improve the current solution are accepted. If a move does not improve the solution, it is skipped and the exam stays in its current position. A sequence is discarded if an improvement is not obtained after the whole sequence is employed.

This approach was applied to four instances (hec92 I, sta83 I,yor83 I and tre92) of the Toronto benchmark exam timetabling problems described in section 2.1 for off-line learning of the best heuristic hybridisations and the order of execution leading to the best improvement. After running this process for (ex50) times, where e is the number of exams causing soft constraint violations in the constructed solution, a set of sequences and the penalties of their corresponding solutions are obtained for further investigation on the effectiveness of the different heuristics used. Finally, an adaptive approach was developed and applied to the Toronto benchmark. Furthermore, to test the generality of the approach, it was applied to the ITC2007 exam timetabling track. The approach is presented in section 4.

3.3 Analysis of hybridising improvement low-level heuristics

In order to clearly observe the effect of the different low-level heuristics in improving solutions, the heuristic sequences generated consist of two heuristics. We use the Kempe chain move(KCM) heuristic as the basic heuristic in the sequences as it has proved to

```
construct a feasible solution using LD with random time-slot assignment
create a randomly ordered list of the exams contributing to the overall penalty incurred
for i = 0 to i = e × 50  //e: number of exams causing penalty
        for n = 1 to n = e
        initialise heuristic sequence h = [KCM KCM … KCM KCM] // h has size e
        h = randomly change n heuristics in h to SM, SS or ST
        construct a solution c using h
        if solution c is feasible
                save h and the penalty of its corresponding solution c
```

Fig. 1. The pseudocode of the random iterative hyper-heuristic with low-level improvement heuristics

be successful in previous work [6, 25]. The rest of the heuristics (ME, SE and ST) are randomly hybridised into the list of KCM.

The random sequences are generated with different percentages of hybridisation by inserting n ME, SE or ST, n = [1,..,e] in the sequences. For each hybridisation of KCM with either ME, SE or ST, fifty samples are obtained for each amount of hybridisation.

We applied this approach to four instances of the Toronto benchmark exam timetabling problems [11]. Table 3 presents the results obtained using ME, SE and ST in a hybridisation with KCM as well as a comparison against using KCM only.

	hec92 I	yor83 I	sta83 I	tre92
KCM without hybridisation Best	13.50	43.84	160.43	8.99
KCM with ME Best	12.03	43.84	157.48	8.91
KCM with SE Best	12.03	42.37	157.75	8.75
KCM with ST Best	**11.30**	**41.79**	**157.27**	**8.57**

Table 3. Best results using KCM without a hybridisation and with several different moves.

It was observed that using a Kempe chain only produces the worst results. After introducing other heuristics in a hybridisation with the Kempe chain moves, better results are obtained. Another observation from table 3 is that swapping timeslots and performing Kempe chain moves produces the best improvement for all the instances. One possible reason may be that swapping timeslots allows the search to be more diverse and to sample different areas of the search space to find good solutions faster. In addition, no obvious trends could be obtained on the amount of ST hybridisation within the best heuristic sequences. However, in all the sequences leading to the best timetables, the ST heuristic is randomly distributed within the sequences and the percentage of hybridisation is less than 50%.

3.4 Variations of Orderings of the exams causing a penalty

To analyse the effect of ordering the unscheduled exams causing a soft constraint violation in a solution, we decided to test different orderings while using the Kempe Chain and swapping timeslot hybridisation stated in the previous section. After the exams causing violations are identified, they are ordered first before being reassigned to a timeslot. Several orderings can be used to guide the search as follows:

- Largest Degree (LD) : The exams are ordered decreasingly according to the number of conflicts each exam has with others.
- Largest Weighted Degree (LWD) : The exams are ordered similarly to LD but the exams are weighted according to the number of students involved in the conflict.
- Saturation Degree (SD) : The exams are ordered increasingly according to the number of remaining timeslots available to assign them without causing conflicts. In the case where ties occur, LWD is used as a tie breaker. From our previous work it was shown that SD produces the best results when LWD is used to break ties in the ordering [9].
- Largest Penalty (LP) : The exams are ordered decreasingly according to the penalty they incur in the current solution.
- Random Ordering (RO) : The exams are ordered randomly.

Table 4 presents the average and best results of applying different orderings to the unscheduled exams, then running a random heuristic sequence of KCM and ST to assign them to better timeslots.

	hec92 I	yor83 I	sta83 I	tre92
KCM with ST + RO Average	11.99	42.63	159.74	8.91
KCM with ST + RO Best	11.60	41.33	158.46	8.64
KCM with ST + LD Average	12.15	42.09	159.39	9.00
KCM with ST + LD Best	11.32	39.69	157.76	8.66
KCM with ST + LWD Average	12.06	42.08	159.74	9.02
KCM with ST + LWD Best	11.39	39.69	157.49	8.66
KCM with ST + LP Average	12.69	42.10	163.32	8.91
KCM with ST + LP Best	12.50	39.69	159.50	8.51
KCM with ST + SD tb LWD Average	12.69	41.74	159.21	8.90
KCM with ST + SD tb LWD Best	**11.19**	**39.47**	**157.18**	**8.49**

Table 4. Results of hybridising KCM with ST using different orderings of the exams causing a soft constraint violation. The notation "X tb Y" means heuristic Y is used to break ties in heuristic X

As shown in table 4, we found that using SD and breaking any ties in the ordering using LWD produced the best results. This is because SD orders the unscheduled exams according to the number of remaining timeslots available to assign them without causing conflicts. Therefore, the chances of moving exams at the top of the SD list

and finding better timeslots for them becomes higher. Ordering the exams according to the penalty they incur proved to be the second best ordering after SD. LD and RO performed the worst when applied.

4 Adaptive Selection of Low-level Heuristics for Improving Exam Timetables

Figure 2 presents the initialisation stage of the adaptive approach. The exams causing a penalty are first identified and are unscheduled. They are then put in a list and ordered using SD. Random heuristic sequences are generated using KCM and ST to reschedule the exams. The sequences are then applied to the ordered exams and the corresponding solutions are saved.

```
construct a feasible solution using LD with random time-slot assignment
create an ordered list of the exams which cause a penalty
for i = 0 to i = e × 10 //e: number of exams causing penalty
        for n = 1 to n = e
        initialise heuristic sequence h = {KCM KCM … KCM KCM}
        h = randomly change n heuristics in h to ST
        construct a solution using h
        if solution c is feasible
                save h and the penalty of its corresponding solution c
```

Fig. 2. The pseudocode of the initialisation stage of the adaptive hyper-heuristic with low-level improvement heuristics

The above observations indicate that the best solutions were obtained when ordering the exams causing violations using SD, and rescheduling them using either a Kempe-chain move or swapping timeslots. It was also observed that the heuristic sequences resulting in the best solutions used the same move for the majority of the exams (i.e. the same heuristic appears in the same position in the majority of the sequences). Therefore, we developed an intelligent approach that performs an analysis to the best 5% of the sequences produced to generate a new set of sequences. The new set of sequences obtained better results for all the problem instances. The adaptive approach was tested and showed to be effective and comparable with the best approaches in the literature.

Figure 3 presents the pseudo-code of the approach which hybridises ST with KCM in two stages. The process is presented as follows:

1. In the first stage, the best 5% of heuristic sequences are collected and analysed. If the same heuristic is used in more than 75% of the heuristic sequences, then it is stored. Otherwise the heuristic is neglected and the position is randomly assigned as KCM or ST.

2. *n* x 5 sequences for the large instances (uta92 I, uta92 II, car91 and car92) and *n* x 10 sequences for the small instances are generated, respectively. The new sequences are then applied to the instance.

```
construct initial heuristic sequences  // see Fig.2
collect the best 5 % of the heuristic sequences
for i = 0 to i < number of exams causing penalty
{
        count = 0
        for j = 0 to j < number of sequences
        {
                if heuristicSequence[i][j] = KCM
                count ++
        }
        if count > 0.75 * number of exams causing penalty
                then finalHeuristicSequence[i] = KCM
        else if count < 0.25 * number of sequences
                then finalHeuristicSequence[i] = ST
        else
                finalHeuristicSequence[i] = empty
}
for i = 0 to i < n // n = number of empty positions * 5 for large instances
                   // or  number of  empty  positions * 10 for small instances
{
        for j = 0 to j < number of exams causing penalty
        {
                if finalHeuristicSequence[j] = empty
                        finalHeuristicSequence[j] = KCM or ST
        }
        construct a solution using finalHeuristicSequence[j]
        if a better solution is obtained
                save finalHeuristicSequence[j] and the penalty of its corresponding
                solution
}
```

Fig. 3. Adaptive generation of heuristic sequences hybridising KCM and ST

4.1 The Toronto Benchmark Results

We tested this approach on the Toronto benchmark exam timetabling problems and present the results in tables 5 and 6. The average computational time across the instances is also presented for 30 runs on a Pentium IV machine with a 1 GB memory.

	hec92 I	yor83 I	ear83 I	sta83 I	car92	car91	uta92 I	ute92	lse91	tre92	kfu93
AIH Average	12.69	41.74	38.98	159.21	4.49	5.39	3.56	27.97	11.45	8.90	15.54
AIH Best	11.19	39.47	35.79	157.18	4.31	5.19	3.44	26.70	10.92	8.49	14.51
Time(s)	397	1683	1692	759	41954	97961	61284	641	1466	4293	2745

Table 5. Results from the the Adaptive Improvement Hyper-heuristic (AIH) approach on the Toronto Benchmark dataset.

	hec92 II	yor83 II	ear83 II	sta83 II	uta92 II
AIH Average	12.43	50.49	41.98	35.00	3.54
AIH Best	11.35	49.72	39.60	32.57	3.45
Time (s)	498	1374	2792	1888	81316

Table 6. Contd. Results from the the Adaptive Improvement Hyper-heuristic (AIH) approach on the Toronto Benchmark dataset.

The best results stated in the literature are presented in table 7. These include the hill-climbing with a late acceptance strategy implemented by Burke et al. [5], the variable neighbourhood search incorporating the use of genetic algorithms used by Burke et al. [6], the sequential construction method developed by Caramia et al. [10] and the Case-Based Reasoning approach employed by Yang et al. [26]. These algorithms are described in section 2.4.

Problems	AIH Best	Burke(2008) Best [5]	Burke(2010) Best [6]	Caramia(2008) Best [10]	Yang(2005) Best [26]
hec92 I	11.19	10.06	10.00	**9.20**	10.83
sta83 I	157.18	157.03	**156.90**	158.20	158.35
yor83 I	39.47	**34.78**	34.90	36.20	36.35
ute92	26.70	24.79	24.80	**24.40**	25.39
ear83 I	35.79	32.65	32.80	**29.30**	33.70
tre92	8.49	**7.72**	7.90	9.40	7.92
lse91	10.92	9.86	10.00	**9.60**	10.35
kfu93	14.51	**12.81**	13.00	13.80	13.82
car92	4.31	**3.81**	3.90	6.00	3.93
uta92 I	3.44	3.16	3.20	3.50	**3.14**
car91	5.19	4.58	4.60	6.60	**4.50**

Table 7. Best results obtained by the Adaptive Improvement Hyper-heuristic (AIH) compared to the best approaches in the literature on the Toronto Benchmark

The results obtained indicate the generality of our approach to different constructed timetables. We also make a comparison with other hyper-heuristics which produced the best results in the literature in table 8. In comparison with the graph-based hyper-heuristic in [8], our approach performs better in all the cases reported. In addition, it performs better in 8 out of 11 cases in comparison with the hyper-heuristics investigated in [21] and [22]. Finally, it performs better in 10 out of 11 cases compared to the Tabu search hyper-heuristic investigated in [15]. Only the problems presented in table 8 were compared to other results since the results for the other instances in table 1 were not reported in the literature.

Problems	AIH Best	Kendall(2004) Best [15]	Burke(2007) Best [8]	Pillay(2009) Best [21]	Qu(2009) Best [22]
hec92 I	**11.19**	11.86	12.72	11.85	11.94
sta83 I	**157.18**	157.38	158.19	158.33	159.00
yor83 I	**39.47**	-	40.13	40.74	40.24
ute92	**26.70**	27.60	31.65	28.88	28.30
ear83 I	**35.79**	40.18	38.19	36.86	35.86
tre92	8.49	**8.39**	8.85	8.48	8.60
lse91	**10.92**	-	13.15	11.14	11.15
kfu93	**14.51**	15.84	15.76	14.62	14.79
car92	4.31	4.67	4.84	4.28	**4.16**
uta92 I	3.44	-	3.88	**3.40**	3.42
car91	5.19	5.37	5.41	**4.97**	5.16

Table 8. Best results obtained by the Adaptive Improvement Hyper-heuristic (AIH) compared to other hyper-heuristics approaches in the literature on the Toronto Benchmark

4.2 The International Timetabling Competition (ITC2007) Results

To test the generality of our approach, we applied it to the ITC2007 exam timetabling dataset. The initial solution is constructed by ordering the exams according to their saturation degree. The exams are assigned a random timeslot in the situation where more than one timeslot is available. After a feasible solution is constructed the Adaptive Improvement Hyper-heuristic was applied to the constructed solution. To allow a fair comparison with the reported competition results, the approach was run for the same amount of time using 11 distinct seeds for each instance. Table 9 presents the results we obtained in comparison with the best in the literature. The description of the approaches used for comparison are presented in section 2.3. We do emphasise that the objective here is not to beat the best reported results but to demonstrate the generality of our approach to different problems with different constraints. A dash in the table means that no feasible solution was achieved.

The Extended Great Deluge in [16] obtained the best results for 5 out of the 8 instances. However, the approach was run for a longer time as it was developed after the competition. In the competition, the best results for all the 8 instances were reported

in [18] using a three phased approach. The GRASP used in [14] produced the second best results.

In comparison to the Constraint Based Solver developed in [3], our approach performed better in 3 out of the 8 instances. The approach using the Drools solver in [12] obtained feasibility for only 5 instances. Our approach outperformed it as we were able to gain feasibility for all the 8 instances. This demonstrates the generality of our approach to solving exam timetabling problems. Finally, our approach performed better on 6 of the 8 instances in comparison with the biologically inspired approach proposed in [19].

Instances	AIH Best	McCollum(2009) Best [16]	Muller(2008) Best [18]	Gogos(2008) Best [14]	Atsuta(2008) Best [3]	De Smet(2008) Best [12]	Pillay(2008) Best [19]
Exam 1	6235	4633	**4370**	5905	8006	6670	12035
Exam 2	2974	405	**400**	1008	3470	623	3074
Exam 3	15832	**9064**	10049	13862	18622	-	15917
Exam 4	35106	**15663**	18141	18674	22559	-	23582
Exam 5	4873	3042	**2988**	4139	4714	3847	6860
Exam 6	31756	**25880**	26950	27640	29155	27815	32250
Exam 7	11562	**4037**	4213	6683	10473	5420	17666
Exam 8	20994	**7461**	7861	10521	14317	-	16184

Table 9. Best results obtained by the Adaptive Improvement Hyper-heuristic (AIH) compared to the best approaches in the literature on the ITC2007 dataset

5 Conclusions

The study presented in this paper implements a hyper-heuristic approach which adaptively adjusts heuristic combinations to achieve the best improvement for constructed timetables. An investigation is made on the low-level heuristics used and the order in which exams causing soft constraint violations are rescheduled. The analysis is performed on a set of four benchmark instances of differing difficulty in an off-line learning process. It is shown that, of the heuristics tried, the best to combine with Kempe chains is swapping timeslots. In addition, better solutions are produced when ordering the exams causing a soft constraint violation using Saturation Degree and breaking any ties with Largest Weighted Degree. Based on the output of the learning process, an adaptive approach which analyzes and adjusts some randomly generated sequences is implemented and applied to the rest of the instances. Furthermore, the approach is applied to a different and more constrained dataset, the ITC2007 dataset. The hyper-heuristic produced very competitive results compared to other approaches in the literature on both datasets.

Future research directions include performing improvements during the timetable construction stage instead of performing the improvements at the end of the construction. Using hybridisations of more than two low-level heuristics could also be investi-

gated. Finally, the approach investigated in this paper can be applied to course timetabling problems.

References

1. H. Asmuni, E.K. Burke, J. Garibaldi, and B. McCollum. Fuzzy multiple ordering criteria for examination timetabling. In E.K. Burke and M. Trick, editors, *Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling*, volume 3616 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2004.
2. H. Asmuni, E.K. Burke, J. Garibaldi, B. McCollum, and A.J. Parkes. An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers and Operations Research*, 36(4):981–1001, 2009.
3. M. Atsuta, K. Nonobe, and T. Ibaraki. Itc2007 track 1: An approach using general csp solver. In *Practice and Theory of Automated Timetabling (PATAT 2008)*, pages 19–22, August 2008.
4. B. Biligan, E. Ozcan, and Korkmaz E.E. An experimental study on hyper-heuristics and exam timetabling. In E. Burke and H. Rudova, editors, *Practice and Theory of Automated Timetabling VI: Selected Papers from the 6th International Conference PATAT 2006*, volume 3867 of *Lecture Notes in Computer Science*, pages 394–412, 2007.
5. E.K. Burke and Y. Bykov. A late acceptance strategy in hill-climbing for examination timetabling problems. In *Proceedings of the conference on the Practice and Theory of Automated Timetabling(PATAT)*, 2008.
6. E.K. Burke, A. Eckersley, B. McCollum, S. Petrovic, and R. Qu. Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research (EJOR)*, 206:46–53, 2010.
7. E.K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Meta-Heuristics*, pages 457–474. Kluwer, 2003.
8. E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176:177–192, 2007.
9. E.K. Burke, R. Qu, and A. Soghier. Adaptive tie breaking and hybridisation in a graph-based hyper-heuristic for exam timetabling problems. *under review at Journal of Operational Research*, 2010.
10. M. Caramia, P. Dell Olmo, and G.F. Italiano. Novel local-search-based approaches to university examination timetabling. *Informs Journal of Computing*, 20(1):86–99, 2008.
11. M.W. Carter, G. Laporte, and S.Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of Operational Research Society*, 74:373–383, 1996.
12. G. De Smet. Itc2007 - examination track. In *Practice and Theory of Automated Timetabling (PATAT 2008)*, pages 19–22, August 2008.
13. E. Ersoy, E. Ozcan, and Uyar S. Memetic algorithms and hill-climbers. In P. Baptiste, G. Kendall, A.M. Kordon, and F. Sourd, editors, *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications Conference(MISTA2007)*, pages 159–166, 2007.
14. C. Gogos, P. Alefragis, and E. Housos. A multi-staged algorithmic process for the solution of the examination timetabling problem. In *Practice and Theory of Automated Timetabling (PATAT 2008)*, pages 19–22, 2008.
15. G. Kendall and N. Mohd Hussin. An investigation of a tabu search based hyper-heuristic for examination timetabling. In G. Kendall, E. Burke, S. Petrovic, and M. Gendreau, editors, *Selected Papers from MISTA 2005*, pages 309–328. Springer, 2005.

16. B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke, and S. Abdullah. An extended great deluge approach to the examination timetabling problem. In *Proceedings of the 4th Multidisciplinary International Scheduling: Theory and Applications 2009 (MISTA 2009), pp. 424-434, 10-12 August , Dublin, Ireland*, 2009.
17. B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, L. Di Gaspero, A. J. Parkes, R. Qu, and E. K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal of Computing, doi: 10.1287/ijoc.1090.0320*, 2008.
18. T. Muller. Itc 2007 solver description: A hybrid approach. In *Practice and Theory of Automated Timetabling (PATAT 2008)*, pages 19–22, August 2008.
19. N. Pillay. A developmental approach to the examination timetabling problem. In *Practice and Theory of Automated Timetabling (PATAT 2008)*, pages 19–22, August 2008.
20. N. Pillay and W. Banzhaf. A genetic programming approach to the generation of hyper-heuristic systems for the uncapicitated examination timetabling problem. In Neves et al., editor, *Progress in Artificial Intelligence*, volume 4874 of *Lecture Notes in Artificial Intelligence*, pages 223–234, 2007.
21. N. Pillay and W. Banzhaf. A study of heuristic combinations for hyper-heuristic systems for the uncapicitated examination timetabling problem. *European Journal of Operational Research*, 197:482–491, 2009.
22. R. Qu and E.K. Burke. Hybridisations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of Operational Research Society*, 60:1273–1285, 2009.
23. R. Qu, E.K. Burke, and B. McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404, 2009.
24. R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, and S.Y. Lee. A survey of search methodologies and automated approaches for examination timetabling. *Journal of Scheduling*, 12(1):55–89, 2009.
25. J.M. Thompson and K.A. Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annuals of Operations Research*, 63:105–128, 1996.
26. Y. Yang and S. Petrovic. A novel similarity measure for heuristic selection in examination timetabling. In *Practice and Theory of Automated Timetabling: Selected papers from the 5th International Conference .Lecture Notes in Computer Science*, volume 3616, pages 377–396, 2005.

Iterated Heuristic Algorithms for the Classroom Assignment Problem

Ademir Aparecido Constantino[1*], Walter Marcondes Filho[1], Dario Landa-Silva[2]

[1]*Department of Computer Science*
State University of Maringá, Brazil
[2]*ASAP Research Group*
School of Computer Science
University of Nottingham, UK

Abstract

We tackle the classroom assignment problem in a large University with the objective of minimising the total distance between all classrooms assigned to teaching activities in the same course. Additional requirements that should be satisfied include: making an efficient utilisation of the space, satisfying room preferences and complying with other administrative requirements. We present two iterated heuristic approaches, each one consisting of an iterative resolution of an assignment problem (the classical assignment problem in the first approach and the bottleneck assignment problem in the second approach) and a third algorithm based on the Variable Neighbourhood Search (VNS) meta-heuristic. We also present and discuss experimental results using real-world data from three consecutive academic sessions.

Key words: classroom assignment problem, iterated heuristic algorithm, variable neighbourhood search.

*Corresponding author: ademir@din.uem.br

1 Introduction

The classroom assignment problem in an academic institution refers to assigning classes, that meet at different timeslots, to rooms while respecting a series of operational restrictions and preferences (Carter and Covey 1992). This paper deals with a real-world classroom assignment problem from a large university involving many courses and classrooms. In our case, like in many other scenarios, the underlying course timetabling problem is solved in two phases (Carter and Laporte 1998). In the first phase, timetables are constructed for each department and each course. Since different courses can share some rooms, the availability of rooms is usually not considered in this first phase (although some courses might have priority for using certain rooms). In the second phase, rooms are assigned centrally to all courses based on the timetables produced in the first phase.

Although the classroom assignment problem is usually part of the well-known university course timetabling problem, it is also a very difficult problem and it has not been investigated on its own so extensively in the timetabling literature. Abdennadher et al. (2000) tackled this problem independently from the associated course timetabling problem and using constraint logic programming. Martinez-Alfaro et al. (1996) employed simulated annealing to assign classrooms to a large number of classes in a University in Mexico. Many times, the classroom assignment problem is tackled as part of the University course timetabling problem or the School timetabling problem (see Adriaen et al. 2006, Dammak et al 2006 and Schaefer 1999).

Carter and Tovey (1992) studied the classroom assignment problem and discussed its computational complexity. They suggested two versions of the problem, *interval* problem and *nointerval* problem, depending on how the concept of class is defined. In the *interval* problem, classes meet only once a week while in the *noninterval* (also called *multiday*) problem classes can meet more than once a week. Furthermore, when a class meets more than once a week, every meeting should occur in the same room. Following this classification, our work deals with a *noninterval classroom assignment problem*. Carter and Tovey (1992) showed

that this problem is NP-complete even for the *satisfice* case in which the problem is to find a feasible solution.

This paper is organized in 5 sections. Section 2 describes the particular classroom assignment problem tackled in this paper. Section 3 introduces some definitions and the proposed algorithms. Section 4 gives an overview of the implementation and the results. Finally, we conclude the paper in Section 5.

2 Problem Description

This work is based on the timetabling problem faced by a public higher education institution which is divided in several administrative centres and each containing related departments. Departments are responsible for offering and coordinating the various courses within their competence. Specifically, the institution is divided in 7 administrative centres and 34 departments. A total of 49 courses are on offer with approximately 4,000 subjects/sections offered to serve approximately 16,500 enrolled undergraduate students. There are 200 available classrooms for lectures plus special rooms or laboratories for practical classes. These practical classes have the special rooms assigned locally by their own departments and hence are not considered as part of the classroom assignment problem tackled here.

Despite this administrative division, the assignment of classrooms is responsibility of the institution's central administration. Students' transfers and adjustments may occur some days before the classes start. This situation makes the problem more difficult because prior assignments might need to be modified and this provokes operational administrative problems.

When assigning classrooms to classes, there are a number of restrictions and special needs for resources which hinder the classroom distribution. Several requirements must be taken into consideration:

1. Except for lectures resulting from the union of groups with practical lessons, only one lesson can be assigned in the same classroom at the same time. The classroom must be accessible to groups in which there are students with special needs.

2. Except for some subjects determined by the course, the number of students in a classroom must not be superior to its capacity.
3. Each course must have a defined geographic area for their academic activities and this serves as reference for the classroom assignment. The goal is to concentrate all classes in the same course within a geographic area of the campus.
4. Classes must be assigned to classrooms numbered according to the class year, i.e. freshman, sophomore, junior or senior year.
5. All the weekly meetings of a class should be preferably assigned to the same classroom. This facture increase the difficulty to solve the problem (*noninterval* case) as it was discussed by Carter and Tovey (1992).

The goal is to assign all the groups of all the subjects and courses to classrooms, maximizing the concentration of students of the same course within a geographical area, thus, minimizing the movement of students inside the campus while also obeying the abovementioned restrictions. Notice that requirements 1, 3 and 4 are considered preferences. In addition, some courses have preference for certain classroorns, these preferences are incorporated into the cost function (see Section 3). According to Carter and Tovey (1992) these preferences are non-monotonic (arbitrary) and increase the complexity of the assignment problem. The present work proposes the use of heuristic algorithms to solve this problem.

2.1 Definitions

There are 6 timeslots every weekday for a total of 34 timeslots per week, as shown in Table 1. Note that these 34 slots are available in each week of the entire academic year and since the allocations are the same for every week, then the solution for one week is all that is needed.

Table 1. Definition of timeslots during a week.

Period	Hour	Week					
		Mon	Tur	Wed	Thu	Fri	Sat
Morning	07:45 - 09:15	1	7	13	19	25	31
	09:30 - 11:45	2	8	14	20	26	32
Afernoon	13:30 - 15:10	3	9	15	21	27	33
	15:30 - 17:10	4	10	16	22	28	34
Night	19:30 - 21:10	5	11	17	23	29	-
	21:30 - 23:00	6	12	18	24	30	-

Consider the following notations for the indices:

m =1...M for the timeslots with M = 34,

k = 1...K for the courses,

t = 1...T_m for the groups (classes) with their timetable in timeslot m,

s = 1...S_k for the years of a course k,

l = 1...L for the classrooms.

A *classroom area* comprises of a building or an agglomerate of classrooms. Normally, the administrative centres have some preferred classroom areas for assigning classes in their courses. For each classroom area a Cartesian coordinate is given (area's central position) which is called the *area's point*.

It is desirable to assign all weekly lessons of a given group to the same classroom and also to have all the classrooms used by the same course and year within a geographic delimitation. In order to achieve this, we defined a *gravitational point* as a point in Cartesian coordinates or a scalar. The gravitational point serves as reference for the arranging of groups, years and courses within a geographic space. Three kinds of gravitational points are considered regarding the course, year and group and identified as: PGC_k, PGS_s and PGT_t, respectively. Each PGC_k corresponds to the Cartesian coordinates extracted from an image of the campus layout. The gravitational points PGS_s and PGT_t correspond to the classroom number. These values are used when attempting to arrange the years and groups following the order of the classroom numbers, i.e., a group in the initial year is assigned to the classrooms with numbers smaller than the other groups of posterior years. The gravitational points are empirically initialised. However, they are self-adjusted while the algorithms are executed.

3 Proposed Algorithms

In order to tackle the above classroom assignment problem (CAP), this section describes two iterated heuristic algorithms. The first one (CAP-A) is based on the successive resolution of the linear assignment problem whereas the second one (CAP-BA) is based on the successive resolution of the bottleneck assignment problem. A third proposed algorithm (CAP-VNS) is based on the variable

neighbourhood search (VNS) meta-heuristic and uses an initial solution obtained from the first phase of the CAP-A algorithm.

3.1. Algorithm CAP-A

This algorithm is based on the successive resolution of the linear assignment problem. The linear assignment problem is a classic linear programming problem equivalent to the minimum-cost perfect matching in a bipartite graph. For each timeslot *m* an instance of the assignment problem is created. The formulation of the assignment problem may be described as:

$$\text{Min } z_m = \sum_{t=1}^{T_m} \sum_{l=1}^{L} c_{tl}^m . x_{tl}^m$$

$$\text{s.t. } \sum_{t=1}^{T_m} x_{tl}^m = 1, \quad l = 1,..., L$$

$$\sum_{l=1}^{L} x_{tl}^m = 1, \quad t = 1,..., T_m$$

$$x_{tl}^m \in \{0,1\}, \quad t = 1,..., T_m, \quad l = 1,..., \text{L}$$

where c_{tl}^m is the cost of assigning group *t* to room *l* within timeslot *m*, and $x_{tl}^m = 1$ if group *t* is assigned to room *l* and 0 otherwise.

3.1.1 Phase 1

This phase consists of solving *M* assignment problems, one for each timeslot. Each assignment problem is defined by the square matrix $\left[c_{tl}^m\right]$; however, the number of groups may be smaller than the number of available classrooms. Thus, $T_m = T_m^{\text{Real}} + T_m^{Ficticious} = L$ will be considered, where T_m^{Real} is the actual number of existing groups and $T_m^{Ficticious}$ is the number of fictitious groups created to make the square matrix. Therefore, the cost matrix can be split in two parts, as shown in Fig 1, having their elements defined as:

> *Part I*: For $t = 1,2,\ldots, T_m^{\text{Real}}$ and $l = 1,2,\ldots,L$, we have $c_{tl}^m = f(t,l)$ where the function *f* (presented in the sequence) defines the cost of each assignment.

Part II: For t= $T_m^{Real}+1,\ldots,L$ (representing fictitious groups) and $l = 1,2,\ldots, L$, c_{tl}^m is the cost of assign a fictitious group in a classroom, in this case a large cost $c_{tl}^m = \infty$. As already mentioned above, the fictitious group is created to complement the matrix and make it square.

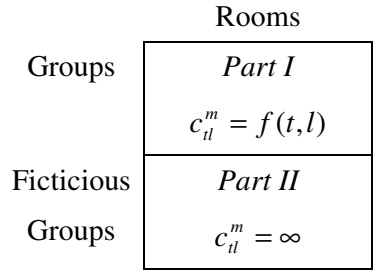


Fig. 1 Matrix basic structure

In *Part I* the function *f(t,l),* is defined as:

$$f(t,l) = \begin{cases} d_1(t,l) \text{ if } l \in SC(t) \text{ and } Size(t) \le Cap(l) \\ d_1(t,l) + p_1 \text{ if } l \in SC(t) \text{ and } Size(t) > Cap(l) \\ d_1(t,l) + p_2 \text{ if } l \notin SC(t) \text{ and } Size(t) \le Cap(l) \\ d_1(t,l) + p_1 + p_2 \text{ if } l \notin SC(t) \text{ and } Size(t) > Cap(l) \end{cases} \quad (1)$$

where:

- $d_1(t,l)$ = Euclidian distance from the PGC_k associated to group *t*, to the area's point related to room *l.*
- *SC*(*t*) is the group of classrooms with accessibility for the group *t* and their priority use is assigned to the courses from the administrative centre to which the group *t* is linked.
- *Size*(*t*) is the number of students in group *t*.
- *Cap*(*l*) is the number of students that the classroom *j* can seat.
- p_1 is the penalty applied when the classroom size does not serve the group's need. This penalty has been empirically defined as $2x10^3$.
- p_2 is the penalty defined as the biggest distance between classroom areas which belong to the same administrative centre to which the group *t* belongs. The penalty serves the purpose of forcing group *t* to be assigned to a room *l* belonging to *SC*(t).

An iteration of this phase involves the resolution of *M* assignment problems, one for each timeslot. In the first iteration, the PGC_k is empirically defined, normally a

point next to the classroom area desired for the course. For the following iterations, PGC_k is the average point estimated from the coordinates among all the classroom areas used for course k in the previous iteration. This procedure is repeated until the PGC_k of all the courses are not modified.

3.1.2 Phase 2

The purpose of this phase is to gather the groups of the same academic year in a course following the order by which the rooms are numbered, e.g. groups in the first year are in rooms with numbers smaller than the groups of the next academic year.

The structure of the cost matrix used in this phase is the same as in the previous phase, although the cost formation is slightly different, as follows:

$$f(t,l) = \begin{cases} d_1(t,l) + d_2(t,l) \text{ if } l \in SC(t) \text{ and } Size(t) \leq Cap(l) \\ d_1(t,l) + d_2(t,l) + p_1 \text{ if } l \in SC(t) \text{ and } Size(t) > Cap(l) \\ d_1(t,l) + d_2(t,l) + p_2 \text{ if } l \notin SC(t) \text{ and } Size(t) \leq Cap(l) \\ d_1(t,l) + d_2(t,l) + p_1 + p_2 \text{ if } l \notin SC(t) \text{ and } Size(t) > Cap(l) \end{cases} \quad (2)$$

where:

- $d_2(t,l) = | PGS_s - Num(l)|$, considering PGS_s the gravitational point of the year s to which the group t is related and $Num(l)$ is room number l.

An iteration of this phase also solves M assignment problems. In the first iteration $PGS_s = s$, $s = 1 \ldots S_k$, for the course k related to the group t. In the following iterations, PGS_s will be the average value of all classroom numbers allocated to the year s. This procedure is repeated until the PGS_s of all the years of every course are not modified.

3.1.3 Phase 3

The goal of this phase is to rearrange the groups gathered in phase 2 following a correspondence order for the group regarding the room numbering, e.g., if the group number 1 has been allocated to room 101, it is desirable that the group number 2 is allocated to room 102.

As in phase 2, the cost matrix structure used is the same as in phase 1, with the cost defined as follows:

$$f(t,l)=\begin{cases} d_1(t,l)+d_2(t,l)+d_3(t,l) \text{ if } \ l\in SC(t) \text{ and } Size(t)\le Cap(l) \\ d_1(t,l)+d_2(t,l)+d_3(t,l)+p_1 \text{ if } l\in SC(t) \text{ and } Size(t)> Cap(l) \\ d_1(t,l)+d_2(t,l)+d_3(t,l)+p_2 \text{ if } \ l\notin SC(t) \text{ and } Size(t)\le Cap(l) \\ d_1(t,l)+d_2(t,l)+d_3(t,l)+p_1+p_2 \text{ if } \ l\notin SC(t) \text{ and } Size(t)> Cap(l) \end{cases} \quad (3)$$

where:

- $d_3(t,l) = |PGT - Num(l)|$, considering PGT the gravitational point of the group t and $Num(l)$ is room number l.

An iteration of this phase also solves M assignment problems. In the first iteration PGT_t = t. In the following iterations, PGT_t will be the average value of the numbers of all the rooms allocated for the M modules in the previous iteration. This procedure is repeated until the PGT_t of all groups of every course are not modified.

3.2 Algorithm CAP-BA

This algorithm is equivalent to the algorithm CAP-A with the difference that the linear assignment model is replaced by the bottleneck assignment model. The bottleneck assignment problem is formulated as follows:

$$\text{Min } Z_m$$

$$\text{s.t. } \sum_{t=1}^{T_m} x_{tl}^m = 1, \quad l=1,...,L$$

$$\sum_{l=1}^{L} x_{tl}^m = 1, \quad t=1,...,T_m$$

$$c_{tl}^m x_{tl}^m \le Z_m, \quad t=1,...,T_m, \quad l=1,...,\mathrm{L}$$

$$x_{tl}^m \in \{0,1\}, \quad t=1,...,T_m, \quad l=1,...,\mathrm{L}$$

The cost matrix $\left[c_{tl}^m\right]$ is defined in the same way as for the previous algorithm. While the linear assignment model minimises the cost sum of all assignments, the bottleneck assignment model minimises the cost of the biggest assignment.

3.3. Algorithm CAP-VNS

This algorithm is based on the variable neighbourhood search (VNS) meta-heuristic, a local search procedure that explores the solution space by systematically changing the neighborhood structure (Hansen and Mladenovic,

2001). R neighbourhoods are defined for the problem in hand, N_1, N_2,...,N_R and if the current solution is not improved using a particular neighbourhood, the next neighbourhood is explored and so on.

Then, our CAP-VNS algorithm starts with a solution obtained in phase 1 of the algorithm CAP-A. Four neighbourhood structures N_r (r = 1, 2, 3 and 4) were defined. A neighbour $N_r(s)$ is obtained by exploring every timeslot for every weekday, and then choosing another assignment at random (see below).

The iterative improvement strategy used is the best descent, i.e., all solutions s" around s' are assessed, and the one giving the best improvement is selected. The evaluation function of a solution, to be minimised, is defined as:

$$f(s) = \sum_{m=1}^{34} \sum_{t=1}^{t_m} \sum_{l=1}^{L} c_{tl}^{m} . x_{tl}^{m} \qquad (4)$$

where c_{tl}^{m} is defined as in phase 3 for the algorithm CAP-A.

The neighbourhood structures N_r can be defined as:

1. N_1: for each timeslot m = 1,...,M and for each classroom area (building) used in the solution, randomly select a classroom in the classroom area and move the groups to an idle room in the same area, if possible.
2. N_2: for each timeslot m = 1,...,M and for each classroom area used in the solution, randomly select two classrooms in the classroom area and change the groups from one room to another, if possible.
3. N_3: for each timeslot m = 1,...,M and for each course with classes in module m, randomly select two classrooms used for the same course (regardless of the classroom area) and interchange all the groups between the two classrooms, if possible.
4. N_4: for each timeslot m = 1,...,M and for each year with classes in module m, randomly select two classrooms used by the year of the same course (regardless of the classroom area) and have the groups change from one room to the other, if possible.

In each iteration of the CAP-VNS algorithm, every neighbourhood is explored and the algorithm stops when there is no improvement within *3* iterations. We then follow the VNS scheme presented in Fig 2.

Initialisation. Select the set of neighbourhood structures N_r (r = 1, 2, 3 and 4); find an initial solution s; choose a stopping condition;

Repeat the following sequence until the stopping condition is met:

(1) Set $r \leftarrow 1$;

(2) *Repeat* the following steps until $r = R$:

(a) Generate a random solution s' from the r^{th} neighbourhood of s ($s' \in N_r(s)$);

(b) *Local search*. Find the best neighbour s'' of s' ($s'' \in N_r(s')$);

(c) *Move or not*. If $f(s'') < f(s)$ then $s \leftarrow s''$ and $r \leftarrow 1$; otherwise, $r \leftarrow r+1$;

Fig 2. Steps of the VNS

4 Results and Analysis

To solve the linear assignment problem, the algorithm proposed by Carpaneto and Toth (1987) was implemented which combines the Hungarian method and the Shortest Augmenting Path method. To solve the bottleneck assignment problem, the algorithm presented by Carraresi and Gallo (1984) was used.

All computational experiments were performed using a PC AMD Atlhon at 2.4 MHz, with 1 GB RAM running on Windows XP. The definition of Cartesian coordinates, used to calculate the distance between classroom areas, was based on a sketch of the institution's campus layout with a drawing scale of 2 cm = 1 m (1:50). The algorithms were tested with real data from three consecutive academic years. The characteristics of the test data used are summarised in Table 2.

Table 2. Characteristics of the test instances

Year	Number of courses	Number of rooms	Number of groups	Number of students
2006	47	170	3,927	15,270
2007	48	192	4,016	16,530
2008	49	192	3,978	16,320

Tables 3-5 present the results obtained by the proposed algorithms applied to the 3 test instances. Column **Total Cost** results from applying the objective function (equation 4) defined in Section 3.3. The number of allocations which satisfied the classroom capacity restriction is shown in the column *Favourable Allocations* (FA) and the number that did not satisfy that restriction other are shown in the column *Unfavourable Allocations* (UF). Column **Iterations** corresponds to the number of times that each phase was executed in order to reach an improved solution.

Table 3. Results for the test instance corresponding to year 2006

Algorithm	Phase	Time hh:mm:ss	Total cost	FA	UF	Iterations
CAP-A	1	00:21:08	2,015,496	3,595	332	8
	2	00:04:52	1,751,583	3,595	332	2
	3	00:17:27	1,598,637	3,595	332	3
CAP-BA	1	00:12:18	2,632,801	3,586	341	4
	2	00:07:15	2,549,259	3,587	340	2
	3	00:18:05	2,507,436	3,591	336	3
CAP-VNS	Initial Solution	00:21:08	2,015,496	3,595	332	-
	Local Search	00:26:02	1,731,850	3,595	332	3

Table 4. Results for the test instance corresponding to year 2007

Algorithm	Phase	Time hh:mm:ss	Total cost	FA	UF	Iterations
CAP-A	1	00:12:10	1,979,099	3,708	308	6
	2	00:08:15	1,733,254	3,708	308	3
	3	00:18:41	1,581,791	3,708	308	3
CAP-BA	1	00:06:11	2,589,698	3,705	311	2
	2	00:07:13	2,529,129	3,705	311	2
	3	00:22:05	2,479,152	3,706	310	4
CAP-VNS	Initial Solution	00:12:10	1,979,099	3,708	308	-
	Local Search	00:35:27	1,693,173	3,708	308	4

Table 5. Results for the test instance corresponding to year 2008

Algorithm	Phase	Time hh:mm:ss	Total Cost	FA	UF	Iterations
CAP-A	1	00:11:30	1,960,146	3,677	301	6
	2	00:09:17	1,697,943	3,677	301	3
	3	00:17:09	1,580,312	3,677	301	3
CAP-BA	1	00:06:02	2,589,036	3,666	312	2
	2	00:05:43	2,506,241	3,669	309	2
	3	00:18:22	2,493,253	3,671	307	3
CAP-VNS	Initial Solution	00:11:30	1,960,146	3,677	301	-
	Local Search	00:34:47	1,690,372	3,677	301	4

Tables 6-8 summarise the results achieved by the three algorithms proposed here together with the solution quality of the manually constructed assignments produced by the human planners. Besides the costs calculated using the objective function, these tables present the sums of the distances between the assigned rooms and the Gravitational Point of each course, measured in meters. Note that these distances are calculated based on a sketch of the campus layout. The minimum, maximum and average distances are also shown.

Table 6. Comparing results for the test instance corresponding to year 2006

Approach	Total cost	FA	UA	Total distance	Minimum Distance	Average distance	Maximum distance
CAP-A	1,598,637	3,595	332	432,855	0	158	1,680
CAP-BA	2,507,436	3,591	336	1,006,023	0	270	1,640
CAP-VNS	1,731,850	3,595	332	577,379	0	223	1,710
Manual	2,295,242	3,293	634	1,010,172	0	265	1,664

Table 7. Comparing results for the test instance corresponding to year 2007

Approach	Total cost	FA	UA	Total distance	Minimum Distance	Average distance	Maximum distance
CAP-A	1,581,791	3,708	308	506,620	0	172	1,762
CAP-BA	2,479,152	3,706	310	1,152,620	0	289	1,724
CAP-VNS	1,693,173	3,708	308	673,983	0	227	1,762
Manual	2,282,502	3,385	631	1,099,214	0	273	1,675

Table 8. Comparing results for the test instance corresponding to year 2008

Approach	Total cost	FA	UA	Total distance	Minimum distance	Average distance	Maximum distance
CAP-A	1,580,312	3,677	301	504,543	0	167	1,724
CAP-BA	2,493,253	3,671	307	1,149,892	0	286	1,724
CAP-VNS	1,690,372	3,677	301	673,057	0	225	1,724
Manual	2,281,593	3,348	630	1,098,053	0	271	1,675

It can be observed that the algorithms CAP-A and CAP-VNS achieved the best results overall. By comparing these results with those obtained manually by the institution, a very considerable improvement in the quality of the solutions can be observed, mainly with respect to the number of unfavourable allocations (UA).

5 Conclusions

In this paper, we tackled a real-world classroom assignment problem and proposed three algorithms: two iterated heuristic algorithms based on successive resolution of (linear and bottleneck) assignment problems and one algorithm based on VNS meta-heuristic. While the first and third algorithms try to minimise the total distance, the second one tries to minimise the maximum distance (min-max problem). Overall, the CAP-A algorithm performed better and reduced by more than 50% the total distance between classrooms of the same course and it also reduced considerably the number of unfavourable allocations when compared to previous manual solutions.

The computational performance of the proposed algorithms was very satisfactory regarding both solution quality and computational time. The computational time of approximately 30 to 40 minutes is quite acceptable since constructing a manual resolution for the problem can take days or weeks of work.

It is particularly important to note that CAP-A and CAP-BA are both deterministic algorithms, so, given a particular input, they always give the same solutions, while the CAP-VNS is a stochastic algorithm.

In particular, both algorithms CAP-A and CAP-BA are quite flexible with respect to the incorporation of new constraints. The required adaptation to accommodate new rules is only on the construction of the cost matrix for each assignment problem, but no change is required on the heuristic algorithms. A new hard constraint can be incorporated as an infinity cost in the cost matrix, whilst a soft constraint would be given a finite penalty cost.

Acknowledgements

This work was partly supported by the CNPq (Brazilian National Council for Scientific and Technological Development) and CAPES (Brazilian Ministry of Education).

References

Abdennadher SM, Saft S, Will S. (2000) Classroom Assignment Using Constraint Logic Programming. In: Proceedings of the Second International Conference and Exhibition on the Practical Application of Constraint Technologies and Logic Programming (PACLP 2000), Manchester.

Adriaen, M., De Causmaecker, P., Demeester, P., Vanden Berghe, G. (2006). Tackling the University Course Timetabling Problem with an Aggregation Approach. In: Proceedings of The 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), 330-335, Brno, Czech Republic.

Carpaneto, G.; Toth, P (1987). Primal-dual algorithms for the assignment problem. Discrete Applied Mathematics, vol. 18, 137-153, North-Holland.

Carraresi, P.; Gallo, G. (1984). A Multi-level Bottleneck Assignment Approach to the Bus Drivers' Rostering Problem. European Journal of Operations Research, vol. 16, 163-173.

Carter, M. W.; Tovey, C. A. (1992). When is the Classroom Assignment Problem Hard? Operations Research, vol. 40, 28-41.

Carter, M. W.; Laporte, G. (1998). Recent Developments in Practical Course Timetabling. In: The Practice and Theory of Automated Timetabling II: Selected Papers From the 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT 1997), LNCS 1408, Springer, 3-19.

Dammak, A.; Elloumi, A.; Kamoun, H. (2006). Classroom Assignment for Exam Timetabling. Advances in Engineering Software, vol. 37, no. 10, 659-666 .

Hansen, P.; Mladenovic, N. (2001). Variable Neighbourhood Search: Principles and Applications. European Journal of Operational Research, vol. 130, no. 3, 449-467.

Martinez-Alfaro, H.; Minero, J.; Alanis, G.E.; Leal, N.A.; Avila, I.G. (1996). Using Simulated Annealing to Solve the Classroom Assignment Problem. In: Proceedings of the 1st Joint Conference on Intelligent Systems/ISAI/IFIS, 370-377.

Schaefer, A. (1999). A Survey of Automated Timetabling. Artificial Intelligence Review, vol. 13, 87-127.

A Variable Neighborhood Search based Matheuristic for Nurse Rostering Problems

Federico Della Croce · Fabio Salassa

Abstract A practical nurse rostering problem, which arises at a ward of an Italian private hospital, is considered. In this problem, it is required each month to generate the nursing staff shifts subject to various requirements. A matheuristic approach is introduced, based on a set of neighborhoods searched by a commercial integer programming solver within a defined global time limit. Generally speaking, a matheuristic algorithm is a heuristic algorithm that uses non trivial optimization and mathematical programming tools to explore the solutions space with the aim of analyzing large scale neighborhoods. The solutions computed by the proposed procedure are compared to the solutions achieved by the pure solver within the same time limit. The results show that the proposed solution approach outperforms the solver in terms of solution quality.

Keywords Variable neighborhood search · Matheuristics · Timetabling · Nurse Rostering Problem

1 Introduction

The paper pertains to a nurse rostering problem, which occurs at a private hospital located in Turin, Italy. The problem consists in optimally assigning a working shift or a day off to each nurse, on each day of a month, according to several contractual and operational requirements. The problem belongs to the family of timetabling problems [5], [6] and [16]. Many works have been published on the nurse rostering problem since the pioneering works of Warner [17] and Miller [14] and the proposed approaches are mainly based on constraint programming and metaheuristic procedures (see for instance [4] and [10]). In [1], a nurse rostering problem similar to the one considered here

F. Salassa
DISPEA - Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Tel.: +39 0110907206
Fax: +39 0110907299
E-mail: fabio.salassa@polito.it

F. Della Croce
DAI - Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
Tel.: +39 0110907059
Fax: +39 0110907099
E-mail: federico.dellacroce@polito.it

was tackled by means of a metaheuristic approach. In particular, in the metaheuristic framework, a simple and effective metaheuristic for combinatorial and global optimization, called variable neighborhood search (VNS) has been successfully applied to solve both general mixed integer programming problems (MIPs) [15] [11] and timetabling problems [3].
Combinatorial optimization problems, such as timetabling problems, usually may be formulated in different ways. Mathematical programming formulations such as integer programs are very popular, since they make possible the use of general-purpose Integer Linear Programming (ILP) solvers, independently of problem specific properties. However, in some hard cases, for instance when the number of variables becomes too large, solvers might not be an adequate choice [12].
Metaheuristic approaches, instead, rely on formulations adapted to be able to handle the special concrete combinatorial optimization problems that have to be faced thus loosing the advantage of working in a generic modelling framework, in fact, even slight changes of the problem description can cause a complete redesign of data structures and algorithms. Indeed, the adaptation of the approach proposed in [1], even if the constraints set was quite similar, proved to be quite tricky. On the other hand, the improvement of ILP solvers in the recent years have made them already competitive (by simply adding a time limit) to metaheuristic approaches in the search of suboptimal solutions within limited CPU time. Further, whenever an ILP model is available, the addition of new constraints or the modification of the objective function is often straightforward.
Recently [9], [13] a new topic has attracted the attention of the community of researchers, the so called Matheuristics. Matheuristics are heuristics algorithms made by the inter operation of metaheuristics and mathematical programming techniques. An essential feature is the exploitation in some part of the algorithms of features derived from the mathematical model of the problem of interest [2].
The aim of the present work is, then, to demonstrate the applicability, presenting comparison results, of a VNS based matheuristic for solving hard timetabling problems. As ILP solver we use XPRESS IVE version 1.19 from Fair Isaac Corporation.
The article is organized as follows. In Section 2 the problem is described. In Section 3 the matheuristic solution approach is described. Section 4 is devoted to computational testing and comparison. Section 5 concludes the paper with final remarks.

2 Problem Description

The problem considered here is based on a real situation encountered in a ward of a private hospital in Turin. At the end of each month, the nurses' shifts for the following month must be scheduled. The ward is made up of a given number of hired nurses, but the timetable of these nurses is not always sufficient to cover the legal minimum of personnel presence in every shift. For that reason the hospital management can make use of freelance nursing staff to cope with the demand of personnel inducing however an extra cost. The problem consists, thus, in optimally assigning a working shift or a day off to each nurse, on each day, according to several requirements in order to reduce the outsourced work. In the considered case all nurses have the same skills' level except for newly recruited personnel. This issue is resolved imposing a period of shadowing between specific nurses or setting the incompatibility in the same shift for specific workers (the newly recruited nurses).

Table 1 Shift types and demand

	Demand						
Shift type	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Morning	3	3	3	3	3	3	3
Afternoon	3	3	3	3	3	3	3
Night	2	2	2	2	2	2	2

In that hospital, the nurses timetabling was generated manually every month by the hospital management, but it was too time consuming leading often to unfair timetables and to an overuse of freelance staff. Indeed the hospital management wanted to try other approaches to overcome those issues and decided then to test an automated timetabling generation procedure in order to better manage the hired personnel and to reduce, as much as possible, the number of outsourced shifts. We were then asked to realize a tool to cope with these requirements.

2.1 Types of shifts

There are five different kinds of shifts considered in the hospital: three of them are working shifts and two are off-duty shifts. During the last week of each month, the nurses can provide to the management their requests of off-duty shifts. In general, no request of specific working shifts are accepted by the hospital management. The working shifts, all lasting 8 hours, are:

- Morning shift (M)
- Afternoon shift (A)
- Night shift (N)

The off-duty shifts are:

- Rest (R)
- Off (O)

The difference between Rest and Off shifts is basically that a number of Rest shifts can be both requested by staff and assigned by the management, Off shifts can only be requested by the nursing hired staff.

2.2 Constraints Description

Hereafter is presented the complete list of constraints integrated in the support tool realized for the hospital.

(C1) The number of R shifts per month must be equal to a predefined value provided by the management.
(C2) The nurses' requirements related to O shifts and R shifts must be satisfied.
(C3) A nurse cannot work consecutively for more than D days.
(C4) Specific shifts types must be allocated in sets of minimum L_k and maximum S_k consecutive days for each shift type k.

(C5) After a set of N shifts, there must be a given number E of R shifts.
(C6) An interval of at least P days (working or not) must occur between two N shifts sets.
(C7) A minimum number of nurses must be guaranteed for each working shift. This parameter, provided by the management, may differ from shift to shift and from day to day. An example of personnel demand is summarized in Table 1.
(C8) Forbidden sequences of shifts (e.g. N-R-N, N-M, N-A, N-O, etc.).
(C9) A balanced assignment of M, A and N shifts must be guaranteed among the nurses.
(C10) Working shifts and days off during weekends mus be evenly assigned.
(C11) At least two off-duty weekends for each nurse/per month.
(C12) All constraints must be respected considering the last days of the previous month.
(C13) Some nurses must (not) work together in particular set of days and for particular shifts (shadowing/incompatibility period).

The considered problem can be easily formulated as an ILP model. Indeed, with n nurses and m days in a month, it is sufficient to introduce a set of 0/1 variables denoted $x_{i,j,k}$ ($i = 1, ...n$, $j = 1, ...m$, $k = 1, ...5$) indicating if nurse i is assigned to shift k ($k = 1$: morning, $k = 2$: afternoon, $k = 3$: night, $k = 4$: rest, $k = 5$: off-duty) on day j. Correspondingly, an integer variable $y_{j,k}$ is introduced indicating the number of freelance nurses used on working shift k on day j. The objective function is then

$$\min \sum_{j=1}^{m} \sum_{k=1}^{3} y_{j,k}.$$

We do not present here the complete ILP formulation (see [7] for details) but just present as an example the formulation of constraint C7 related to the morning shift. If we denote by M_j the requirement of nurses for the morning shift of day j, we have

$$\sum_{i=1}^{n} x_{i,j,1} + y_{j,1} \geq M_j \quad \forall j = 1, ...m.$$

For the considered hospital ward, we have $n = 10$. Correspondingly, there are approximately 90 integer variables $y_{j,k}$ and 1200 binary variables $x_{i,j,k}$ (the off-duty shifts are pre-determined). Also, there are approximately 2000 constraints.

3 The VNS Matheuristic Based Approach

The pure ILP model solved by means of the ILP solver (XPRESS) already gave satisfactory results so that the hospital management decided after the very first results to stop creating nurse timetables manually, considering the results of the ILP solver much better in terms of required outsource shifts (that does not consume personnel time with respect to the generation of the timetable). However, the hospital management wanted also to generate more than one timetable, for the ward under analysis, considering, for example, different weekly demands of nurses.
During the testing of that feature for the support tool we found that, sometimes, the considered ILP solver was not performing sufficiently well as for some problems the

gap between upper bound (the feasible solution obtained within the considered time limit) and lower bound (the bound given by the solver at the end of the considered time limit) was significant. Hence, a new matheuristic approach based on VNS was proposed to be compared to the ILP solver.

3.1 Description of the matheuristic approach

The core of our approach is made by two further constraints: the first one states that the number of variables $x_{i,j,k}$ that can change their value from the current solution is less than a parameter value K, that is the neighborhood size is controlled by parameter K. Indeed, K is the maximum Hamming distance between the current solution and all feasible solutions of the neighborhood, or, in other words, K is the maximum number of changing shifts in the new solution with respect to the incumbent solution.
The second constraint states the structure of the neighborhoods. In particular we have defined 12 types of neighborhoods, for the considered problem, each of different size. For each of these subproblems, the ILP solver used as a black box is applied searching for the optimal solution (limiting though the search by means of a given time limit). In order to escape from possible local minima, we integrate in our approach, as an additional feature, the use of neighborhhod structures not only limited by the value K but also by the value 2*K, expanding, in that way, the search space thus generating globally 24 different neighborhoods.

We have used the variable fixing method for generating our neighborhoods defining $N_{J_g,T}(x)$ as the structure of neighbors of x.
In particular $g = 1..3$ and $J = 1..30$ days of the current month where $g = 1$ represents the first 10 days of the month, $g = 2$ the second decade and $g = 3$ the last ten days. $T = 1..5$ represents the shift types and in our matheuristic are taken into account with the sequence order:

- fix all T shifts of all i nurses for a period J_g of days and K value
- fix all T shifts of all i nurses for a period J_g of days and K2 value
- fix $T = 1$ shift of all i nurses for a period J_g of days and K value
- fix $T = 2$ shift of all i nurses for a period J_g of days and K value
- fix $T = 3$ shift of all i nurses for a period J_g of days and K value
- fix $T = 1$ shift of all i nurses for a period J_g of days and K2 value
- fix $T = 2$ shift of all i nurses for a period J_g of days and K2 value
- fix $T = 3$ shift of all i nurses for a period J_g of days and K2 value

We decided not to include the off-duty shifts because they are mainly requested by nurses so a priori fixed in the model.
The mixing of these two constraints generates a sequence of increasing size neighborhoods at each iteration. Note that after adding one (or several) constraints the resulting MIP has the same structure as the original MIP but a smaller solution space.
The strength of a matheuristic procedure can be seen also in another relevant aspect: the effort spent on the generation of different neighborhoods and on the analysis of their quality it is noticeably lower than the majority of pure heuristics or metaheurisics approaches as we had to add two constraints to the original model to implement our neighborhood structure. Hereafter is the pseudo code of our matheuristic procedure.

Pseudocode

```
Begin
(1) choose an initial feasible solution z
(2) iter:=1
(3) while iter <=24 do
  (4) |z - x|<=K(iter)
  (5) set the variables belonging to N(iter) to their corresponding values in z
  (6) solve subproblems within local time limit
  (7) if f(x) < f(z) then do
     (8) update solution
     (9) iter:=1
  (10) else do
     (11) iter:=iter+1
  (12) end if
  (13) if the overall CPU time is greater than 3600 s. or iter>24 then
     (14) EXIT
  (15) end if
(16) end-do
End
```

3.2 Implementation

It is important to underline that the difference between the current solution and the incumbent solution, as our decision variables are binary, corresponds to the Hamming distance between two strings of [0,1] values and, in our approach, that distance must be lower than K. It is possible to model this feature in a very compact way by taking advantage of the linear programming formulation. For instance, by denoting with $z_{i,j,k}$ the value of the $x_{i,j,k}$ variables in the incumbent solution, the Hamming distance constraint can be modeled as follows.

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{5} z_{i,j,k}(1 - x_{i,j,k}) + (1 - z_{i,j,k})x_{i,j,k} \leq K.$$

3.3 Parameters Settings

The parameters present in our matheuristic are the value K, and time T for each iteration. The K values used in all our tests are K1= 50 and K2=100. Even though these values of K could appear very large, they were the ones that gave best results in preliminary testing. This can be explained by the strong structure of the ILP models related to timetabling problems. The global time stopping criterion, as told before, has been set to 3600 seconds which has been considered by hospital management a reasonable time window to wait for solutions of hard instances. In order to evaluate the quality of our approach, we decided to generate a number of instances similar to the real ones. We also decided to generate different size instances for testing in order to widen the possibilities of application of the approach. For that reason we used different time limit at each iteration with relation to the different sizes of the models.

1. Instances with 10 hired nurses - time limit = 60 seconds for each iteration

2. Instances with 20 hired nurses - time limit = 120 seconds for each iteration
3. Instances with 30 hired nurses - time limit = 240 seconds for each iteration

3.4 The Instances Generation Process

This paragraph presents a short description of the rules used to randomly generate instances[1]. The random generation concerned:

1. the last 5 days of the previous month, respecting all constraints, i.e. without any schedule violation;
2. the requests of R shifts and O shifts of the current month are generated nurse by nurse: for each nurse the requests are drawn from a uniform distribution according to the following probabilities:
   - 60% no requests
   - 5% 15 days of O shifts
   - 10% seven days of O shifts
   - 20% three days of R shifts
     - 50% three single R shifts
     - 50% two consecutive R shifts and a single R shift.
   - 5% two consecutive R shifts
3. shadowing or incompatibility constraints (from 0 to 4 w.r.t. nurses number);
4. different nurses demand structures (3-3-2; 4-3-2; 3-3-3; 3-2-1 mixing also weekday and weekend demands) multiplied by the size of the instances (e.g. 30 nurses, request 9-9-6).

4 Computational results

The proposed procedure was tested on 20 instances generated as mentioned before. The approach was implemented directly in XPRESS and tested on a Pentium IV Quad Core at 2.4 GHz. The results and the comparison with XPRESS with the time limit of 60 minutes are presented in Table 2. In the table, the first column depicts the instance size and name, the second column depicts the lower bound computed by XPRESS after 60 minutes of CPU time and the third and fourth column indicate the cost function values (the total number of outsourced shifts) of the solutions computed by MathVNS and the pure XPRESS solver respectively within a time limit of 60 minutes.
From this table we notice that the MathVNS procedure reaches solutions that are better than or equal to those of the pure Xpress solver in all cases but one. These results indicate that the proposed MathVNS approach is a viable option in handling nurses rostering problems.

5 Concluding Remarks

A VNS based matheuristic approach was proposed for a real life nurse rostering problem. The local search steps work on a smaller solution space specifying that at most

[1] Instances are available upon request from authors

Table 2 Comparing MathVNS vs XPRESS

Instance size/name	LP	MathVNS (1h)	Xpress (1h)
10n*1	43	54	54
10n*2	71	71	72
10n*3	9	9	9
10n*4	21	27	27
10n*5	72	80	80
10n*6	44	53	53
10n*7	24	28	30
10n*8	40	43	46
10n*9	21	26	27
10n*10	74	81	82
20n*1	51	68	71
20n*2	141	141	141
20n*3	56	64	67
20n*4	88	102	108
20n*5	35	46	50
30n*1	94	103	107
30n*2	139	158	161
30n*3	73	83	85
30n*4	35	45	45
30n*5	135	161	160

K decision variables can be complemented and fixing iteratively subsets of them. To do so we have exploited the strength of the mathematical programming formulation making use of a compact modellization of the Hamming distance between two strings and drawing advantage from the capability of generating and analyzing different neighborhoods' structures and sizes in a very short time. The proposed procedure shows a very good behavior in terms of solutions quality with CPU time limit of 60 minutes as presented by the achieved results. A software has been developed which is currently under testing in the hospital ward. This software can handle different nurse planning scenarios with different personnel requirements and different nurses' requests at the same time.

References

1. Bellanti F., Carello G., Della Croce F., Tadei R., A greedy-based neighborhood search approach to a nurse rostering problem, European Journal of Operational Research, 153 , pp. 28-40, (2004).
2. Boschetti M.A., Maniezzo V., Roffilli M., Bolufe Rohler A., Hybrid Metaheuristics, 6th International Workshop, pp., 171-177, Spriger,(2009).
3. Burke E., De Causmaecker P., Petrovic S., Vanden Berghe G., Variable neighborhood search for nurse rostering problems. In Metaheuristics: Computer Decision-Making, Applied Optimization, 86, Kluwer, pp.153-172, (2004).
4. Burke E. K., De Causmaecker P., Vanden Berghe G., Van Landeghem H., The State of the Art of Nurse Rostering, Journal of Scheduling, 7, pp. 441-499,(2004).
5. De Werra D., An introduction to timetabling, European Journal of Operational Research, 19, pp. 151-162, (1985).
6. De Werra D., The combinatorics of timetabling, European Journal of Operational Research, 96, pp. 504-513, (1997).
7. Della Croce F., Salassa F., A VNS based Matheuristic Procedure for Nurse Rostering Problem, Internal Report, D.A.I. Politecnico di Torino, (2009).
8. Ernst A.T., Jiang. H., Krishnamoorthy M., Sier D., Staff scheduling and rostering: A review of applications, methods and models, European Journal of Operational Research, 153, pp. 3-27, (2004).

9. Fischetti M., Lodi A., Local Branching, Mathematical Programming B, 98, 23-47, (2003).
10. Grobner M., Wilke P., Buttcher S., A Standard Framework for Timetabling Problems in Practice and Theory of Automated Timetabling IV, pp. 24-38, Springer, (2003).
11. Hansen P., Mladenovic N., Variable neighborhood search: Principles and applications, European Journal of Operational Research, 130, pp. 449-467, (2001).
12. Hansen P., Mladenovic N., Urosevic D., Variable neighborhood search and local branching, Computers and Operations Research, 33, pp. 3034-3045, (2006).
13. Maniezzo V., Stutzle T., Voss S., Matheuristics: Hybridizing Metaheuristics and Mathematical Programming, Annals of Information Systems, 10, Springer,(2009)
14. Miller H.E., Nurse scheduling using mathematical programming, Operations Research, 24, pp. 857-870, (1976).
15. Mladenovic N., Hansen P., Variable neighborhood search, Computers and Operations Research, 24, pp. 1097-1100, (1997).
16. Nanda R., Browner J., Introduction to Employee Scheduling, Van Nostrand Reinhold, New York, (1992).
17. Warner D.M., Scheduling nursing personnel according to nursing preference: A mathematical programming approach, Operations Research, 24, pp. 842-856, (1976).

On-line timetabling software

Florent Devin · Yannick Le Nir

Abstract Timetable design is a really important and difficult task. Timetable hand-building consumes a lot of time. In this paper we address two main difficulties of automatic timetabling, that is data acquisition and timetable computation. The former task is made using new advanced technologies in the area of Rich Internet Application. This offers very powerful, and easy to use, interfaces to acquire data. The latter task is the computation of the timetable itself. We use the constraint programming and one implementation in swi-prolog to compute the timetable. Finally we show some results of our application on a real case study.

Keywords Timetabling · CSP · Prolog · Java framework · ZK · Google API

1 Introduction

In this paper we present a timetabling software. Timetabling application can be split into two different parts, the design of a valid solution and the data acquisition. Many operational software need to hand-build a timetable. Computing helps are only given to verify constraints and to acquire input data.

To acquire data, we create an RIA[1]. This choice allows us to provide an original solution for timetabling. First we decide to use web services to allow us using a particular algorithm to solve the problem. Moreover by using web services we are able to interact with Google Calendar. At last, this use allows the software to be built in our IT system. The figure 1 shows the general architecture of our software. This distributed approach, and using an RIA allows us to delegate data acquisition as we will see.

Florent Devin · Yannick Le Nir
EISTI, 26 avenue des Lilas, 64062 Pau Cedex 9
Tel.: +33 5 59 14 85 34
Fax: +33 5 59 14 85 31
E-mail: florent.devin@eisti.fr

E-mail: yannick.lenir@eisti.fr

[1] Rich Internet Application

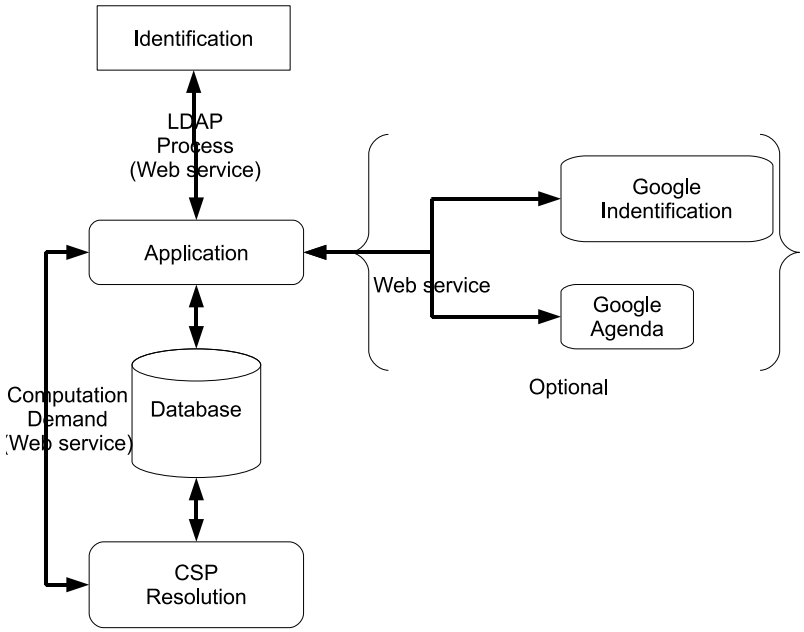


Fig. 1 General architecture system

The design part, in our solution, is made with an automated task as the solution of a CSP[2]. Such approach based on CSP has already been studied for examination and lectures timetabling in school and university (Abdennadher et al 2007)(Abbas and Tsang 2001). We present in the last sections of this paper an instantiation of our model in a similar way (university timetabling) with some computational results under global parameters variation. Other computational models can be used, as mentioned in (Qu et al 2009). We decide to choose CSP, that is easy to use and very powerful for timetabling applications (Wallace 1996), especially in our general architecture where compatibility and performance issues are not problematic.

2 Definitions

In this section, we introduce some definitions that we will use to present our solution for an on-line timetabling software in next sections.

The first notions we have to define are *timetable* and *time slot*. The timetable is defined as a consecutive list of time intervals. A time slot is the minimal time interval we can find on the timetable. Its duration is fixed and then it is only specified with its starting time.

Then, we have to define the *resources* we need to access, that is in our case the resources that describe the context of the timetable. It is specific to every timetable but belongs to one of the following categories:

[2] Constraint Satisfaction Problem

– main resources: elements to be planed in the timetable (lectures, talks, meetings, ...),
– static resources: elements that are already linked to main resources before the computation (peoples, holidays, ...),
– dynamic resources: elements that will be linked to main resources by the computation (rooms, materials, ...).

All these resources are the *descriptive resources* of the timetable. In our application, they are collected in a relational *database*.

Now, we can define the *availability* of the different descriptive resources of the database. The timetable is made of time slots, and then an availability can be associated to each resource as a starting time slot and an ending time slot. We can also define *unavailability* as the complementary of an availability. The (un)availabilities will be considered as *constraints* in the following, mainly in the section on CSP.

The last notion we have to define is the *timetabling*. It consists to an assignment of all the main resources of the database on the timetable for a specific period, with respect of all constraints on descriptive resources.

Some more specific definitions will follow in the next sections. However, these specifics terms are not a prerequisite for understanding.

3 Rich Internet Application for Timetabling

3.1 Motivation for rich interface

Whilst there is no need to argue for a tool for timetabling, we have to discuss why we have to create a rich interface. In an ideal world, all constraints are known long before they occur. In this same world, the constraints do not often change. But this never happens in reality, where very often constraints are modified. There is a real need to change the constraints, whatever the real motivation; for example, we need to provide a tool for specifying unavailability[3].

In this case, we have two choices:

– A centric application: This means that there is only one person who creates or edits the timetable. This amounts to saying also, that only one person takes responsibility for validating or invalidating the constraint. But this step is a little tricky, because you must contact the person who has created the constraint.
– A distributed application: That means you delegate checking constraints to the user.

The former approach has several disadvantages. Such as the problem of knowing the validity of the constraints. The latter approach allows all users to input or delete their constraints. This is a real advantage for the distributed approach.

Once we have chosen a distributed approach, again there are two choices:

– a classical third party application;
– a Rich Internet Application.

Rich Internet Application (RIA) does not require any installation on the client side. Also, neither a particular operating system nor a particular software. RIA allows users

[3] A constraint for a contributor can be an unavailability

to use the application without any requirement. Nevertheless using an RIA implies that you have to look after the security of your software(Lehtinen 2009), mainly if this one is accessible from the Internet.

3.2 Implementation

ZK is an open-source Ajax Web application framework written in Java(Seiler 2009). It uses the server centric approach(Yeh 2007) so that the communication between components is done by the engine. Security process is involve, synchronisation with a database is easier. In the other hand as ZK can use Java, we are able to use Hibernate framework. Also ZK provide a framework for mobiles(Yeh 2007). Even if we do not, for instance, provide a mobile interface this can be very useful for several contributor.

Timetabling involve a lot of contributors. Some of them will have some constraints to input, modify or delete, others will have to create the planning, etc. All of them need to use the RIA. In addition, we want to keep the job process in a single location. This can be done if you use a framework that keeps it on the server. For all these reasons, and (Yeh 2006), we choose to use the ZK framework[4].

3.3 Presentation

The entry point is the RIA. Currently in our application, users[5] have different roles:

- User: a contributor who is allowed to submit some constraints on his own timetable and view all timetables (rooms, class, users, etc).
- Admin : someone entitled to accept or refuse the constraints created by a user. The admin can also generate a timetable and export it into all contributors' Google Calendar[6], if they want to.

3.3.1 User

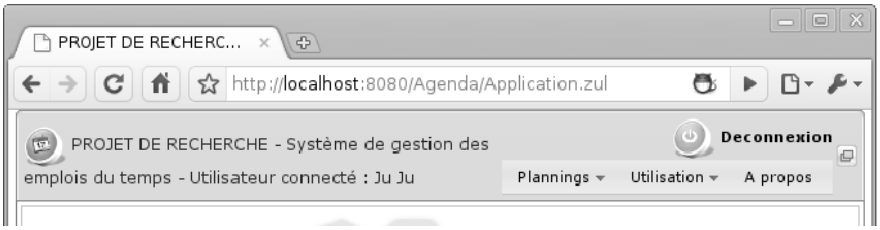


Fig. 2 User welcome screen

At present users have two options to enter their constraints: either through our application or through Google Calendar application. If they choose the second option,

[4] ZK framework : `http://zkoss.org`
[5] users : someone who have to use our application. We use the term of contributor as well.
[6] We will see the usage of Google Calendar, later on.

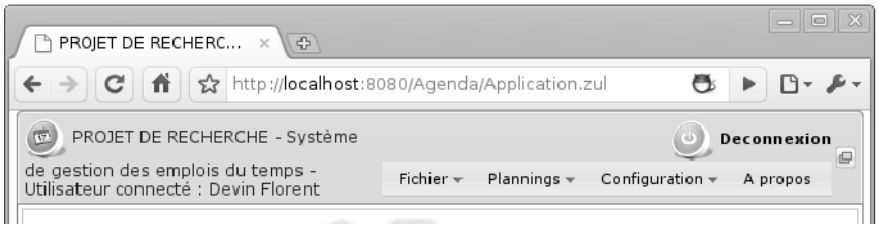


Fig. 3 Admin welcome screen

they have either to provide to the administrator a link to their Google Calendar, or to log in into our application once to save this URL. Once this is done, the users may put their constraints in their own calendar. Later on, the *admin* will validate or invalidate users' constraints. Validation by the *admin* is required by the policy of our institution and to avoid any abuse. To allow the *admin* to make better decisions, in case he has to invalidate a constraint, we plan to use a scale for fixing the degree of unavailability. As soon as the *admin* creates the timetable, the corresponding time slot will appear in all users' calendar. The same operation can be done in our application. There is also an interface to create, modify or delete any users' constraints as is shown in figure 6.

3.3.2 Admin

The *admin* work load is significantly reduce thanks to our application.

Creating a timetable by hand is an extremely time consuming tasks, which might take up to 3 days a week. This is a huge task, because of many reasons, for example: *A videoconferencing can be used for teaching.* This is a really strong constraint because it occurs in two different places. On another hand, in our institution a school term is about 16 weeks long, during which many courses last for about 20. Each course has different contributors and durations in a week, and in a period. This involves creating a different timetable for almost every week. On top of this, contributors' constraints may change according to their other activities, rendezvous, and so on.

Fig. 4 Lecture time-line

By using our application the task of the *admin* comes down to putting all lectures in a time-line, as is shown in figure 4. By doing this, the *admin* also specifies the number of courses per week, number of time slots devoted to practice, ... This simply task takes roughly one hour per term. Then he has to specify which contributor will give what lecture, and also which features are required, and for which class the lecture is addressing as is show in figure 5. This step may take up to four hours once. This is what we can call the initialisation phase.

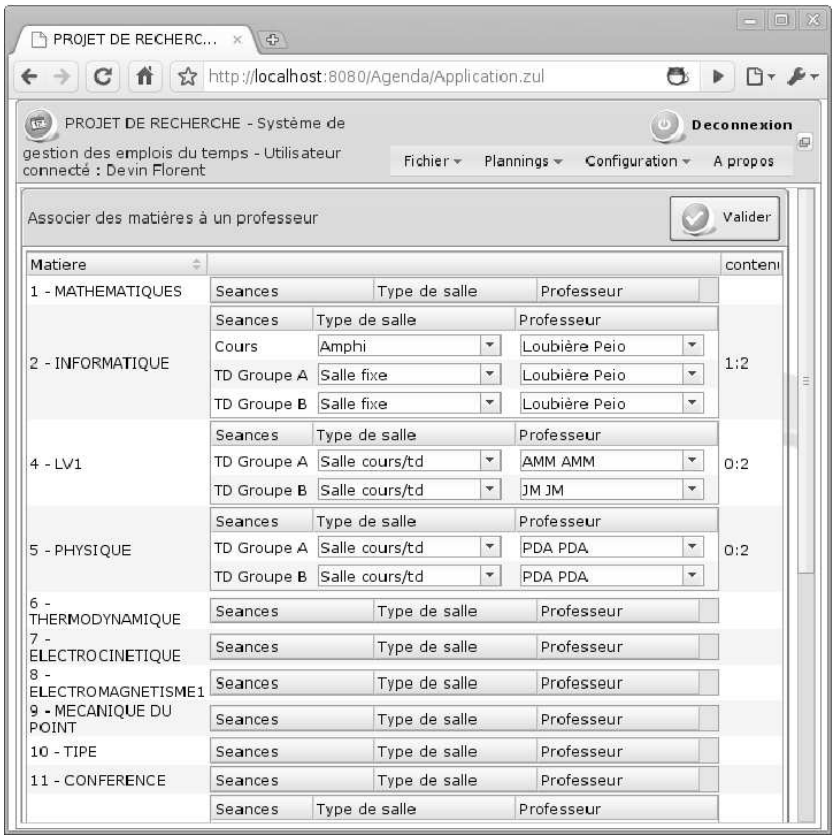
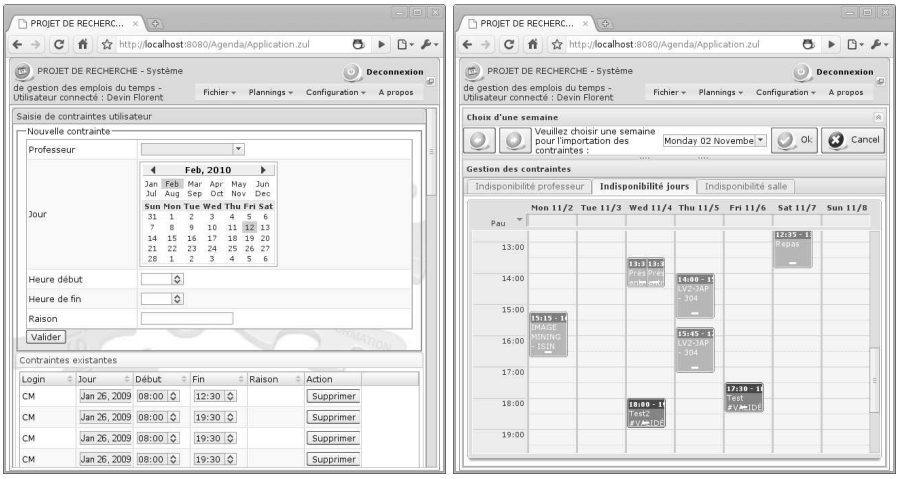


Fig. 5 Courses, features, contributor association

Now we have all the elements to compute an intermediate solution. The result is a timetable which does not take into consideration the users' constraints. If no users' constraint has been specified, this is a valid timetable. Otherwise the *admin* has to validate or invalidate users' constraints. To simplify this step, our application presents two different screens.

One is textual screen, figure 6(a), and another one is a dashboard, figure 6(b), which partitions all constraints into three categories:

– rooms (un)availability constraints;



(a) Constraints textual input screen shot (b) Constraints dashboard screen shot

Fig. 6 Constraints input

- contributors' constraints, as we have seen before;
- free session, holidays, which are used to set some empty time slots for student activities, or holidays.

For each category, the dash board uses a color code to show the state of the constraint. There are three possibilities:

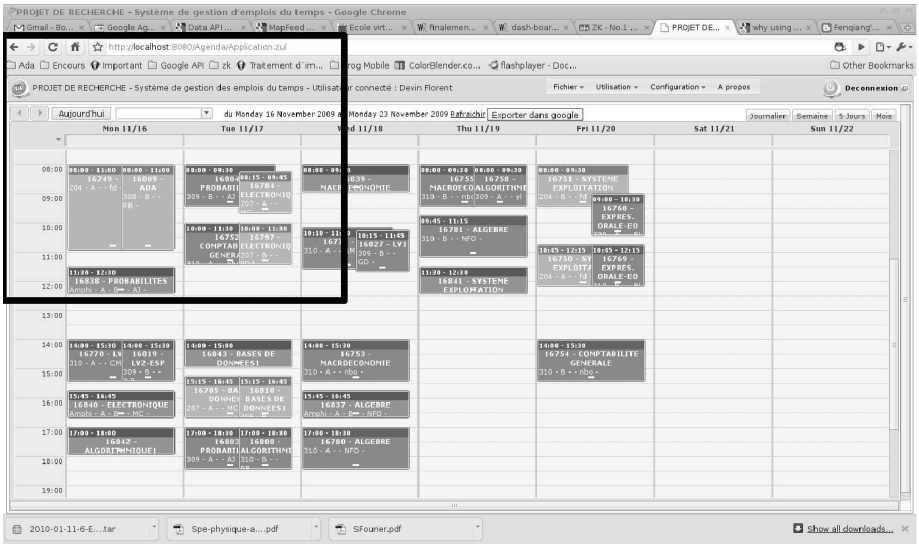
- Green : the constraint has already been validated. This usually means a recurrent constraint. The *admin* can still invalidate it.
- Orange : the constraint has never been validated. The *admin* must validate it if he wants to take it into account.
- Red : the constraint has been validated, but the contributor has changed it afterwards.

Another possibility is to fix a time slot. This will later be taken by the computation as a hard constraint. Eventually, the admin can create a real timetable, by invoking the computational service as mentioned before. Figure 7 shows a computed timetable involving all the required constraints.

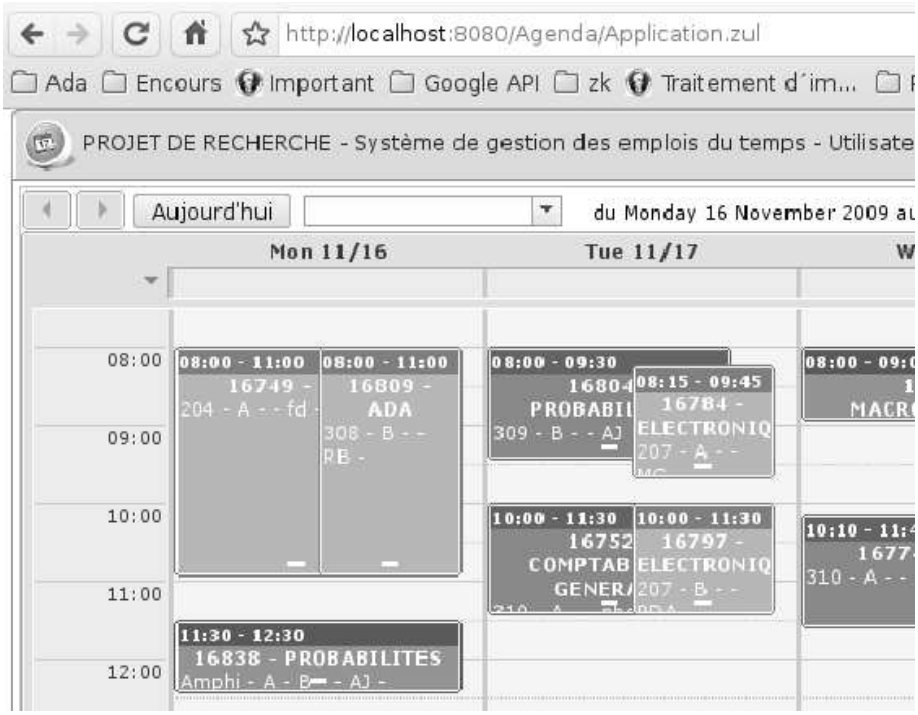
4 Distributed approach

4.1 Motivation for using web services

Creating a new application entails deploying it in an existing IT system. In this system, there is always a user identification process. This process can be used as a service. That was the first reason why we decided to use web services; on the other hand, we want to create an RIA, but we have to compute a timetable. To solve the timetabling problem, as mentioned above, a backtrack search algorithm is used. This algorithm was implemented in Prolog. This implies that the RIA, and the Prolog program have to interact. We choose to use the web services technology for doing this.



(a) A compute timetable



(b) A partial zoom of a timetable

Fig. 7 Timetable

In addition, some contributors may wish to use their own calendar. In our institution there is no shared calendar, but we plan to use Google Calendar soon, which

many contributors are already using.. So we decided to allow the contributors to use their own calendar. This implies that the contributors' constraints are generated in two different ways. Some can simply put their constraints on Google Calendar, the others may use our application once again; we need web services[7] to import and export some slots. Some others reasons to use web services will be explained later.

4.2 Export and import from Google Calendar

4.2.1 Google API

If there are two main components that access the database, 1, there is one which does not access the database. In our application, we have a component that can export from or import to a Google Calendar. This component does not directly interact with the database, but it can modify the database through the RIA. Google releases an API to use a lot of their components. This API allows our contributor to use their own calendar.

4.2.2 Constraints acquisition

A major feature of our application is the acquisition of the users' constraints. This can be done by the RIA, or by putting the constraints in the Google Calendar. The only restriction is to have a particular calendar, which is used to show unavailability. This calendar will be used later to input the compute time slot. The contributor has to put his constraints on Google calendar. He can use all of the Google features. These constraints will be validated by the *admin*. When the *admin* checks the users' constraints, he has the choice to validate or to invalidate it. Both of the choices will inform the user by putting a message in the appropriate time slot.

To perform this kind of operation, the contributor has to allow our application to modify his own calendar. Ideally, the contributor creates a particular calendar, and uses this one to manage his professional time slot.

4.2.3 Timetable visualisation

When we have full access to a contributor's calendar, we are able to create a full timetable. For instance, we have created a specific user. This user has several calendars, one for each room, one for each class, and possibly one for each contributor. This allows when needed, to visualize all timetables with all possible views. Of course, the compute timetable can only be viewed by classes, but we provide any possible view.

As the contributor has a Google calendar, his own timetable is created on it. This allows a convenient view for all who need it.

[7] As Google calendar is accessible via web services.

5 Database modeling for Timetabling

5.1 Required resources for timetabling

The main goal of timetable generation is to compute time slots from descriptive resources. Such resources are needed to provide the context model we want to use. For example, to compute the timetabling of lectures, we need to know all information dealing with lectures such as:

- who will follow this lecture
- who will teach this lecture
- how long is this lecture
- which material is needed

Less specifically, we can split such information in two distinct categories: inner and outer properties. Inner properties deal with information directly linked to lectures (duration, starting time, fixed time). Outer properties deal with information linked to external resources (people, materials, calendar). Therefore, the global data architecture is made of resources associated to time values. We can assume that time values associated to each resource contain an interval of availabilities. Then, the database should contain tables for each type of resource and at least a field for its availability. Therefore, to compute a time slot associated to a main resource, we then first have to collect all the associated availabilities that can be found in the records of the associated resources. Without loss of generality, we can assume that in this particular case, the values that we have to compute (time slots), can be represented by integers corresponding to an interval of times between 1 and N, where N depends on the number of time slots of the timetable. This value can change with the granularity of the timetabling, that is the difference between two consecutive time slots. With this modeling, we can transform our problem in a constraint satisfaction problem on finite domains (Apt and Zoeteweij 2007), what is detailed in section 7.

6 Database implementation

6.1 Motivation to use a central database

Once we have all constraints, we need to compute a timetable which complies with all of them. We need to save these constraints somewhere. This can be done by using a database, XML or anything that stores data. As the computation is done by Prolog, we have to provide an access to this data.

A central point to notice is that our application use the web services to communicate. This means that you design several small applications and assemble them to make a bigger one. The communication between each part of the application has to send or receive some data. Possibly we could use a data feed. It is really simple to send the data to the computational part, and it is also as simple to receive the result. But for providing the users with an interface, we have to keep the generated data. Also, we have to keep the users' entries. That means that we need something to save the data. The best solution for us was to use a database. Indicating this has changed the approach of our application. We choose to put the database at the center of the application.

Also, using a central database allows the RIA to modify data, while Prolog is computing a timetable. Why can this be done ? The database itself prevents multiple modification on the same data. Therefore once Prolog has gathered the data to compute the timetable, they are not missing anymore, while modifying another data. As the computation is done by invoking a web service, we can focus the computation on a particular week, and then begin to modify this week. In fact, the computation accesses the data to get *all* the constraints in one shot, then computes the timetable and then commits the result to the database. As a web service is an asynchronous method, we can run the computation, and leave it. As the same time you may modify the data you need. But if you want this new data to be considered, you have to rerun the process.

On the other side, we have the RIA that has to use the database. Since ZK framework is a java framework, we can use Hibernate tools to do the mapping from database to objects. By using this technology, we can save time(Minter and Linwood 2006) when creating the RIA. It also provides us with a way to automatically create an RIA for a CSP.

6.2 Shared Database for asynchronous communication

Currently, our application is made up of several *black boxes*. Each of these has to access and modify the database. For now, there are two main components which use the database : the RIA, and the computational program. The benefits of doing this is that we can keep running the service at all times. There is no need to run all parts of our application for it to work. You can use only the RIA to update the data, or use only the computed part to create a timetable.

We use a web service to order the computational part to run. Whilst the web service allows us to mix Java and Prolog in one application, it has some disadvantages too. Among them, there is the timeout problem. This problem can occur at various times, depending on the system, the network . . . Again using a central database helps us to resolve this problem. Thus, we communicate via the database, when the RIA asks to the computational part, this simply returns an acknowledgment message. Then, later the RIA will check the database to see if the computational program has finished computing the timetable. This approach has several advantages. One of them is that we can use the RIA without waiting for the end of the computational timetable. Another one is that the computational program can put some statistics in the database . The RIA just has to query the database. Then we can create a report of the timetabling.

To simplify the work both of the component use an ODBC to interact with the database. For Prolog we use a classical ODBC, and for ZK we use hibernate. Using the ODBC allows us to change quite dynamically the database server. It also provides us with a convenient way to access to the data. We do not have to consider the multiple access. This job is done by the ODBC. We can have concurrent access, the processing is solved by the database itself.

In this section, we now explain how we can make the design part of timetabling using CSP on finite domain. We first define CSP and then formulate timetabling problem in this context. Finally we give an example of the use of our model with an instantiation in real case of university timetabling from which we explain some computational results.

7 CSP on finite domain

7.1 Definitions

A constraint satisfaction problem (CSP) is modeled with a triple $P = (X, D, C)$ where $X = \{X_1, \cdots, X_n\}$ is a finite set of variables to assign, D is the domain function of each variable, that is $(D(X_i))$ contains all the possibles values of variables. From this point forward, we will consider to finite domains. The last element of the triple P is a finite set of constraints $C = \{C_1, \cdots, C_m\}$. A constraint is a relation between sub-sets of variables $W \subseteq X$, and a sub-set of values $T \subseteq D^W$. A mapping is a set of pairs (variable-value): $A = \{(X_j \leftarrow v_j)\}$ with $X_j \in X$ and $v_j \in D(X_j)$. A mapping is total if for each variable there is a value assigned. It is valid with C if every relation of C_i is true for all variables in A. A solution to a CSP is a total and valid mapping.

7.2 Modeling of Timetabling in CSP

With this data description and computational model, we can describe the timetabling problem in CSP as follow:

- $X = \{X_1, \cdots, X_n\}$ is the finite set of time slots we have to assign to entities. X_j is the starting time of j^{th} entity. The number of entities (n) is obtained from the database.
- $D(X_i)$ is the interval of possible values for the i^{th} entity. It depends on the availability of the entity, that is also obtained from the database.
- $C = \{C_1, \cdots, C_m\}$ is the set of all constraints on variables and can be split to the following categories :
  - scattering constraint checks that among all pairs of distinct variables from a subset $S \in X$, there is a break of at least l_d:
    $scattering(S, l_d) \equiv \forall X_i \forall X_j \ ((X_i \in S) \wedge (X_j \in S) \wedge (X_i \neq X_j)) \Rightarrow ((X_j > (X_i + d_i + l_d)) \vee (X_i > (X_j + d_j + l_d)))$
    where d_i is the duration of the i^{th} entity
  - overlapping constraint checks that among all pairs of distinct variables from a subset $S \in X$, there is an overlap if we want a break of at least l_d:
    $overlapping(S, l_d) \equiv \forall X_i \forall X_j \ ((X_i \in S) \wedge (X_j \in S) \wedge (X_i \neq X_j)) \Rightarrow ((X_i < X_j < (X_i + d_i + l_d)) \vee (X_j < X_i < (X_j + d_j + l_d)))$
  - relation constraint checks that among variables of all couples from a subset $P \in X^2$, there is a relation r:
    $relation(P, r) \equiv \forall X_i \forall X_j ((X_i, X_j) \in P) \Rightarrow r(X_i, X_j)$
  - separation constraint checks that around a specific value,v_b, there is a break of at least length l_b between all pairs of variables from a subset $S \in X$:
    $separation(S, v_b, l_b) \equiv \forall X_i \forall X_j \ ((X_i \in S) \wedge (X_j \in S) \wedge (X_i < v_b) \wedge (X_j > v_b)) \Rightarrow (X_j > (X_i + d_i + l_b))$.

Therefore computing time slots consists of four consecutive steps:

1. Requests:
   Extraction from the database of all the different entities we need to apply constraints and their availabilities. We can build the X part of the CSP modeling.

2. Domains:
   Affectation of interval values to the variables from X, that is the D part of the CSP modeling.
3. Constraints:
   Application of the different constraints to each group of entities. We build the C part of the CSP modeling for timetabling.
4. Computation:
   Computing times slots for each entity, as a solution of the obtained CSP problem in the two previous step.

8 Prolog implementation

Our application is built upon a SOA architecture. The CSP part is implemented in swi-prolog and communicates with the RIA part using Web services and with the database using classic ODBC. We use the *clpfd* library of swi-prolog to implement the CSP. This library deals with finite domain, thus we can compute time slots as consecutive integers on the interval $\{i_1 \cdots, i_n\}$, where i_1 and i_n are the first and last available time slots of the timetabling period.

We can express all the constraints in this library using arithmetic or domain constraints and then apply the "labeling" predicate to obtain a solution to the CSP. This implementation follow the classical model of constraint programming (Jaffar and Maher 1994)(Frühwirth and Abdennadher 2003)

9 Model instantiation

9.1 A concrete example for lectures timetabling

Finally, we end this section with an application of our model to the generation of lectures in a university U during a week S. The resources are the following:

- lectures and their properties (duration, type of classroom required, week, teachers, groups of students),
- teachers and their availabilities,
- groups of students and their size,
- type of classrooms including their size and equipment.

9.2 Requests

In the first step of the computation, in this example, the resources we need to collect to apply constraints to, are the following:

- general parameters of the system:
  - list of time slots: $lts(U, S)$
  - list of students groups: $lg(U, S)$
  - list of teachers: $lt(U, S)$
  - list of type of classrooms: $lc(U, S)$
  - list of classrooms of a specific type C: $lct(C)$

- lectures lists:
  - list of lectures for a specific students group G: $llg(G)$
  - for a given group G, the list of pairs of lectures such that first one has to be given before the second one: $lpl(G)$
  - list of lectures for a specific teacher T: $llt(T)$
  - list of lectures with the specific classroom type C: $llc(C)$
- availabilities list:
  - list of availabilities for a specific teacher T: $lat(T)$
  - list of availabilities for a specific students group G: $lag(G)$

9.3 Domains

In the second step, we assigned interval values to all variables (lectures):

- lectures are assigned to time slots:
  $\forall G(G \in lg(U,S)) \Rightarrow (llg(G) \in lts(U,S))$
- teacher availabilities:
  $\forall T(T \in lt(U,S)) \Rightarrow (llt(T) \in lat(T))$
- group availabilities:
  $\forall G(G \in lg(U,S)) \Rightarrow (llg(G) \in lag(G))$

9.4 Constraints

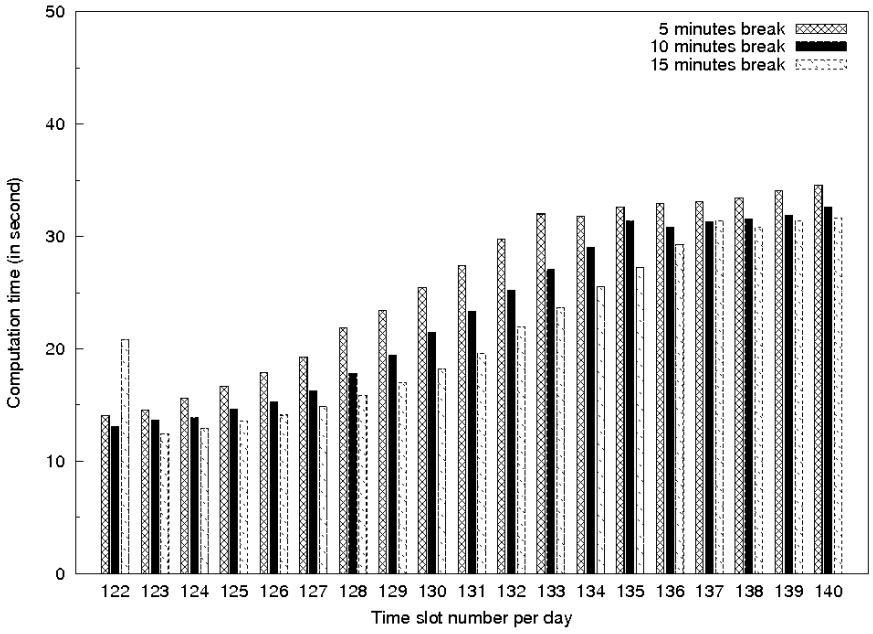
The third step of the computation is now an instantiation of constraints schemes 7.2:

- lesson lectures are always take place before practice lectures for a specific group:
  $\forall G \in lg(U,S), relation(lpl(G), <)$
- lunch break of at least duration d around time t for a specific group:
  $\forall G \in lg(U,S), separation(llg(G), d, t)$
- break of at least duration d between two lectures of a specific group:
  $\forall G \in lg(U,S), scattering(llg(G), d)$
- break of at least duration d between two lectures of a specific teacher:
  $\forall T \in lt(U,S), scattering(llt(T), d)$
- the number of overlapped lectures with classroom type C is less or equal than the number of classroom of type C:
  $\forall C, \forall T, ((C \in lc(U,S)) \wedge (T \subseteq llc(C)) \wedge overlapping(T, l_b)) \Rightarrow (|T| \leq |lct(C)|)$

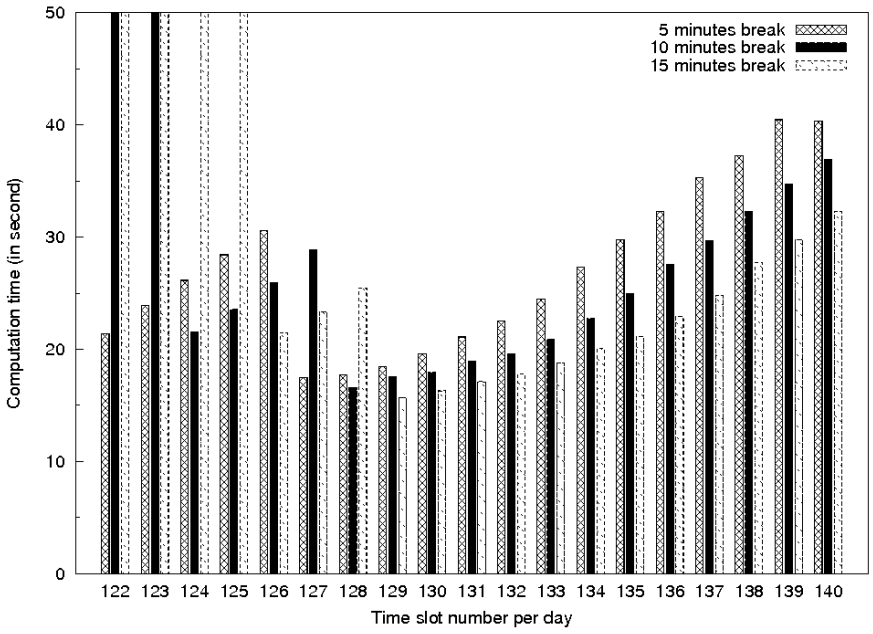
10 Results

In this part we present results on computation time of the CSP part of the application with different values for the following parameters:

- time slot number per day (from 120 to 140)
- break between two consecutive lectures (5, 10 or 15 minutes)
- lunch break (90 or 120 minutes)



(a) Lunch break: 90 minutes



(b) Lunch break: 120 minutes

Fig. 8 Computation times

The computation was done for 3 different classes ($C1$, $C2$ and $C3$). $C1$ was made of two subgroups ($C1_a$, $C1_b$), as well as C3 ($C3_a$, $C3_b$). $C1$ consumes 590 time slots. Among them 36 was shared between the two subgroups. $C1_a$ and $C1_b$ got 282 different time slots. $C2$ only consumes 372. Finally $C3$ had 84 shared time slots, and each subgroup was using 192.

There were only one global constraint : "Thursday afternoon is devoted to sport, so we do not want to have courses on this particular period".

The computation was launched once. In figure 8 the 50 seconds value, means that the algorithm was not able to find an answer in the accorded time. It does not means obligatory that there is no solution. Running more than once the computation does not change the look of the results neither the interpretation that we can make on it.

There are some interesting things to notice :

- In the general case, 5 minutes break is longer to compute than 15 minutes break. This can be explained by the fact that the definition domain is shorter for 15 minutes break than for 5 minutes break. This means that there is less case to test. You can find this case in figure 8(a) from 123 slots per day, and in figure 8(b) from 129 slots.
- In the specific case, we can find a solution for 5 minutes break, but not for 15 minutes break. This means that if you want a solution you have to less restrain the domain. You can find this case in figure 8(b) before 126.
- If we are not in the two previous case, the results may be inverted. We can explain this. There is some solution, but they are more frequent with 5 minutes break, than with 15. So the algorithm is able to find one quicker for 5 than for 15 minutes break.

11 Conclusion

In this paper we have presented a modern approach to timetabling, mixing the advantages of RIA for data acquisition and the power of constraint programming to find a solution. To validate this approach, we have created an application for university timetabling from which we have computed results with some parameters variations. The obtained application validate the choices we have made. This software is a very useful tool to design timetable, as expected. But it also provide a tool to analyse the complexity variation of timetabling under realistic data sets. In future works, we plan to design other type of timetabling with our approach and to improve the result interpretation with many more data, that is now possible thanks to our global architecture.

References

Abbas A, Tsang E (2001) Constraint-based timetabling-a case study. Computer Systems and Applications, ACS/IEEE International Conference on

Abdennadher S, Aly M, Edward M (2007) Constraint-based timetabling system for the german university in cairo. In: INAP/WLP, pp 69–81

Apt KR, Zoeteweij P (2007) An analysis of arithmetic constraints on integer intervals. Constraints 12(4):429–468

Frühwirth T, Abdennadher S (2003) Essentials of Constraint Programming. Springer Verlag

Jaffar J, Maher MJ (1994) Constraint logic programming: A survey. J Log Program 19/20:503–581

Lehtinen J (2009) Ria security. In: JAZOON09
Minter D, Linwood J (2006) Beginning Hibernate: From Novice to Professional. Apress
Qu R, Burke EK, Mccollum B, Merlot L, Lee SY (2009) A survey of search methodologies and automated system development for examination timetabling. J of Scheduling 12:55–89
Seiler D (2009) Ria with zk. In: JAZOON09
Wallace M (1996) Practical applications of constraint programming. CONSTRAINTS 1:139–168
Yeh TM (2006) Zk ajax but non javascript
Yeh TM (2007) Server-centric ajax and mobile

Soccer Tournament Scheduling Using Constraint Programming

Mike DiNunzio · Serge Kruk

Abstract Larger soccer tournaments of school-age children in Michigan can host as many as 600 teams divided into three age groups, all of which must play a set number of games, on a fixed number of fields, during a weekend. This leads to three scheduling problems of roughly 200 teams that must be done concurrently. With side constraints involving resting time, limited number of games in a day and playoffs, the task soon becomes unruly if done by hand. This paper summarizes the results of efforts to develop a Constraint Programming solution to this Soccer Tournament Scheduling Problem, and concludes that an appropriate model, combined with an appropriate search strategy, can handle problems of practical size.

Keywords Tournament · Constraint Prohramming

1 Introduction

1.1 Background

Based on one of the authors' multi-year experience, scheduling soccer tournaments is an exercise typically done with pencil and paper. As a result, there is an inherent unfairness in the schedules that are computed, as well as certain inefficiencies (empty fields, long hours). Overflow games, or games that do not fit neatly into the Saturday/Sunday preliminary schedule, are usually scheduled on the previous Friday night, bringing players, referees, and tournament officials to the tournament site for an additional evening. While commercial sports tournament scheduling software is available, most is of the drag and drop variety (not functionally different from using pencil and paper), or does not address the issues and requirements that are particular to soccer tournaments.

M. DiNunzio
Oakland University
MI, USA
E-mail: mrdinunz@oakland.edu

S. Kruk
Oakland University
MI, USA
E-mail: kruk@oakland.edu

This paper examines a practical solution to these issues through the use of constraint programming. A number of different approaches were considered, some of which did not work at all, and others which worked moderately well (scheduling 10-50 teams). Ultimately, tuning the search strategy, a solution technique was reached which works well with over 200 teams.

1.2 How Soccer Tournaments are Organized

The first step in scheduling a soccer tournament is organizing the divisions, small groups of three to five teams of comparable ability and skills. This aggregation is done by the coaches to offer to the players a reasonable yet challenging experience. Then divisions are paired to offer intra and inter-divisional games. Usually three preliminary games are played, followed by a playoff game where the winners of divisions face off against each other. Divisions of varying sizes are scheduled in the following manner:

(3 Teams) Each team plays the other two teams in their division, plus a team from a 2^{nd} division. A championship playoff game is scheduled pitting the winners of the two divisions.
(4 Teams) Each team plays the other 3 teams within their division. The winner of the division plays the winner of another division in a playoff game.
(5 Teams) Each team plays the other 4 teams within their division. A playoff game may or may not be scheduled.

The ability to schedule divisions of 3,4, or 5 teams gives the tournament organizers a great deal of flexibility in setting up the divisions and handling last minute team additions. A typical tournament has preliminary games all day Saturday, and the 1st half of Sunday. The 2^{nd} half of Sunday is set aside for the playoff games. If necessary, games can be played (local teams only) on the Friday night prior to the start of the tournament.

Soccer fields are one of three sizes; 6v6, 8v8, and 11v11 referring to the number of players on a side. Younger players, up to about age 9, play on the 6v6 fields. Players from about the age of 9 through 11 play on the 8v8 fields. Older players use the 11v11 fields. There is no mixing of the three groups. Scheduling a soccer tournament then becomes three completely separate problems, one for each of the three field sizes.

Based on the experience of one author, the following issues stand to be improved through better soccer tournament scheduling:

1. Games are often widely spread throughout the day. It is not uncommon for a team to play a game at 7 am, then have to return to play again at 7 pm. Most families would prefer to play at 7 am and 11 am, for example, freeing up the rest of the day to do something else. We would like to cap the length of time a team has to wait between games.
2. Fields are not always efficiently used; there are often empty fields throughout the day. Is it possible to compress the schedule for increased efficiency?
3. Related to 2, is that overflow and hard-to-schedule games are put into a Friday night time slot, prior to the official start of the tournament. This uses tournament resources as well as disrupting families Friday nights. Can we eliminate the need for Friday night games?

4. Scheduling a tournament is a drawn out process, generally taking 2-3 weeks from the time registration closes to several days before the start of the tournament. Can we reduce that 2-3 week time period to under an hour even when scheduling concurrently the three age groups?
5. When scheduling playoff games, we would like to begin playoff games for a particular pair of divisions before the end of the preliminary schedule, provided the slots are open and there is no chance of any preliminary games interfering with the playoffs for that division.
6. A large soccer tournament can have 200 or more teams to be scheduled for each of the three field sizes. Our solution has to be capable of handling that many teams.

1.3 Constraints to be Implemented

From the above observations, we formulate the problem. Our goal is to schedule all preliminary games and playoff games, given a number of teams, divisions, playing fields, and time slots. The following hard and soft constraints are under consideration.

1. No team plays more than 2 preliminary games in a day.
2. Each team plays the prescribed number of games against division and cross-division opponents.
3. Each team needs a rest period between games.
4. There is a maximum time gap between games, so that families do not need to spend all day at the field.
5. A playoff game can only be scheduled after the last scheduled preliminary game for the divisions involved, with an appropriate rest period between the games.
6. Each fields use is maximized so that Friday night games do not need to be scheduled (soft constraint).
7. Use the minimum number of time slots to fill the schedule, possibly allowing teams to finish earlier and/or start later, thereby increasing goodwill (soft constraint).

1.4 Objectives

The soft constraints could be seen as an objective function: minimizing the total number of time slots so that we do not schedule on Friday night. Note that minimizing more than that is neither useful not required. A second additional objective (assuming that the 'no Friday games' is achieved) is to minimize the time between the first and last game of each team in a day. Minimizing idle time provides the player and their families a better experience.

1.5 Previous work in Sports Scheduling

The definitive annotated bibliography of sport scheduling is [3]. Much research has been done in the area of starting with the pioneering work of de Werra [12, 11, 10]. Much of the focus has been on round-robin or double round-robin tournaments, where a team plays once home and once away, [1, 6], or some other variation where a certain balance between home and away is the goal [2]. Sometimes, there may be a limit on one or the

other tournament [1,8], possibly based on the strength of the team or the geographical distance. Some other restrictions are based on the sharing of home facilities between teams [13].

Much of this research has little connections with our problem since our teams are all converging on a single venues, with multiple fields, for a weekend of games. More closely related is the work of Schaerf [8] on a general Constraint Programming approach. Our secondary objective of minimizing the time between the first and last game of a day is related to the break minimization problem [9,7,4]. Our sharing of fields among all teams is related to the work of [5] though they used a simulated annealing algorithm, and the work of [2].

In addition to being a problem combining temporal as well as spacial constraints, we are looking at a very large number of teams. A large scale soccer tournament for one age group (say 200 teams) can be thought of as perhaps 30 small round robin tournaments, each of which must be scheduled around each other, and fit into the confines of a single weekend. The challenge, then, is to efficiently schedule each team for the desired number of games while maintaining constraints discussed earlier. By efficiently, we mean that, as late as on the Friday or even Saturday morning, it must be possible to add teams and produce an amended schedule in a few seconds or minutes.

2 Moving Toward a Solution

It was decided early on to use constraint programming to work toward a solution. ECLiPSe (`http://eclipse-clp.org/`) was chosen because it easily allows experimentation and prototyping. Once the implementation proved correct, we could always, if needed, re-implement using a faster library. In implementing, these steps were followed:

1. Choose an appropriate model, using only integer values.
2. Input the parameters: number of playing fields, time slots, start time, etc, in a form that is compatible with the data structure and the organizers' experience.
3. Apply as many constraints as possible to limit the search, prior to assigning the variables.
4. Label for the variables, paying special attention to the order in which these variables are chosen.
5. After scheduling the preliminary games, schedule playoff games using time/field slots not used during the preliminary games.
6. Manipulate the results as necessary to present them in a format meaningful to the organizers.

2.1 Failures

2.1.1 The Multi-Dimensional Array

One strategy that was unsuccessful was to simulate a large multi-dimensional array, with a separate dimension for: team number, opponent team number, field number, time, and day. If all the variables coincided, a 1 value would signify that the game was on. Otherwise, the value of the variable was assigned 0.

This solution worked, but not for more than 10 teams. The search space was simply too large (or pruning was too ineffective) to come up with a solution in a reasonable

time. To use this technique with 50 teams, and 15 fields, with 10 time slots over each of 2 days, would require a total of 750,000 variables! And, the objective was to be able to schedule 200+ teams, which would have involved a considerably larger search space.

2.1.2 The 2-Dimensional Array

The second approach involved simulating a 2-dimensional array, each dimension with N entries, where N is the total number of Teams. Each variable in the array would contain either a 0, or, if a game was to be played between those two teams, a game number greater than 0. The game number was encoded to contain both time slot and field information:

$$\lfloor (G-1)\ N \rfloor + 1 = T$$
$$G - (T-1) \times N = F$$

where G is the game number, N is the number of fields, T is the time slot and F is the field number of the game. For example, in the matrix

	Team1	Team 2	Team 3	Team 4
Team 1	0	3	117	230
Team 2	3	0	238	239
Team 3	117	238	0	115
Team 4	15	239	115	0

Teams 1 and 4 will meet on game number 230. Since we have 15 playing fields available and $\lfloor (230-1)/(15) \rfloor + 1 = 16$, the game will be played on time slot 16. Consequently, $230 - (16-1) * N = 5$ which means it will be played on field 5.

A strictly triangular matrix could also be used . This approach also worked, but not for the required number of teams. It was capable of scheduling a tournament with about 50 teams in what we considered a reasonable time.

2.2 Success

It was decided that limiting the overall number of variables might yield better results, so the following model was chosen. The Game Number encoding approach was kept but the matrix was discarded in favour of lists. Note that every variable holds meaningful data (none take the value 0). We introduced two lists of lists of the same size. One for opponent and one for game. For example, the same schedule between teams $1, 2, 3$ and 4 would be encoded as

```
OpponentList = [[2, 3, 4], [1, 3, 4], [1, 2, 4], [1, 2, 3], ...]
GameList = [[3, 117, 237], [3, 238, 239], [117, 238, 115], [15, ...],...]
```

The lists GameList and OpponentList contain all the coded information necessary to represent a game Both Team Numbers, Time, Day, and Field Number. This model was the one which ultimately provided a solution meeting the objectives stated earlier in this paper. Note that there is still some redundancy that could be eliminated but this structure proved to be small enough not to cause memory problems. It could be allocated from the start, as we know exactly how many preliminary games each team

will play and the symmetry constraint (team 1 vs team 2 is identical to team 2 vs team 1) is easy to impose.

The playoff games were scheduled after all preliminaries games were scheduled. The overall structure of the approach was:

```
solve(GameList,OpponentList):-
initialize(GameList,OpponentList,NumGames),
constrain(GameList,OpponentList),
  labeling(OpponentList),!,
ourlabeling(GameList),
outputSchedule(OpponentList,GameList),
schedulePlayoffs(GameList,PlayoffGameList),
outputPlayoffs(PlayoffGameList).
```

After the appropriate model and the encoding that allowed us to have variable domains representing with one integer both the spatial component (the field) and the temporal component (the time slot), the key to success was the search strategy, i.e. the labeling variable order and the value order. Specifically,

- We label the Opponent List first. When backtracking occurs in an effort to find a solution, there is no need to swap opponents around. If 1 is scheduled to play 2, then swapping 2 for 3,is most likely not getting any closer to a solution. So we label the Opponent List first, which is easy, and then focus primarily on getting the Game List right.
- Labeling the games in order by team caused problems: too many backtracks. Instead, the variable ordering was to choose the team that has the most games to be assigned. Followed by the possible opponent that has the most games to be assigned. Assign a game number to those two teams at that point.
- The game is chosen to minimize the time to the last game assigned to the team, while maintaining the appropriate rest period. This is done by pruning domains of the games not yet assigned immediately after a game assignment, an easy constraint given our encoding.
- Note that there is no need for one labeling on time slots and one on fields, with possible ensuing conflicts. The encoding allowed, in a sense, both labellings at the same time. This was also crucial for the success of the approach.

2.2.1 Objective function

Recall that we had two objectives: first to eliminate, if possible Friday games; second to minimize the spread between the first and last game of a team. We aim at this multiple objective by minimizing the time between first and last game of each team, constraining time slots to avoid Fridays. If the time spent backtracking is too large, we restart the search allowing Friday time slots. This is clearly heuristic and it is possible to miss an optimal solution with small game spread and no Friday games. But on the instances we tried, Friday games were never used.

2.3 Efficiency of the Solution

Actual tournament data was run through the application. We had available the data for two tournaments. From these, we synthesised variations, by deleting playing fields,

deleting time slots, adding teams here and there more or less haphazardly to stretch the implementation and test its robustness.

For the largest real tournament instance, comprised of 227 teams in the 11v11 bracket, the manually produced schedule used 23 fields, Friday evening games were scheduled and some teams had to wait up to 10 hours between weekend games in a given day. We now comment on the automated solution achieved for this instance and whether the objectives were met.

1. Our ECLiPSe implementation was able to come up with a solution where the maximum time between games was 4 time slots. This is a considerable improvement over the 8,9, or 10 time slots that teams commonly have to contend with.
2. In the largest instance solved, 343 games were scheduled across 19 time slots using 19 fields (361 possible events). A clear improvement on the 23 fields required by the manual solution. Theoretically, one needs $\lfloor 343/19 \rfloor = 19$ fields. Therefore 19 fields is the absolute minimum that can be used. After scheduling the preliminary games field utilization was just over 95%. In addition, due to the value ordering of our encoding, most of the unused slots were at the end of the preliminary game schedule and were used later to schedule playoff games.
3. Playoff scheduling was efficient and made optimal use of the remaining time slots. After filling the unused preliminary slots with playoff games, only 1 slot could not be used, for an efficiency of 360/361 or over 99%.
4. It was not necessary to schedule any games on Friday evening.
5. Execution time was below 10 seconds on an average PC.
6. One unintended benefit is that each age group, since they were assigned team numbers close to each other, tended to play similar schedules. So a coach of several teams across different age groups has an excellent chance of attending all the games. It is now conceivable, through applying additional constraints, to ensure this additional constraint because the current solution technique is fast enough.

3 Conclusion

We defined a scheduling problem modeling school-age children soccer tournaments in Michigan. We suspect the problem is almost identical elsewhere in the country. The problem has temporal and spatial constraints, both soft and hard, and is fairly large scale, as far as sport schedules are concerned. The first key element of the implementation is an encoding that allows scheduling time and place of a game with one instantiation, allows efficient pruning of variable domains for hard constraints and also allows simple minimization of violation of soft constraints. The second key element is a search based on a dynamic choice of the variables ordered according to the number of games yet to be scheduled.

Experiments with real data from two tournaments shows that our implementation in ECLiPSe, can do an excellent job of scheduling large scale soccer tournaments; much better, for the instances tested, than the manual schedules that were used. We are now in a position to add a number of secondary constraints and further improve the tournament experience for all participants, players, coaches, families and friends.

References

1. Easton, K., Nemhauser, G., Trick, M.: CP based branch-and-price. In: Constraint and integer programming, *Oper. Res./Comput. Sci. Interfaces Ser.*, vol. 27, pp. 207–231. Kluwer Acad. Publ., Boston, MA (2004)
2. Hamiez, J.P., Hao, J.K.: Using solution properties within an enumerative search to solve a sports league scheduling problem. Discrete Appl. Math. **156**(10), 1683–1693 (2008). DOI 10.1016/j.dam.2007.08.019. URL http://dx.doi.org/10.1016/j.dam.2007.08.019
3. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: an annotated bibliography. Comput. Oper. Res. **37**(1), 1–19 (2010). DOI 10.1016/j.cor.2009.05.013. URL http://dx.doi.org/10.1016/j.cor.2009.05.013
4. Knust, S.: Scheduling sports tournaments on a single court minimizing waiting times. Oper. Res. Lett. **36**(4), 471–476 (2008). DOI 10.1016/j.orl.2007.11.006. URL http://dx.doi.org/10.1016/j.orl.2007.11.006
5. Lim, A., Rodrigues, B., Zhang, X.: Scheduling sports competitions at multiple venues—revisited. European J. Oper. Res. **175**(1), 171–186 (2006). DOI 10.1016/j.ejor.2005.03.029. URL http://dx.doi.org/10.1016/j.ejor.2005.03.029
6. Nemhauser, G., Trick, M.: Scheduling a major college basketball conference. Operations Research **46**, 1–8 (1997)
7. Régin, J.C.: Minimization of the number of breaks in sports scheduling problems using constraint programming. In: Constraint programming and large scale discrete optimization (Piscataway, NJ, 1998), *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, vol. 57, pp. 115–130. Amer. Math. Soc., Providence, RI (2001)
8. Schaerf, A.: Scheduling sport tournaments using constraint logic programming. Constraints **4**(1), 43–65 (1999). DOI 10.1023/A:1009845710839. URL http://dx.doi.org/10.1023/A:1009845710839
9. Suzuka, A., Miyashiro, R., Yoshise, A., Matsui, T.: The home-away assignment problems and break minimization/maximization problems in sports scheduling. Pac. J. Optim. **3**(1), 113–133 (2007)
10. de Werra, D.: Scheduling in sports. In: Studies on graphs and discrete programming (Brussels, 1979), *Ann. Discrete Math.*, vol. 11, pp. 381–395. North-Holland, Amsterdam (1981)
11. de Werra, D.: On the multiplication of divisions: the use of graphs for sports scheduling. Networks **15**(1), 125–136 (1985). DOI 10.1002/net.3230150110. URL http://dx.doi.org/10.1002/net.3230150110
12. de Werra, D.: Some models of graphs for scheduling sports competitions. Discrete Appl. Math. **21**(1), 47–65 (1988). DOI 10.1016/0166-218X(88)90033-9. URL http://dx.doi.org/10.1016/0166-218X(88)90033-9
13. de Werra, D., Jacot-Descombes, L., Masson, P.: A constrained sports scheduling problem. Discrete Appl. Math. **26**(1), 41–49 (1990). DOI 10.1016/0166-218X(90)90019-9. URL http://dx.doi.org/10.1016/0166-218X(90)90019-9

Truck Driver Scheduling and Australian Heavy Vehicle Driver Fatigue Law

Asvin Goel

Abstract In September 2008 new regulations for managing heavy vehicle driver fatigue entered into force in Australia. According to the new regulations there is a chain of responsibility ranging from drivers to dispatchers and shippers. Thus, carriers must explicitly consider driving and working hour regulations when generating truck driver schedules. This paper presents various heuristics for scheduling driving and working hours of Australian truck drivers.

Keywords Vehicle Scheduling · Working Hour Regulations

1 Introduction

According to a survey of truck drivers in Australia, fatigue is felt as contributing factor in every fifth accident (Williamson et al. [2001]). One out of five drivers reported at least one fatigue related incident on their last trip and one out of three drivers reported breaking road rules on at least half of their trips. Many drivers feel that fatigue is a substantial problem for the industry and feel that their companies should ease unreasonably tight schedules and should allow more time for breaks and rests during their trips. In their efforts to increase road safety the Australian Transport Ministers adopted new regulations for managing heavy vehicle driver fatigue. According to the new regulations there is a chain of responsibility ranging from drivers to dispatchers and shippers. Consequently, road transport companies must ensure that truck driver schedules comply with Australian Heavy Vehicle Driver Fatigue Law. An important key in managing fatigue is to explicitly consider driving and working hour regulations when generating truck driver schedules. Planning problems considering driving and working hours of truck drivers, however, have so far attracted very little interest in

Asvin Goel
MIT-Zaragoza International Logistics Program
Zaragoza Logistics Center, Spain
E-mail: asvin@mit.edu
and
Applied Telematics/e-Business Group,
Department of Computer Science, University of Leipzig, Germany
E-mail: asvin.goel@uni-leipzig.de

the vehicle routing and scheduling literature and to the best of the author's knowledge there are currently no planning tools available that allow for truck driver scheduling considering Australian Heavy Vehicle Driver Fatigue Law.

Driver scheduling in road freight transportation differs significantly from airline crew scheduling and driver scheduling in rail transport or mass transit systems which are covered by a comprehensive annotated bibliography by Ernst et al. [2004]. The difference stems from the fact that in road freight transportation it is usually possible to interrupt transportation services in order to take compulsory breaks and rest periods. Furthermore, time constraints in road freight transport are usually not as strict and departure and arrival times can often be scheduled with some degree of freedom.

The first work known to the author explicitly considering government regulations in vehicle routing and scheduling is the work by Xu et al. [2003] who study a rich pickup and delivery problem with multiple time windows and restrictions on drivers' working hours imposed by the U.S. Department of Transport. Xu et al. [2003] conjecture that the problem of finding a feasible schedule complying with U.S. hours of service regulations is NP-hard in the presence of multiple time windows. Archetti and Savelsbergh [2009] show that if weekly rest periods do not need to be considered and all locations shall be visited within single time windows, schedules complying with U.S. hours of service regulations can be determined in polynomial time. U.S. hours of service regulations differ significantly from Australian Heavy Vehicle Driver Fatigue Law because they do not demand short break periods for recuperation. Such short break periods are also included in European legislation which is studied by Goel [2009], Kok et al. [2009], and Goel [2010]. Goel [2009] presents a *Naive* and a *Multi-Label* scheduling method embedded to a Large Neighbourhood Search meta-heuristic for combined vehicle routing and scheduling. Kok et al. [2009] present a truck driver scheduling method extending the Naive method which considers additional provisions of the regulation which are ignored in Goel [2009]. Goel [2010] presents the first approach for scheduling driving and working hours of European truck drivers which is guaranteed to find a feasible truck driver schedule if such a schedule exists. This paper studies the Australian Heavy Vehicle Driver Fatigue Law, which, to the best of the authors knowledge, has yet not been tackled in the scheduling literature.

The remainder of this paper is organised as follows. Section 2 describes the Australian Heavy Vehicle Driver Fatigue Law. Section 3 presents the Australian Truck Driver Scheduling Problem (AUS-TDSP). In Section 4 some structural properties of the AUS-TDSP are given and solution approaches are presented in Section 5. Computation experiments are reported in Section 6.

2 Australian Heavy Vehicle Driver Fatigue Law

In Australia new regulations for managing heavy vehicle driver fatigue entered into force on September 29, 2008. The new regulations comprise three different sets of rules. Operators accredited in the National Heavy Vehicle Accreditation Scheme may operate according to the Basic Fatigue Management Standard (National Transport Commission [2008c]) or the Advanced Fatigue Management Standard (National Transport Commission [2008b]). One condition for being accredited is that operators must plan schedules and rosters to ensure they comply with the respective operating limits. Without accreditation operators must comply with the Standard Hours option

(National Transport Commission [2008a]) which imposes the following constraints on drivers' wschedules:

1. In any period of $5\frac{1}{2}$ hours a driver must not work for more than $5\frac{1}{4}$ hours and must have at least 15 continuous minutes of rest time
2. In any period of 8 hours a driver must not work for more than $7\frac{1}{2}$ hours and must have at least 30 minutes rest time in blocks of not less than 15 continuous minutes
3. In any period of 11 hours a driver must not work for more than 10 hours and must have at least 60 minutes rest time in blocks of not less than 15 continuous minutes
4. In any period of 24 hours a driver must not work for more than 12 hours and must have at least 7 continuous hours of stationary rest time
5. In any period of 168 hours (7 days) a driver must not work for more than 72 hours and must have at least 24 continuous hours of stationary rest time
6. In any period of 336 hours (14 days) a driver must not work for more than 144 hours and must have at least 4 night rest breaks (2 of which must be taken on consecutive days)

In the last provision a "night rest break" means a rest break consisting of (a) 7 continuous hours of stationary rest time taken between 10 PM and 8 AM on the following day; or (b) 24 continuous hours of stationary rest time.

If truck drivers do not work on Saturdays and Sundays, the last two provisions of the regulation are automatically satisfied. For simplicity, we will assume in the remainder that we are only interested in generating schedule for a planning horizon starting on Monday and ending on Friday of the same week.

3 The Truck Driver Scheduling Problem

This section gives describes the Australian Truck Driver Scheduling Problem for a planning horizon starting on Monday and ending on Friday of the same week. Let us consider a sequence of locations denoted by $n_1, n_2, \ldots, n_\lambda$ which shall be visited by a truck driver. At each location n_μ some stationary work of duration w_μ shall be conducted. This work shall begin within a time window denoted by T_μ. We assume that n_1 corresponds to the driver's current location and that the driver completes her or his work week after finishing work at location n_λ. The (positive) driving time required for moving from node n_μ to node $n_{\mu+1}$ shall be denoted by $\delta_{\mu,\mu+1}$. Let us assume that all values representing driving or working times are a multiple of 15 minutes.

In order to give a formal model of the problem, let us denote with `DRIVE` any period during which the driver is driving, with `WORK` any period of working time in which the driver is not driving (e.g. time in which the driver is loading or unloading the vehicle), with `REST` any period in which the driver is neither working nor driving. A truck driver schedule can be specified by a sequence of activities to be performed by the drivers. Let $\mathcal{A} := \left\{a = (a^{\text{type}}, a^{\text{length}}) \mid a^{\text{type}} \in \{\texttt{DRIVE}, \texttt{WORK}, \texttt{REST}\}, a^{\text{length}} > 0\right\}$ denote the set of driver activities to be scheduled. Let $\ll . \gg$ be an operator that concatenates different activities. Thus, $a_1.a_2. \ \ldots \ .a_k$ denotes a schedule in which for each $i \in \{1, 2, \ldots, k-1\}$ activity a_{i+1} is performed immediately after activity a_i. During concatenation the operator merges consecutive driving and rest periods. That is, for a given schedule $s := a_1.a_2. \ \ldots \ .a_k$ and an activity a with $a_k^{\text{type}} = a^{\text{type}}$ we have $s.a = a_1. \ \ldots \ .a_{k-1}.(a_k^{\text{type}}, a_k^{\text{length}} + a^{\text{length}})$. For a given schedule $s := a_1.a_2. \ \ldots \ .a_k$ and $1 \leq i \leq k$ let $s_{1,i} := a_1.a_2. \ \ldots \ .a_i$ denote the partial schedule composed of

activities a_1 to a_i. Recall that we assumed that the drivers do not work on Saturdays and Sundays and that we are only interested in generating schedules for a planning horizon starting on Monday and ending on Friday of the same week. For simplicity, we will thus only consider schedules which begin with a rest period representing the rest taken on the weekend preceding the planning horizon. That is, we only consider schedules $s := a_1.a_2. \ \ldots \ .a_k$ with $a_1^{\text{type}} = \texttt{REST}$.

We use the following notation for determining whether a schedule complies with the regulation. For each schedule $s := a_1.a_2. \ \ldots \ .a_k$ with $a_1^{\text{type}} = \texttt{REST}$ we denote with parameter i_s^{420c} the index of the last rest activity of 420 minutes (7 hours) continuous rest, and with parameters i_s^{τ} the index of the last rest activity contributing to a total amount of at least τ minutes of rest before the end of the schedule. More formally, the parameters are defined by

$$i_s^{420c} := \max\left\{i \mid a_i^{\text{type}} = \texttt{REST}, a_i^{\text{length}} \geq 420\right\}$$

and

$$i_s^{\tau} := \max\left\{i \mid \sum_{\substack{i \leq j \leq k \\ a_j^{\text{type}} = \texttt{REST}}} a_j^{\text{length}} \geq \tau\right\}.$$

According to provision 1, the total duration of all non rest activities in schedule s which are scheduled after the rest period with index i_s^{15} must not exceed 315 minutes ($5\frac{1}{4}$ hours). According to provision 2, the total duration of all non rest activities in schedule s which are scheduled after the rest period with index i_s^{30} must not exceed 450 minutes ($7\frac{1}{2}$ hours). According to provision 3, the total duration of all non rest activities in schedule s which are scheduled after the rest period with index i_s^{60} must not exceed 600 minutes (10 hours). According to provision 4, the total duration of all non rest activities in schedule s which are scheduled after the rest period with index i_s^{720} must not exceed 720 minutes (12 hours). If the last activity of schedule s is not a rest period, provision 4 furthermore requires that the total duration of all activities which are scheduled after the rest period with index i_s^{420c} must not exceed 1020 minutes (17 hours). If the last activity of schedule s is a rest period, this rest period can still be extended to rest a period if at least 420 minutes (7 hours). In this case, provision 4 requires that the total duration of all activities which are scheduled after the rest period with index i_s^{420c} and before the last rest period must not exceed 1020 minutes (17 hours).

Let us consider a schedule $s = a_1. \ \ldots \ .a_k$ with $a_1^{\text{type}} = \texttt{REST}$ which complies with the regulation and let a denote some driver activity. Then, schedule $s.a$ complies with the regulation if and only if $a^{\text{type}} = \texttt{REST}$ or

$$a^{\text{length}} \leq 315 - \sum_{\substack{i_s^{15} < j \leq k \\ a_j^{\text{type}} \in \{\texttt{DRIVE},\texttt{WORK}\}}} a_j^{\text{length}} =: \Delta_s^{15}$$

$$a^{\text{length}} \leq 450 - \sum_{\substack{i_s^{30} < j \leq k \\ a_j^{\text{type}} \in \{\texttt{DRIVE},\texttt{WORK}\}}} a_j^{\text{length}} =: \Delta_s^{30}$$

$$a^{\text{length}} \leq 600 - \sum_{\substack{i_s^{60} < j \leq k \\ a_j^{\text{type}} \in \{\texttt{DRIVE},\texttt{WORK}\}}} a_j^{\text{length}} =: \Delta_s^{60}$$

$$a^{\text{length}} \leq 720 - \sum_{\substack{i_s^{720} < j \leq k \\ a_j^{\text{type}} \in \{\texttt{DRIVE},\texttt{WORK}\}}} a_j^{\text{length}} =: \Delta_s^{720}$$

$$a^{\text{length}} \leq 1020 - \sum_{i_s^{420c} < j \leq k} a_j^{\text{length}} =: \Delta_s^{420c}$$

For a given sequence of locations $n_1, n_2, \ldots, n_\lambda$ and a schedule $s = a_1.a_2. \ \ldots \ .a_k$ with $a_1^{\text{type}} = \texttt{REST}$, let us denote with $i(\mu)$ the index corresponding to the μth stationary work period, i.e. $a_{i(\mu)}$ corresponds to the work performed at location n_μ. With this notation we can now give a formal model of the problem. The Australian Truck Driver Scheduling Problem (AUS-TDSP) is the problem of determining whether a schedule $s := a_1.a_2. \ \ldots \ .a_k$ with $a_1^{\text{type}} = \texttt{REST}$ exists which satisfies

$$\sum_{\substack{1 \leq j \leq k \\ a_j^{\text{type}} = \texttt{WORK}}} 1 = \lambda \ \text{ and } \sum_{\substack{i(1) \leq j \leq i(\lambda) \\ a_j^{\text{type}} = \texttt{DRIVE}}} a_j^{\text{length}} = \sum_{\substack{1 \leq j \leq k \\ a_j^{\text{type}} = \texttt{DRIVE}}} a_j^{\text{length}} \tag{1}$$

$$a_{i(\mu)}^{\text{length}} = w_\mu \text{ for each } \mu \in \{1, 2, \ldots, \lambda\} \tag{2}$$

$$l_{s_{1,i(\mu)-1}}^{\text{end}} \in T_\mu \text{ for each } \mu \in \{1, 2, \ldots, \lambda\} \tag{3}$$

$$\sum_{\substack{i(\mu) \leq j \leq i(\mu+1) \\ a_j^{\text{type}} = \texttt{DRIVE}}} a_j^{\text{length}} = \delta_{\mu,\mu+1} \text{ for each } \mu \in \{1, 2, \ldots, \lambda - 1\} \tag{4}$$

$$a_i^{\text{length}} \leq \min\{\Delta_{s_{1,i-1}}^{15}, \Delta_{s_{1,i-1}}^{30}, \Delta_{s_{1,i-1}}^{60}, \Delta_{s_{1,i-1}}^{720}, \Delta_{s_{1,i-1}}^{420c}\} \quad \text{for each } i \in \{1, \ldots, k\} \text{ with } a_i^{\text{type}} \in \{\texttt{DRIVE}, \texttt{WORK}\} \tag{5}$$

Condition (1) demands that the number of work activities in the schedule is λ and that all driving is conducted between the first and the last work activity. Condition (2) demands that the duration of the μth work activity matches the specified work duration at location n_μ. Condition (3) demands that each work activity begins within the corresponding time window. Condition (4) demands that the accumulated driving time between two work activities matches the driving time required to move from one location to the other. Condition (5) demands that the schedule complies with the regulation. In the remainder of this paper, we will say that a schedule $s := a_1.a_2. \ \ldots \ .a_k$ with $a_1^{\text{type}} = \texttt{REST}$ is *feasible* if and only if it satisfies conditions (1) to (5).

4 Structural Properties

Let us now give some structural properties of the truck driver scheduling problem which help us solving the AUS-TDSP without exploring unnecessarily many partial schedules. The first lemma gives us conditions when we can postpone rest periods in order to schedule a driving or working.

Lemma 1. *Let $s := a_1.\ldots.a_k$ be a feasible schedule with $a_i^{type} = \texttt{REST}$ and $a_{i+1}^{type} \in \{\texttt{DRIVE}, \texttt{WORK}\}$ for some $1 < i < k$. If the partial schedule*

$$a_1. \ \ldots \ .a_{i-1}.a_{i+1}$$

complies with the regulation and all relevant time window constraints, then

$$a_1. \ \ldots \ .a_{i-1}.a_{i+1}.a_i.a_{i+2}. \ \ldots \ .a_k$$

is a feasible schedule.

Proof Let $s' := a_1.\ \ldots\ .a_{i-1}.a_i.a_{i+1}$ and $s'' := a_1.\ \ldots\ .a_{i-1}.a_{i+1}.a_i$. Obviously s'' complies with the regulation because $a_i^{\text{type}} = \texttt{REST}$. The only difference between schedule s' and s'' is that in s'' one rest period is moved to a later point in time. Thus, we have

$$\Delta_{s''}^{15} \geq \Delta_{s'}^{15}, \Delta_{s''}^{30} \geq \Delta_{s'}^{30}, \Delta_{s''}^{60} \geq \Delta_{s'}^{60}, \Delta_{s''}^{720} \geq \Delta_{s'}^{720}, \text{ and } \Delta_{s''}^{420c} \geq \Delta_{s'}^{420c}.$$

Assume we have two schedules s' and s'' which comply with the regulation and all relevant time window constraints and which satisfy above conditions. Assume we have, furthermore, an activity a for which $s'.a$ complies with the regulation and all relevant time window constraints. Then we have $a^{\text{type}} = \texttt{REST}$ or $a^{\text{type}} \in \{\texttt{DRIVE}, \texttt{WORK}\}$ and $a^{\text{length}} \leq \min\{\Delta_{s'}^{15}, \Delta_{s'}^{30}, \Delta_{s'}^{60}, \Delta_{s'}^{720}, \Delta_{s'}^{420c}\} \leq \min\{\Delta_{s''}^{15}, \Delta_{s''}^{30}, \Delta_{s''}^{60}, \Delta_{s''}^{720}, \Delta_{s''}^{420c}\}$. Thus, $s''.a$ complies with the regulation and all relevant time window constraints. Furthermore, we have

$$\Delta_{s''.a}^{15} \geq \Delta_{s'.a}^{15}, \Delta_{s''.a}^{30} \geq \Delta_{s'.a}^{30}, \Delta_{s''.a}^{60} \geq \Delta_{s'.a}^{60}, \Delta_{s''.a}^{720} \geq \Delta_{s'.a}^{720}, \text{ and } \Delta_{s''.a}^{420c} \geq \Delta_{s'.a}^{420c}.$$

Therefore, $a_1.\ \ldots\ .a_{i-1}.a_{i+1}.a_i.a_{i+2}.\ \ldots\ .a_k$ is a feasible schedule. □

The next lemma gives us further conditions when we can postpone a rest period in order to schedule some driving time.

Lemma 2. *Let $s := a_1.\ \ldots\ .a_k$ be a feasible schedule with and $a_i^{type} = \texttt{REST}$ and $a_{i+1}^{type} = \texttt{DRIVE}$, $a_{i+1}^{length} > 15$ for some $1 < i < k$. If the partial schedule*

$$a_1.\ \ldots\ .a_{i-1}.(\texttt{DRIVE}, 15)$$

complies with the regulation, then

$$a_1.\ \ldots\ .a_{i-1}.(\texttt{DRIVE}, 15).a_i.(\texttt{DRIVE}, a_{i+1}^{length} - 15).a_{i+2}.\ \ldots\ .a_k$$

is a feasible schedule.

Proof Analogue to first lemma.

The next lemma gives us conditions when we can postpone a part of a rest period of less than 420 minutes.

Lemma 3. *Let $s := a_1.\ \ldots\ .a_k$ be a feasible schedule with and $a_i^{type} = \texttt{REST}$, $15 < a_i^{length} < 420$, and $a_{i+1}^{type} \in \{\texttt{DRIVE}, \texttt{WORK}\}$ for some $1 < i < k$. If*

$$a_1.\ \ldots\ .a_{i-1}.(\texttt{REST}, a_i^{length} - 15).a_{i+1}$$

complies with the regulation and time window constraints, then

$$a_1.\ \ldots\ .a_{i-1}.(\texttt{REST}, a_i^{length} - 15).a_{i+1}.(\texttt{REST}, 15).a_{i+2}.\ \ldots\ .a_k$$

is a feasible schedule.

Proof Analogue to first lemma.

The next lemma gives us further conditions when we can postpone a part of a rest period of less than 420 minutes in order to schedule a some driving time.

Lemma 4. *Let* $s := a_1. \ \ldots \ .a_k$ *be a feasible schedule with and* $a_i^{type} = \texttt{REST}$, $15 < a_i^{length} < 420$, *and* $a_{i+1}^{type} = \texttt{DRIVE}$, $a_{i+1}^{length} > 15$ *for some* $1 < i < k$. *If*

$$a_1.a_2. \ \ldots \ .a_{i-1}.(\texttt{REST}, a_i^{length} - 15).(\texttt{DRIVE}, 15)$$

complies with the regulation, then

$$a_1.a_2.\ldots.a_{i-1}.(\texttt{REST}, a_i^{length}-15).(\texttt{DRIVE}, 15).(\texttt{REST}, 15).(\texttt{DRIVE}, a_{i+1}^{length}-15).a_{i+2}.\ldots.a_k$$

is a feasible schedule.

Proof Analogue to first lemma.

Because of these lemmata we can now state some conditions that we impose on all schedules to be considered when solving the AUS-TDSP. We say that a feasible schedule $s := a_1. \ \ldots \ .a_k$ is *normal form* if and only if

$$\begin{gathered}\text{for all } 1 < i < k \text{ with } a_i^{\text{type}} = \texttt{REST} \text{ and } a_{i+1}^{\text{type}} \in \{\texttt{DRIVE}, \texttt{WORK}\}: \\ a_1. \ \ldots \ .a_{i-1}.a_{i+1} \text{ violates the regulation or some time window}\end{gathered} \tag{N1}$$

$$\begin{gathered}\text{for all } 1 < i < k \text{ with } a_i^{\text{type}} = \texttt{REST} \text{ and } a_{i+1}^{\text{type}} = \texttt{DRIVE}, a_{i+1}^{\text{length}} > 15: \\ a_1. \ \ldots \ .a_{i-1}.(\texttt{DRIVE}, 15) \text{ violates the regulation}\end{gathered} \tag{N2}$$

$$\begin{gathered}\text{for all } 1 < i < k \text{ with } a_i^{\text{type}} = \texttt{REST}, 15 < a_i^{\text{length}} < 420 \text{ and } a_{i+1}^{\text{type}} \in \{\texttt{DRIVE}, \texttt{WORK}\}: \\ a_1. \ \ldots \ .a_{i-1}.(\texttt{REST}, a_i^{\text{length}} - 15).a_{i+1} \text{ violates the regulation or some time window}\end{gathered} \tag{N3}$$

$$\begin{gathered}\text{for all } 1 < i < k \text{ with } a_i^{\text{type}} = \texttt{REST}, 15 < a_i^{\text{length}} < 420, a_{i+1}^{\text{type}} = \texttt{DRIVE}, a_{i+1}^{\text{length}} > 15: \\ a_1. \ \ldots \ .a_{i-1}.(\texttt{REST}, a_i^{\text{length}} - 15).(\texttt{DRIVE}, 15) \text{ violates the regulation}\end{gathered} \tag{N4}$$

If a feasible schedule for a given tour exists, there also exists a feasible schedule in normal form. Thus, we can ignore all schedules which are not in normal form when searching for a feasible truck driver schedule.

5 Solution Approaches

Assume we knew the start time and end time of each rest period of 7 hours or more. We could try to construct a feasible schedule in normal form by iteratively scheduling driving or working activities as early as possible and rest activities as late as possible. The duration of all driving activities would be set to the largest possible value and the duration of all rest activities scheduled would be set to the smallest possible value. If no feasible schedule in normal form can be constructed by this procedure, no feasible schedule exists.

Unfortunately, determining when a rest period of at least 7 hours should be scheduled and how long this rest period should be is a difficult task. Therefore, we will presents several heuristics for scheduling these rest periods in this paper. These heuristics use the framework given in Figure 1. The heuristic framework begins by choosing a partial schedule s which is feasible for the tour $n_1, \ldots, n_\mu$ and sets δ to the driving

time required to reach location $n_{\mu+1}$. As long as the next location is not yet reached (i.e. $\delta > 0$), the maximum amount of driving allowed with respect to condition (5) is determined. If condition (5) forbids any driving, a rest period of 15 minutes is appended to the schedule. Otherwise, the longest possible driving period is appended to the schedule and δ is updated. When the next location is reached (i.e. $\delta = 0$), as much rest time as necessary in order to be able to schedule the next working period of duration $w_{\mu+1}$ is appended to the schedule. Then, depending on the specific method, the set $\mathcal{S}(s, \mu+1)$ is determined and included into the set of schedules found for tour $n_1, \ldots, n_{\mu+1}$.

---

1. choose $s \in \mathcal{S}_\mu$, set $\delta := \delta_{\mu,\mu+1}$
2. while $\delta > 0$ do
   - $\Delta := \min\{\Delta_s^{15}, \Delta_s^{30}, \Delta_s^{60}, \Delta_s^{720}, \Delta_s^{420c}\}$
   - $s := \begin{cases} s.(\texttt{REST}, 15) & \text{if } \Delta = 0 \\ s.(\texttt{DRIVE}, \min\{\delta, \Delta\}) & \text{if } \Delta > 0 \end{cases}$
   - $\delta := \delta - \min\{\delta, \Delta\}$
3. while $w_{\mu+1} > \min\{\Delta_s^{15}, \Delta_s^{30}, \Delta_s^{60}, \Delta_s^{720}\}$ do
   - $s := s.(\texttt{REST}, 15)$
4. $\mathcal{S}_{\mu+1} := \mathcal{S}_{\mu+1} \cup \mathcal{S}(s, \mu+1)$

Fig. 1 Heuristic framework for scheduling activities for the trip from location n_μ to $n_{\mu+1}$

The AUS1 heuristic is a greedy heuristic in which $\mathcal{S}(s, \mu+1)$ contains at most one schedule. If $l_s^{\text{end}} \in T_{\mu+1}$ and $w_{\mu+1} \leq \Delta_s^{420c}$ then $\mathcal{S}(s, \mu+1) := \{s.(\texttt{WORK}, w_{\mu+1})\}$. Otherwise, if for some $\Delta > 0$ a feasible schedule $s.(\texttt{REST}, \Delta).(\texttt{WORK}, w_{\mu+1})$ for tour $n_1, \ldots, n_{\mu+1}$ exists, then $\mathcal{S}(s, \mu+1)$ contains the feasible schedule $s.(\texttt{REST}, \Delta).(\texttt{WORK}, w_{\mu+1})$ with the smallest value Δ. If no such schedule exists, then $\mathcal{S}(s, \mu+1) := \emptyset$.

In the AUS2 heuristic $\mathcal{S}(s, \mu+1)$ contains at most two schedules. The first is the schedule which is also determined by the AUS1 heuristic. The second schedule is only included to the set if $l_s^{\text{end}} < \min T_{\mu+1}$ or if the last activity of s is of type REST and has a duration of less than 7 hours. Let a denote the last activity of s and let

$$\Delta' := \begin{cases} a^{\text{length}} & \text{if } a^{\text{type}} = \texttt{REST} \\ 0 & \text{else} \end{cases}$$

If for some $\Delta > 0$ with $\Delta' + \Delta \geq 420$ a feasible schedule $s.(\texttt{REST}, \Delta).(\texttt{WORK}, w_{\mu+1})$ for tour $n_1, \ldots, n_{\mu+1}$ exists, then the feasible schedule $s.(\texttt{REST}, \Delta).(\texttt{WORK}, w_{\mu+1})$ with the smallest such value Δ is included to $\mathcal{S}(s, \mu+1)$. If no such schedule exists then $\mathcal{S}(s, \mu+1)$ is the same as for the AUS1 heuristic.

In the AUS3 heuristic $\mathcal{S}(s, \mu+1)$ contains at most three schedules. The first two are the schedules which are also determined by the AUS2 heuristic. The third schedule is only included to the set if $l_s^{\text{end}} < \min T_{\mu+1}$. Let s' denote the schedule which is obtained by extending the last rest period in s, which has a duration of at least 420 minutes, by the largest value which does not exceed $\min T_{\mu+1} - l_s^{\text{end}}$ and for which time window constraints are not violated for any work location visited after the last rest period. If $l_{s'}^{\text{end}} < \min T_{\mu+1}$ and $w_{\mu+1} + \min T_{\mu+1} - l_{s'}^{\text{end}} \leq \Delta_{s'}^{420c}$ then $s'.(\texttt{REST}, \min T_{\mu+1} - l_{s'}^{\text{end}}).(\texttt{WORK}, w_{\mu+1})$ is included to $\mathcal{S}(s, \mu+1)$. If $l_{s'}^{\text{end}} = \min T_{\mu+1}$ and $w_{\mu+1} \leq \Delta_{s'}^{420c}$ then $s'.(\texttt{WORK}, w_{\mu+1})$ is included to $\mathcal{S}(s, \mu+1)$. Otherwise, $\mathcal{S}(s, \mu+1)$ is the same as for the AUS2 heuristic.

The AUS1, AUS2, and AUS3 heuristics begin with

$$\mathcal{S}_1 := \{(\texttt{REST}, \max\{2880, \min T_1\}).(\texttt{WORK}, w_1)\}$$

and $\mu = 1$. Then, the method illustrated in Figure 1 is invoked until each schedule in $\mathcal{S}_\mu$ has been selected. If $\mathcal{S}_{\mu+1} = \emptyset$, the heuristic terminates because no feasible schedule is found for tour $n_1, \ldots, n_{\mu+1}$. Otherwise, μ is incremented and the process is repeated until $\mathcal{S}_\lambda \neq \emptyset$.

To reduce the computational effort of the AUS2 and AUS3 heuristic some partial schedules are removed from $\mathcal{S}_\mu$ before invoking the scheduling method. Let us consider two schedules $s', s'' \in \mathcal{S}_\mu$ for some $1 < \mu < \lambda$. If $l_{s'}^{\text{end}} + 720 \leq l_{s''}^{\text{end}}$, then s'' is removed from $\mathcal{S}_\mu$. In the case that s'' is not removed and s' ends with a rest period followed by a work period let a denote this rest period and let

$$\Delta := \begin{cases} \max\{420, 720 - a^{\text{length}}\} & \text{if } a^{\text{length}} < 420 \\ \max\{0, 720 - a^{\text{length}}\} & \text{if } a^{\text{length}} \geq 420. \end{cases}$$

If $l_{s'}^{\text{end}} + \Delta \leq l_{s''}^{\text{end}}$, then s'' is removed from $\mathcal{S}_\mu$.

6 Computational Experiments

Scheduling of driving and working hours is of particular importances for long distance haulage where drivers do not return home every day. In order to evaluate the scheduling method presented in this paper we generate benchmark instances for a planning horizon starting on Monday morning and ending on Friday evening. In the benchmark set each handling activity requires one hour of working time (i.e. $w_\mu = 1$ for all $1 \leq \mu \leq \lambda$) and the driving time between two subsequent locations is 4, 8, 12, or 16 hours (i.e. $\delta_{\mu,\mu+1} \in \{4, 8, 12, 16\}$ for all $1 \leq \mu < \lambda$). Assuming an average speed of 75 km/h, this implies that the distance between two subsequent locations ranges from 300 km to 1200 km. Each location must be visited on a specific day between 6.00h and 13.59h or between 14.00h and 21.59h.

Algorithm	Instances	Computation time
AUS1	64,785	123'43"
AUS2	67,556	275'57"
AUS3	67,556	360'29"

Table 1 Number of instances for which a feasible schedule is found by the method and total computation time required

In total, around 15.6 million instances where generated in which time window constraints could be satisfied if driving and working times were unrestricted. Only 3,166,146 of these instances do not exceed the accumulated weekly working time of 72 hours. All other instances are discarded by the heuristics before starting to construct schedules. Table 1 gives an overview of the number of instances for which the AUS1, AUS2 and AUS3 heuristic find a feasible schedule and the total computation time required. The AUS2 heuristic can find a feasible schedule for 2,771 instances more than the AUS1 heuristic. However, it requires almost double the computation time. Although it is easy to find examples in which the AUS3 heuristics is superior to the

AUS2 heuristic, the AUS3 heuristic cannot find a feasible schedule for more instances considered in this experiment. The AUS1 heuristic has by far the smallest running time and is capable of finding a feasible schedule for approximately 96 per cent of the instances for which the more sophisticated approaches can find a feasible schedule. Thus, it appears that, for similarly structured practical problem instances, the AUS1 heuristic has the best trade-off between exactness and computational effort.

7 Summary

This paper studies the Australian Heavy Vehicle Driver Fatigue Law and formulates the Australian Truck Driver Scheduling Problem. Structural properties of the problem are analysed and used to develop heuristics for solving the problem.

Acknowledgements This research was supported by the German Research Foundation (DFG) and the National ICT Australia (NICTA).

References

C. Archetti and M. W. P. Savelsbergh. The trip scheduling problem. *Transportation Science*, 43(4):417–431, 2009. doi: 10.1287/trsc.1090.0278.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127(1):21–144, March 2004. doi: 10.1023/b:anor.0000019087.46656.e2.

A. Goel. Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, 43(1):17–26, 2009. doi: 10.1287/trsc.1070.0226.

A. Goel. Truck Driver Scheduling in the European Union. *Transportation Science*, (to appear), 2010.

A. L. Kok, C. M. Meyer, H. Kopfer, and J. M. J. Schutten. Dynamic Programming Algorithm for the Vehicle Routing Problem with Time Windows and EC Social Legislation. Beta working paper 270, University of Twente, 2009.

National Transport Commission. Heavy Vehicle Driver Fatique Reform – Standard Hours explained. Information Bulletin, 2008a.

National Transport Commission. Heavy Vehicle Driver Fatique Reform – Advanced Fatigue Management explained. Information Bulletin, 2008b.

National Transport Commission. Heavy Vehicle Driver Fatique Reform – Basic Fatigue Management explained. Information Bulletin, 2008c.

A. Williamson, S. Sadural, A. Feyer, and R. Friswell. Driver Fatigue: A Survey of Long Distance Heavy Vehicle Drivers in Australia. Information Paper CR 198, National Road Transport Commission, 2001.

H. Xu, Z.-L. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37(3):347–364, 2003.

Distributed Scatter Search for the Examination Timetabling Problem

Christos Gogos[1,2], George Goulas[1], Panayiotis Alefragis[1,3], Vasilios Kolonias[1] and Efthymios Housos[1]

[1]*University Of Patras. Dept. of Electrical and Computer Engineering, Rio Patras, Greece.*

[2]*Technological Educational Institute of Epirus. Dept. of Finance and Auditing, Psathaki, Preveza, Greece.*

[3]*Technological Educational Institute of Mesolonghi. Dept. of Telecommunication Systems and Networks, Varia, Nafpaktos, Greece.*

Abstract: Examination Timetabling for Universities is a problem with significant practical importance. It belongs to the general class of educational timetabling problems and has been exposed to numerous approaches for solving it. We propose a parallel/distributed solution which is based on the metaheuristic method Scatter Search combined with Path Relinking in an attempt to diversify the search procedure by producing promising new timetables. Our approach improves on the best publicly available results for the datasets of ITC2007 (International Timetabling Competition 2007-2008). The constraint of limited execution time that was imposed by ITC2007 was disregarded in an effort to pursue the best values our approach could reach. We consider this specific examination timetabling problem as a "test bed" for timetabling problems in general and we expect to provide insight for developing effective solution processes for other practical scheduling problems.

Keywords: scatter search, path relinking, examination timetabling

1. Introduction

The advent of multicore processors, cloud computing and programmable Graphical Processing Units, to name just three of recent massive processing technologies, offer nowadays abundant processing power. Difficult practical problems can be revisited and new solution methods can be sought under the presence of distributed or parallel environments of execution. Under conditions of vast availability of computational resources, mixed integer and dynamic programming approaches, which generate provable optimal solutions, become interesting. Equally interesting are metaheuristic approaches that sacrifice optimality but are in general simpler to implement and can give very good results.

The Examination Timetabling Problem (ETP) has received ample attention and numerous approaches from and across various disciplines have been proposed for solving it. Some of the approaches that have given satisfactory results are: Constraint Programming (David, 1998), Hybrid methods (Qu and Burke, 2009), Hyper Heuristics (Pillay and Banzhaf, 2008) and various

Metaheuristics (Ersoy et al., 2007). In this contribution the ITC2007[1] model of the Examination Timetabling Problem is solved by a Scatter Search metaheuristic and the whole process is undertaken by our distributed execution framework called SchedScripter. The solutions that we have obtained indicate that significant potential of using analogous methods to similar problems exists. We also observed that the benefit of reaching better solutions was complemented by the robustness of the procedure.

The rest of the paper is organized as follows. Section 2 describes the ETP. Section 3 presents an introduction to parallel/distributed execution environments and metaheuristics. The SchedScripter framework is also presented in the same section. SchedScripter is used for communications and workload management needed by the distributed Scatter Search application. Section 4 describes the distributed Scatter Search solution approach, with subsections describing Scatter Search and Path Relinking. Section 5 presents the experimental results. Section 6 concludes the paper asserting that distributed Scatter Search can outperform various single processor approaches in terms of solution quality.

2. Problem Description

The ETP belongs to the general class of timetabling problems which are known to be NP-complete under certain conditions (Schaerf, 1999). Several different formulations of ETP exist ranging from rather simple ones to more complicated (Qu et al., 2009). Historically, early formulations considered only the avoidance of exam conflicts thus drawing parallelism between ETP and graph coloring problems (Carter, 1996) while gradually details regarding rooms, periods and exams were added. The main objective of the ETP is to produce timetables giving adequate time between exams for all participating students.

In our contribution, we use the ITC2007 formulation of the problem. Under this formulation a set of exams has to be scheduled in a set of time periods and rooms while at the same time a number of hard constraints have to be satisfied. Hard constraints are avoidance of conflicts between exams for all students, respect of room capacities, matching of exam duration with assigned period duration and various constraints regarding ordering between exams and pre-assignment of exams to certain rooms. The quality of the solution is measured against cost penalties imposed by a set of soft constraints. Examples of soft constraints considered are the presence of two exams in a row and two exams in the same day for a particular student, early scheduling of exams with large audiences, mixing of exams with different durations in the same room and the usage of rooms and periods that have been marked as undesirable. Period spread is also a soft constraint meaning that occurrences of consecutive exams for each student within a predefined range of periods are penalized. Every soft constraint is related with a weight that prescribes its penalty. Each university defines the weight of every constraint according to its preferences thus creating a vector called the Institutional Model Index (IMI). The objective function is a weighted sum of the soft constraint violations according to the weights defined in IMI.

[1] http://www.cs.qub.ac.uk/itc2007

A thorough description of the ITC2007 ETP formulation can be consulted in (McCollum et al. 2009a) while further details regarding ITC2007 issues in general can be found in (McCollum et al. 2009b). Twelve datasets were provided for benchmarking and data associated with them are presented in Table 1. The last four of the datasets were characterized as hidden indicating the intention of the organizers to make them publicly available after the completion of the competition. The 12 datasets present significant variation in their characteristics which is also observed in real life examination timetabling problems (McCollum, 2007). Therefore, a generic successful algorithmic approach should not make any assumptions about specific values.

ITC2007-DATASETS														
	Exams	Students	Periods	Rooms	Period HC	Room HC	Two In A Row Penalty	Two In A Day Penalty	Period Spread Penalty	No Mixed Durations Penalty	Number Of Largest Exams	Number Of Last Periods To Avoid	Frontload Penalty	Conflict Density
Dataset 1	607	7891	54	7	12	0	7	5	5	10	100	30	5	5.05%
Dataset 2	870	12743	40	49	12	2	15	5	1	25	250	30	5	1.17%
Dataset 3	934	16439	36	48	170	15	15	10	4	20	200	20	10	2.62%
Dataset 4	273	5045	21	1	40	0	9	5	2	10	50	10	5	15.00%
Dataset 5	1018	9253	42	3	27	0	40	15	5	0	250	30	10	0.87%
Dataset 6	242	7909	16	8	23	0	20	5	20	25	25	30	15	6.16%
Dataset 7	1096	14676	80	15	28	0	25	5	10	15	250	30	10	1.93%
Dataset 8	598	7718	80	8	20	1	150	0	15	25	250	30	5	4.55%
Dataset 9	169	655	25	3	10	0	25	10	5	25	100	10	5	7.84%
Dataset 10	214	1577	32	48	58	0	50	0	20	25	100	10	5	4.97%
Dataset 11	934	16349	26	40	170	15	10	50	4	35	400	20	10	2.62%
Dataset 12	78	1653	12	50	9	7	35	10	5	5	25	5	10	18.45%

Table 1. Datasets characteristics

3. Parallel / Distributed Execution Environments and Metaheuristics

It is often the case that real life problems generate problem instances that require vast amounts of CPU time to create optimal feasible solutions. Although the use of metaheuristics allows significant reduction of the search process computational complexity, the wall clock time is still the major performance measurement. End-users in many application areas require that high quality solutions be obtained as soon as possible. In such applications the cost of hardware resources that may be required is considered a minor issue. A detailed presentation of the possibilities in metaheuristic design can be found in (Talbi 2009). Recently, parallel processing has again gained significant attention as various technology advancements has been performed in processor design (multicore processors, GPU computing) and interconnecting networks (e.g. Infiniband). Moreover, Grid technologies allow the exploitation of vast amounts of usually volatile loosely coupled computational resources, creating an opportunity to exploit such architectures for the design and implementation of parallel metaheuristics.

Many aspects, design decisions and goals have to be considered during parallel metaheuristic algorithm design. The most usual goal is to speed up the search process or the improvement of the quality of the obtained solutions. The former allows the design of real-time or interactive optimization methods, while the latter allows the cooperating metaheuristics to obtain better convergence characteristics while reducing search time. Researchers or practitioners using parallel processing have managed to improve the robustness of the obtained solutions for various difficult

practical problems. This is usually achieved by reducing the sensitivity of the metaheuristics to their parameters and by managing to examine the search space in greater detail. During parallel metaheuristic design, questions such as exchange decision criterion (when?), exchange topology (where?), information exchanged (what?) and integration policy (how?) have to be answered. Detailed presentation of various parallel metaheuristics can be found in (Alba, 2005).

Our approach was to create a middleware architecture that isolates parallel algorithm design components from the mapping to parallel architectures, providing a toolbox to rapidly create different parallelization approaches.

3.1 SchedScripter framework

SchedScripter (Gogos et al., 2009), (Goulas et al., 2009) is a software framework to assist in the development of distributed, grid-based, human resources scheduling applications. SchedScripter covers an area between loosely-coupled grid-based workflow systems (Yu and Buyya 2005) and message passing libraries. Grid workflow systems create execution workflows based on independent jobs and their data dependencies, while message passing libraries assist in the creation of parallel/distributed applications using tight message passing protocols like the MPI (Gropp et al., 1999). SchedScripter applications main components are the SchedScripter registry, the SchedScripter worker nodes and the application coordination process (Figure 1). SchedScripter installs a web service container on every worker node to provide a set of services designed to allow the worker node to be considered as a single process of a worker collection. A single master node provides a registry for the worker node services and ensures that workers remain available. All SchedScripter services are offered as XML Web Services for interoperability and ease of access. A SchedScripter-based application needs to access the registry to find resources and then transfer tasks and code to worker nodes. This process is assisted by an application level task scheduler offered in the SchedScripter API.

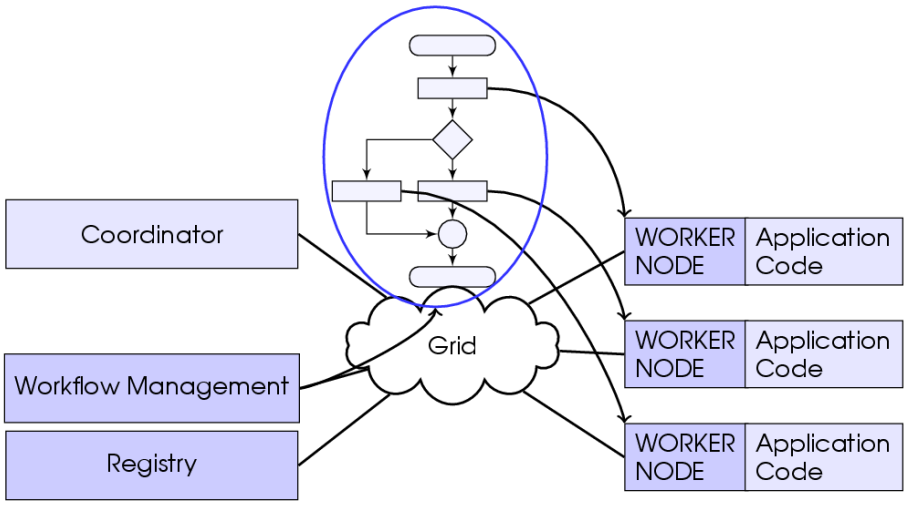


Figure 1. SchedScripter architecture

A natural coordination scheme for web services, supported by SchedScripter, is the use of Business Process Execution Language (BPEL[2]) workflows. BPEL allows for web service message interactions providing programming language constructs. BPEL tools offer graphical representation of application workflows which is a great visualization tool. In order to assist resource discovery and utilization using BPEL, the SchedScripter application-level task scheduler is available as a web service. Furthermore, since BPEL processes are themselves web services, BPEL can offer layers of abstraction, composite services and reuse. The SchedScripter master includes Apache ODE[3] as a BPEL engine, compatible with both BPEL versions, BPEL4WS 1.1 and WS-BPEL 2.0.

While web services and BPEL are an easy way to create distributed applications, it was clearly demonstrated that it is common for developers to have difficulties in understanding the concepts of Service Oriented Architecture so as to use web services and BPEL effectively. To narrow this knowledge gap, SchedScripter tries to abstract the process of applying specific distributed application patterns and offers them as Application Templates in the form of a Java API. The main patterns supported are the master/worker paradigm and the swarm pattern. The master/worker is a very popular, centralized pattern where a single master process splits the workload into tasks and assigns them to worker nodes. The swarm pattern, also usually referred as peer-to-peer, assumes a set of independent processes, each of them deciding on the task to accomplish in order for the whole swarm to solve the general problem. These processes communicate frequently with broadcasted messages, in order to publish their findings and provide hints to the swarm, which may or may not be used. The swarm, in SchedScripter, has a single master, whose role is to monitor message exchanges and store the result at the end of the process. While SchedScripter was originally developed to be used in the EGEE[4] grid infrastructure, the framework is generic and can be used outside EGEE as well, even in a cluster environment without grid middleware. Indeed, the results of this paper were collected during runs on the small cluster of servers of a newly acquired blade system running Ubuntu Karmic 9.10 server operating system.

4. Solution Process

The solution process that we have chosen is based on metaheuristic Scatter Search and uses data structures and algorithms from our previous work described in (Gogos et al, 2010). A stage of Simulated Annealing and a stage of Shaking are combined in a single improvement phase that is employed whenever an attempt to improve a solution occurs. In the Shaking stage a set of exams is formed based on the cost per student contribution of each exam giving preference to higher values. The selected exams are removed and then scheduled once again causing distortion to the timetable.

---

[2] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[3] http://ode.apache.org

[4] http://www.eu-egee.org

4.1 Scatter Search

Scatter Search is a well known metaheuristic that was originally proposed by Glover for Integer Programming problems as a way of combining critical constraints in order to produce new "surrogate" constraints (Glover, 1977). It was also included as a stage in the original Tabu Search metaheuristic but it was seldom used in various implementations of it. Paper (Glover 1998) played a key role in establishing Scatter Search as an independent self contained metaheuristic. Detailed descriptions of Scatter Search can be found in (Glover et al., 2002), (Glover et al., 2003) and (Laguna and Marti, 2006) while in (Laguna and Armentano, 2000) several practical advices about implementing Scatter Search are illustrated. It has to be mentioned that Scatter Search has been recently employed in (Mansour et al., 2009) for tackling a modified formulation ETP giving promising results.

Scatter Search is classified as an evolutionary algorithm. It maintains a population of solutions that are recombined in order for better solutions to be achieved. Similarities exist with the widely known Genetic Algorithm metaheuristic but Scatter Search maintain smaller population of solutions, is less dependent on randomization, improves the solution after each recombination of solutions and systematically injects diversity into the population. A key point of Scatter Search is the concept of reference points (RPs). RP is a "good" solution that has been obtained from a previous solution effort. RPs are systematically combined in order for new solutions to be generated. Scatter Search starts by generating a pool of good and diverse solutions. From this pool a reference set (RefSet) is formed that contains not only the best solutions but also solutions that are different from the other already included in the RefSet. Next, new solutions are formed by combining members of the RefSet. Each produced solution is improved through heuristic methods. The best improved solutions are inserted into the RefSet causing the removal of inferior solutions. The procedure continues in an iterative manner creating combinations of the newly inserted solutions with already existing solutions of the RefSet. When no new solutions can be inserted into the RefSet then either the procedure stops or certain actions are initiated that diversify RefSet by replacing solutions with less good but significantly different ones. The general template of Scatter Search consists of five methods forming a cycle of continuous improvement over a set of solutions. Our implementation of the Scatter Search is shown in (Figure 2) where each solution is depicted by a hexagon. The role that each method plays in solving the ETP is presented in the following paragraphs.

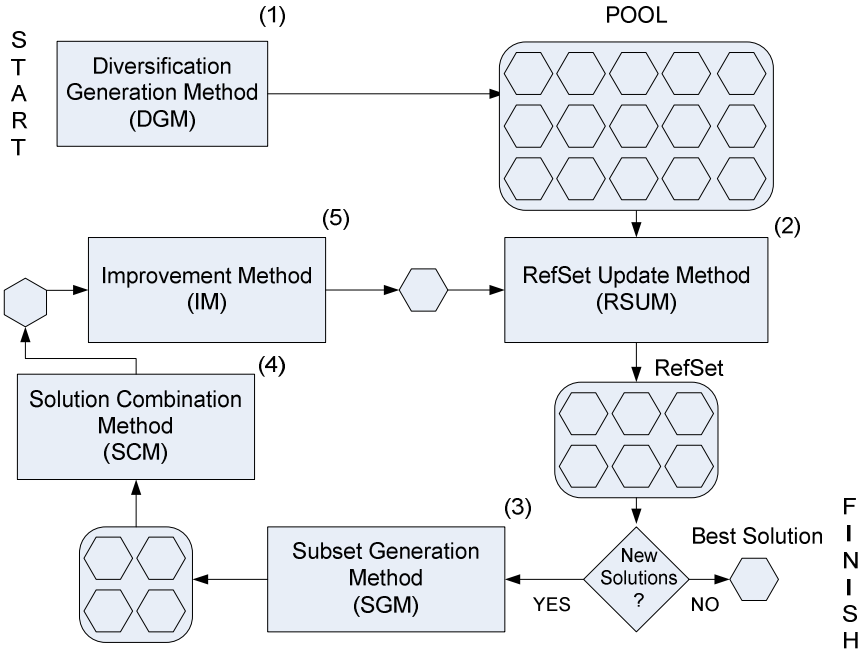


Figure 2. Scatter Search 5 method template

Diversification Generation Method (DGM).

The purpose of this method is to generate a collection of diverse timetables. In our approach we simply loaded a pool with 100 solutions from past experiments done in (Gogos et al., 2010). Alternatively, the pool could be generated on demand by running several time bounded constructions and improvements while modifying various parameters like construction over improvement time, simulated annealing cooling schema etc. Experiments showed that the exact method of populating the pool does not hinder the robustness of the approach. Then, an initial RefSet is constructed by selecting exams from the pool. Half of RefSet solutions are selected from the pool based on their cost. The other half is completed by selecting solutions that exhibit the biggest dissimilarity with the timetable of the RefSet that is more similar to the selected solution. It has to be noted that diversification is not the same as randomization because the whole process is biased towards selecting solutions that differ from other ones.

Improvement Method (IM).

Improvement of each solution is undertaken by a cycle of two stages. These stages are "simulated annealing" and "shaking". Simulated annealing allows, in a systematic way, moves to inferior solutions thus enabling the possibility of escape from local best values basins. On the other hand the purpose of the shaking stage is to dislocate the current solution so as new searches for better values to start from yet unexplored points of the search space. In our distributed execution approach a master/slave schema is employed. The improvement method executes in parallel by worker nodes while a coordinator is responsible for collecting results and producing new jobs under the SchedScipter framework described in Section 3.

RefSet Update Method (RSUM).

Solutions become members of the RefSet based on their cost. Inferior timetables may also be accepted provided that they are sufficiently different from existing timetables in RefSet. The

similarity of two solutions is computed based on the number of exams that are scheduled in the same period and room in both solutions.

Subset Generation Method (SGM).

This method operates on the RefSet and produces a subset that will be used in order for solutions to be combined. In our approach pairs of solutions are formed ensuring that each pair has not already been examined in the past.

Solution Combination Method (SCM).

Path Relinking occurs in this method. Pairs of timetables are combined in order to construct new solutions. A given timetable is gradually transformed to another timetable (guiding solution) by repositioning exams to periods and rooms. In order for the solution to be in the feasible area backtracking moves that remove offending exams and then reschedule them might be necessary.

4.2 Path Relinking

Suppose the existence of two complete timetables A and B assuming that A is superior to B considering cost value. The objective during Path Relinking is to gradually transform A to B hoping that promising timetables will arise during the process (backward relinking). At each step of the procedure, certain attributes of A are modified so as to become identical with those of B. During the transformation process intermediate, but complete, solutions are stored whenever they are considered to be of value for subsequent stages of the overall process.

More specifically timetable A becomes timetable B gradually in cycles. In each cycle a group of exams are selected from timetable A and rescheduled to new periods and rooms so as to comply with those found in timetable B. It is possible that the rescheduling of a group of exams in A will result in cascaded repositions of other exams in order to retain the feasibility of the solution. Whenever the resulting timetable has more differences than the original, a rollback is performed to the previous state of timetable A and an extra exam is added to the initial group of exams. The use of the Command design pattern (Gamma et al., 1994) made rollback moves easy to be programmed. The pseudo-code of the above process follows:

Step 1: Set N=1. Find differences between A and B and put them on set DIFFS. Set MIN_SIZE=|DIFFS|. Set LEVEL=MIN_SIZE

Step 2: If MIN_SIZE=0 then terminate. Else all exams of A which are members of DIFFS that can be scheduled to the same period and room as in B are repositioned accordingly.

Step 3: N exams are selected randomly from set DIFFS. For each selected exam other exams must be removed from the timetable so as to allow the scheduling of the selected exam to the period and room dictated by B. Priority for removal is given to exams that are not scheduled in the same period and room as in B. If removing these exams is not sufficient extra exams are selected on the basis of evoking minimum removals of other exams in previous stages of the algorithm. Extra exams are selected one by one until the selected exam can be scheduled in the correct period and room. Each one of the N exams are scheduled in the period and room found in solution B, while a set of exams called UNSCHED contains exams that no longer belong to the timetable.

Step 4: All exams of set UNSCHED are scheduled to any of the available periods and rooms until set UNSCHED become empty. In the process, removal of already scheduled exams is possible. If this is the case then exams for removal are selected based again as in Step 3 on the history of removals caused by each candidate exam. It should be mentioned that during this step no restrictions apply on which exams could be removed.

Step 5: Update set DIFFS by finding differences between solution A and B. If |DIFFS|>=MIN_SIZE then N=N+1 else MIN_SIZE=|DIFFS|.

Step 6: If MIN_SIZE<LEVEL then LEVEL=MIN_SIZE and N=1. Return to Step 2.

5. Experiments

While the application development and debugging has been carried out on a small cluster of desktop PCs and the South East Europe part of the EGEE infrastructure, the results were obtained running on a brand-new IBM Blade system. This system consists of 14 independent servers in a single enclosure, each of them powered by a quad core hyper-threaded CPU Xeon processor based on Nehalem architecture. These processors were coordinated by another server, which we call the head node, a dual CPU quad core Xeon based on Clovertown architecture.

The head node hosts the SchedScripter master, which includes the services registry, as well as the Distributed Scatter Search application coordinator. The 14 blade servers host the worker node process and in order to fully exploit the 8 virtual cores (4 real ones hyper-thread), 8 processes start on each server. This system was installed recently, so there was little time to measure the impact of possible memory bandwidth bottlenecks. The different coordination process runs were managed using Sun Grid Engine and a local queue with one slot on the head node. Several configuration experiments were taking place at one or two nodes while the experiments were running, removing these servers eventually from the available resources, so the resource pool was varying from 12 to 14 blades, or from 96 to 112 worker processes. The same experiment running on the EGEE grid would suffer much more uncertainty in resource numbers, as on the grid resources enter and leave at random times during runtime as the system lacks rendezvous mechanisms.

Every time the coordinator finds a new worker node, it transfers the necessary executables and instructs it to load the problem dataset, which is about a minute long process. After this initial step, specific improvement tasks are sent to the worker nodes. An improvement task is an instruction to perform local search (Simulated Annealing and Shaking), starting from a specific solution with a specific set of local search parameters and a timeout period. When the worker node finishes the local search, it sends the resulting solution back to the coordinator, using a web service.

SchedScripter, being a grid framework, is highly tolerant to resource failure events and dynamically resizes the resource pool at runtime. Distributed Scatter Search Coordinator as well, has been created as a fault-tolerant process, able to dynamically resize its workers pool. Indeed, since the biggest part of the debug sessions took place in the EGEE grid, where we submit about a couple of hundred of worker jobs, these jobs start at random times as the grid workload management system decides. These potential problems in reliability lead us to include fault-tolerance as an inherent part of the coordination approach.

Table 2 displays the best results for each dataset of the problem under the runtime limit of about 10 minutes imposed by the ITC2007 competition. The second column contains the best results collected from the winner of the competition, Tomas Muller, after 100 runs (Muller, 2008). It should be noted that no data exist in this paper for the former hidden datasets because at that time those datasets were not available. The next two columns are the best results collected after 51 independent runs as cited in (McCollum et al., 2009). The last column of the table depicts the best results over 100 runs as cited in (Gogos et al., 2010). The last 4 values of the last column are not included in (Gogos et. al., 2010) so new runs for those datasets were made and the values recorded are the best results collected over 100 runs.

Instance	(Muller, 2008)	(McCollum et al, 2009) (Post ITC2007)	Muller (Post ITC2007)	(Gogos et al, 2010)
Dataset 1	**4.356**	4.663	4.370	4.775
Dataset 2	390	405	**385**	**385**
Dataset 3	9.568	9.064	9.378	**8.996**
Dataset 4	16.591	15.663	**15.368**	16.204
Dataset 5	2.941	3.042	2.988	**2.929**
Dataset 6	25.775	25.880	26.365	**25.740**
Dataset 7	4.088	**4.037**	4.138	4.087
Dataset 8	7.565	**7.461**	7.516	7.777
Dataset 9	X	1.071	1.014	***964***
Dataset 10	X	14.374	14.555	***13.203***
Dataset 11	X	29.180	31.425	***28.704***
Dataset 12	X	5.693	5.357	***5.197***

Table 2. Best results under time limit

The configuration used in our experiment with Scatter Search included a RefSet of 20 solutions, collection of 4 solutions during each Path Relinking, 120 seconds available for each improvement attempt and total runtime of 4 hours. In Table 3 the results of our approach are presented and compared alongside with results obtained in (Gogos et al., 2009) with the current approach giving better results in all datasets. The last column of the table shows the percentage of improvement achieved over best results of Table 2. For all 12 datasets improvement was achieved, while for certain datasets improvement was beyond our initial expectations.

Instance	(Gogos et al, 2009)	Current approach (Scatter Search)	Percentage of improvement over best results of Table 2
Dataset 1	4.699	**4.128**	5,23%
Dataset 2	385	**380**	1,30%
Dataset 3	8.500	**7.769**	13,64%
Dataset 4	14.879	**13.103**	14,74%
Dataset 5	2.795	**2.513**	14,20%
Dataset 6	25.410	**25.330**	1,59%
Dataset 7	3.884	**3.537**	12,39%
Dataset 8	7.440	**7.087**	5,01%
Dataset 9	X	**913**	5,29%
Dataset 10	X	**13.053**	1,14%
Dataset 11	X	**24.369**	15,19%
Dataset 12	X	**5.095**	1,96%

Table 3. Best results under no hardware or time limits

In order to be fair in our comparisons a few points have to be stressed out. The execution environment is different between approaches included in Table 2 and Table 3. In the former case a

time limit of about 10 minutes was given while in the latter no practical time limit was imposed. Furthermore, the number of runs is different. In Table 2 the results are the best collected from 51 or 100 runs while in Table 3 the results are collected from a single experiment that consumed more processing power than all runs in each dataset of Table 2. However, we believe that our approach demonstrates an effective method for using additional time and CPU resources if they are available.

6. Conclusions

Evolutionary algorithms usually can be well adapted for execution in parallel or distributed systems. This is the case for Scatter Search too. Creation of the initial population and processing of solution combinations can be greatly accelerated using distributed workers. In this contribution we have proposed a Scatter search technique for tackling the examination timetabling problem harnessing the processing power of a very powerful computer system. Results show improvements over best known values for all datasets of the ITC2007 ETP problem, which are for some cases significant. We acknowledge the fact that results we compare with were obtained under a small fraction of the processing time that we spent but we believe that for certain problems this extra time can be allocated. Consequently, whenever "very good" solutions for difficult practical problems are of significant importance over simply "good" solutions experimentation with scatter search or other evolutionary metaheuristics combined with distributed execution environments have increased possibilities to pay off.

References

Alba, E. Editor (2005). Parallel Metaheuristics, A New Class of Algorithms, ISBN 978-0-471 - 67806-9, John Wiley & Sons, Inc.

Carter M, Laporte G, Lee S (1996). Examination Timetabling: Algorithmic Strategies and Applications. Journal of Operational Research Society, Volume 47, pp 373-383.

David P. (1998). A Constraint-Based Approach for Examination Timetabling Using Local Repair Techniques. In: E.K. Burke and M.W. Carter (eds). Practice and Theory of Automated Timetabling: Selected papers from the 2nd International conference, LNCS, Volume 1408, pp 169-186. Springer-Verlag, Berlin, Heidelberg.

Ersoy E., Ozcan E. and Sima Uyar A. (2007). Memetic Algorithms and Hyperhill-Climbers. Proceedings of the 3nd Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA07, pp 159-166.

Gamma E., Helm R., Johnson R., Vlissides J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. ISBN-13: 978-0201633610.

Glover F (1977). Heuristics for Integer Programming Using Surrogate Constraints. Decision Sciences, Vol 8, No 1, pp. 156-166.

Glover F (1998). A template for scatter search and path relinking. In: Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D (eds). Artificial evolution. Lecture notes in computer science, vol 1363.

Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control Cybern 29(3):653–684.

Glover F, Laguna M, Martí R (2002) Scatter search. In: Ghosh A. Tsutsui S (eds) Theory and applications of evolutionary computation: recent trends. Springer, Berlin, pp 519–529

Glover F, Laguna M, Martí R (2003) Scatter search and path relinking: advances and applications. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic, Dordrecht, pp 1–36

Gogos C, Alefragis P and Housos E (2010). An Improved Multi-Staged Algorithmic Process for the Solution of the Examination Timetabling Problem. Annals of OR, DOI: 10.1007/s10479-010-0712-3.

Gogos C, Goulas G, Alefragis P and Housos E (2009). Pursuit of Better Results for the Examination Timetabling Problem Using Grid Resources, 2009 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched), Nashville, Tennessee, USA, 30 Mar-2 Apr 2009, pp 48-53, DOI:10.1109/SCIS.2009.4927014.

Goulas G, Gogos C, Alefragis P and Housos E (2009). SchedScripter: Workflows for Grid-based Human Resources Scheduling Applications, 4th EGEE User Forum/OGF 25 and OGF Europe's 2nd, Catania, Sicily, Italy, 2-6 March 2009

Gropp W, Lusk E and Skjellum A (1999). Using MPI: Portable Parallel Programming with the Message Passing Interface, 2nd Edition, MIT Press.

Laguna M and Armentano V (2005). Metaheuristic Optimization via Memory and Evolution Tabu Search and Scatter Search. Lessons from applying and experimenting with Scatter Search, pp. 229-246. Operations Research Computer Science Interfaces Series, Vol. 30.

Laguna M. and Marti R. (2006). Metaheuristic Procedures for Training Neural Networks edited by Alba E. and Marti R. Scatter Search. Springer Science+Business Media, LLC.

Mansour N, Isahakian V and Ghalayini I. (2009). Scatter Search Technique for Exam Timetabling. App Intell, DOI 10.1007/s10489-009-0196-5.

McCollum B. (2007). A Perspective on Bridging the Gap between Theory and Practice in University Timetabling. In: PATAT 2006, LNCS 3867, pp 3-23, ISBN 978-3-540-77344-3. Berlin: Springer.

McCollum B., McMullan P., Burke E., Parkes A., Qu R. (2009a). A New Model for Automated Examination Timetabling. Submitted to post PATAT Annals of OR.

McCollum B., Schaerf A., Paechter B., McMullan P., Lewis R., Parkes A., Di Gaspero L., Qu R., Burke E. (2009b). Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS Journal of Computing 2009. DOI:10.1287/ijoc.1090.0320.

Muller T. (2008). ITC 2007: Solver description. Proceedings of the 7th International Conference on Practice and Theory of Automated Timetabling. University of Montreal, Canada.

Pillay N. and Banzhaf W. (2008). A Study of Heuristic Combinations for Hyper-Heuristic Systems for the Uncapacitated Examination Timetabling Problem. European Journal of Operational Research. DOI:10.1016/j.ejor.2008.07.023.

Qu R, Burke E, McCollum B Merlot L and Lee S (2009). A Survey of Search Methodologies and Automated System Development for Examination Timetabling. Journal of scheduling, 12(1), pp 55-89. DOI:10.1007/s10951-008-0077-5.

Qu R. and Burke E.K. (2009). Hybridizations Within a Graph Based Hyper-Heuristic Framework for University Timetabling Problems. Journal of Operational Research Society. DOI:10.1057/jors.2008.102

Schaerf A. (1999). A Survey of Automated Timetabling. Artificial Intelligence Review, Volume 13, pp 87-127. Kluwer Academic Publishers, Netherlands.

Talbi, El-Ghazali(2009). Metaheuristics, from Design to Implementation, ISBN 978-0-470-27858-1, John Wiley & Sons, Inc.

Yu J. and Buyya R. (2005). A Taxonomy of Workflow Management Systems for Grid Computing, Journal of Grid Computing, Volume 3, Numbers 3-4, Pages: 171-200, Springer, New York, USA.

A Comparison of Three Heuristics on a Practical Case of Sub-Daily Staff Scheduling

Maik Günther · Volker Nissen

Abstract Sub-daily personnel planning, which is the focus of our work offers considerable productivity reserves for companies in certain industries, such as logistics, retail and call centers. However, it also creates complex challenges for the planning software. We compare particle swarm optimisation (PSO), the evolution strategy (ES) and a constructive agentbased heuristic on a set of staff scheduling problems derived from a practical case in logistics. All heuristics significantly outperform conventional manual full-day planning, demonstrating the value of sub-daily scheduling heuristics. PSO delivers the best overall results in terms of solution quality and is the method of choice, when CPU-time is not limited. The approach based on artificial agents is competitive with ES and delivers solutions of almost the same quality as PSO, but is vastly quicker. This suggests that agents could be an interesting method for real-time scheduling or re-scheduling tasks.

Keywords personnel planning · sub-daily scheduling · metaheuristics · artificial agents

1 Introduction

Staff scheduling involves the assignment of an appropriate employee to the appropriate workstation at the appropriate time while considering various constraints. This work describes a method for solving the problem of *subdaily* staff scheduling with individual workstations. According to current research employees spend on average 34.3% of their working time unproductively [26]. Major reasons include a lack of planning and controlling. The problem can be faced with demandoriented staff scheduling. Key planning goals are increased productivity, reduction of staff costs, prevention of overtime, better motivation of employees with positive results for sales and service [29].

In practice, the application of a system for staff scheduling has not been very prevalent up to now. Most often planning takes place based on prior experience or with the aid

M. Günther, V. Nissen
Ilmenau University of Technology, Faculty of Economic Sciences
Chair of Information Systems in Services (WI2)
Postfach 100565, D-98684 Ilmenau, Germany
Tel.: +49-3677-694047
E-mail: maik.guenther@gmx.de, volker.nissen@tu-ilmenau.de

of spreadsheets [2]. It is obvious that the afore-mentioned goals of demand-oriented staff scheduling cannot be realised with these planning tools. Even with popular staff planning software employees are regularly scheduled for one workstation per day. However, in many branches, such as logistics and trade, the one-employee-one-station concept does not correspond to the actual requirements and sacrifices potential resources. Intra-day variations in demand require more flexible changes of employees among workstations. This is the only way to prevent or at least reduce phases of over- and understaffing. This issue is critical in our application domain logistics, because on the one hand a high service level is contractually obligated to the customers and on the other hand the level of competition is high and strict cost management is required. Therefore, sub-daily planning should be an integral component of demand-oriented staff scheduling.

It may be argued that the introduction of sub-daily scheduling is likely to generate resistance among the workers. However, in markets with intense competition, such as logistics, a company must use it's opportunities to provide good service at a reduced cost level to secure the long-term competitiveness and survival of the firm. This is ultimatively also in the interest of employees. Moreover, in industries with similar characteristics and requirements, such as retailing and call centers, the concept of sub-daily scheduling is already in use.

In our work, we pursue three intertwined research goals. According to Puppe et al. [27], centralized scheduling approaches are difficult to employ successfully. One of our research goals is, therefore, to investigate whether this is actually true for sub-daily staff scheduling problems. To this end, we develop different variants of centralized scheduling approaches based on modern metaheuristics, namely the evolution strategy (ES) and particle swarm optimisation (PSO).

The ES was chosen, because our earlier experiments on other application problems [22] [21] showed great potential of the ES for combinatorial optimisation, even though this is still a rather neglected field of research and the vast majority of publications on ES deals with real-valued parameter optimisation. PSO was chosen, because our prior work on staff scheduling [23] demonstrated that this metaheuristic can produce very good results on this type of application. We build upon our previous work here by adding a repair heuristic to further improve results.

The metaheuristics are tested and compared on a set of eight problem instances generated from a practical logistics case. Here, a second research goal is to contribute to the comparison of metaheuristics on practical problems of realistic size and complexity. We compare the results to the manually generated plan (as taken from the cooperating company) as well as results from a constructive approach.

Ernst et al. [10] provide a comprehensive overview of problems and solution methods for personnel scheduling and rostering from more than 700 analysed sources. Constructive methods were applied in 133 of those works. Because of it's popularity, this method is included in our comparison. We develop a constructive heuristic approach that builds a single solution by interaction of multiple artificial agents, following other successful agent-based scheduling approaches in the literature [27] [18] [9].

Finally, with our research we aim to contribute to the solution of a practical and non-trivial optimisation problem that is gaining significance in industries such as trade and logistics as well as call centers. We use real-world data sets and the results of our heuristic approaches were conceived very positively by the management of the respective company. However, we have so far not integrated the new solution methods in a commercial software product. It is worth mentioning here that workforce management consists of many facets besides optimization that should be adressed within a holistic commercial software solution.

In the following section, the application problem is described and the mathematical model is given. Then, we discuss work related to our own research before developing approaches based on particle swarm optimisation, the evolution strategy and artificial agents in section 4. The experimental setup and empirical results are presented and discussed in section 5. The paper concludes with a short summary and some indications for future work.

2 Description of the Practical Application Problem

The present problem originates from a German logistics service provider. This company operates in a spatially limited area. The planning problem covers seven days (20 hours each), divided into 15-minute intervals. It includes 65 employees and, thus, an uncompressed total of 36,400 dimensions for the optimisation problem to be solved. The planning task is to find a staff schedule that respects certain hard constraints and minimizes the violation of soft contraints. Nine different workstations need to be filled, with seven having qualification requirements. For real-world data sets and benchmarks see [37].

The problem starts out assuming a set of employees $\mathscr{E} = \{1,\ldots,E\}$, a set of workstations $\mathscr{W} = \{1,\ldots,W\}$ and a discrete timeframe $\mathscr{T}$ with the index $t = 0,\ldots,T-1$, where each period t of the range has a length l_t greater than zero. The demand d_{wt} of employees per workstation and period cannot be negative.

$$\begin{array}{ll} l_t > 0 & \forall t \in \mathscr{T} \\ d_{wt} \geq 0 & \forall w \in \mathscr{W} \text{ and } \forall t \in \mathscr{T} \end{array} \tag{1}$$

The general availability of the employees is known for each interval from the previous full-day planning. Employees are quite flexible in terms of their working hours, which results in a variety of shifts. Shift planning was done for 13 possible shifts plus a planned off-shift. Several considerations are included, such as presence and absence, timesheet balances, qualifications and resting times etc. Therefore the availability of employees is known at the beginning of the sub-daily planning and is determined using the binary variable a_{et}.

$$a_{et} = \begin{cases} 1 & \text{if employee } e \text{ is available at period } t \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The assignment of an employee to a workstation is controlled using the binary variable x_{ewt}.

$$x_{ewt} = \begin{cases} 1 & \text{if } e \text{ is assigned to } w \text{ at } t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

An employee e can only be associated with a workstation w in the period t if he or she is actually present. Additionally, an employee can only be designated to one workstation at a time.

$$\sum_{w=1}^{W} x_{ewt} = a_{et} \qquad \forall e \in \mathscr{E} \text{ and } \forall t \in \mathscr{T} \tag{4}$$

A staff schedule is only valid if any one employee is assigned to one workstation at a time and if absent employees are not included in the plan. These hard constraints can be contrasted with soft constraints, such as the avoidance of understaffing or qualifications. The violation of soft constraints is penalized with error points. The error points used in our work are from an interview with the logistics service provider and reflect that company's requirements. Basically, they reflect a ranking of constraint violations.

Special attention must be paid during the scheduling process to compliance with the required qualifications. The tasks of employees concern loading and unloading, short distance transportation and other logistic services. There are regulations especially with regard to qualifications because the assignment of unqualified employees might lead to significant material damage and personnel injury. The employer regularly invests substantial time and money in qualification measures. Thus, many employees can work at several different workstations. The variety of qualifications was summarised in four qualification groups. Any workstation can require a set of qualifications Q_w, and employees have a set of qualifications Q_e at their disposal. If an employee is planned for a workstation but does not meet all necessary qualifications, error points P_q are generated for the duration of the assignment according to the error point size c_q. The error point size itself is independent of the particular workstation or employee.

$$P_q = \sum_{t=0}^{T-1} \sum_{w=1}^{W} \sum_{e=1}^{E} c_q l_t x_{ewt} \qquad \begin{array}{ll} c_q > 0 & \text{if } e \text{ is not qualified} \\ & \text{for } w, \\ c_q = 0 & \text{else} \end{array} \tag{5}$$

The personnel requirements for each workstation are known in advance and even short-term alterations occur extremely rarely, which yields a high certainty in planning. If a discrepancy arises from the workstation staffing target d_{wt}, error points P_d are generated for the duration and size of the erroneous assignment according to the error point size. Three types of errors can be distinguished: c_{do} represents overstaffing when the demand $d_{wt} > 0$, c_{dn} signals overstaffing when the demand $d_{wt} = 0$, c_{du} signals cases of understaffing. Again, the three error point sizes are not dependent on the particular workstation.

$$P_d = \sum_{t=0}^{T-1} \sum_{w=1}^{W} (c_{dn} + c_{do} + c_{du}) l_t \left| \left(\sum_{e=1}^{E} x_{ewt} \right) - d_{wt} \right|, \tag{6}$$

with:

$c_{dn} > 0$ if w is overstaffed at t and $d_{wt} = 0$, else $c_{dn} = 0$

$c_{do} > 0$ if w is overstaffed at t and $d_{wt} > 0$, else $c_{do} = 0$

$c_{du} > 0$ if w is understaffed at t and $d_{wt} > 0$, else $c_{du} = 0$

To avoid an excessive number r_e of sub-daily workstation (job) rotations for any employee c_r error points arise for such rotations.

$$P_r = c_r \sum_{e=1}^{E} r_e \tag{7}$$

Therefore, the objective function to be minimised becomes:

$$minP = P_q + P_d + P_r. \tag{8}$$

Currently, monthly staff scheduling is carried out manually within MS EXCEL™. The personnel demand for the workstations is subject to significant variations during the day. However, employees are generally scheduled to work at the same workstation all day, causing large phases of over- and understaffing. This lowers the quality of service and the motivation of employees and leads to unnecessary personnel costs as well as downtime. Today, sub-daily workstation rotation is only rarely used in the planning. Usually, department managers intervene directly on site and reassign the employees manually. Obviously, demand-oriented staff scheduling cannot be realised with this approach.

3 Related Work

In [10] Ernst et al. offer a summary of papers related to the issue of staff scheduling – about 700 papers between the years 1954 and 2004 have been included. They identify certain categories of problems, such as the category *flexible demand*. This category is characterised by little available information on schedules and upcoming orders. In our problem a demand per time interval is given as well as a required qualification. Thus, the application problem discussed here can be classified in the group flexible demand schemes. It can additionally be classed under task assignment. *Task assignment* is used to generate assignments requiring certain qualifications and needing to be completed in a certain period of time, which are then distributed amongst the employees. The employees have already been assigned shifts.

As work related to our research Vanden Berghe [33] presents a heuristic to sub-daily planning. Here, demand is marked by sub-daily time periods, which allows the decoupling of staff demand from fixed shifts resulting in fewer idle times. However, scheduling ist not performed at the detailed level of individual workstations as in our research.

In [20] Schaerf and Meisels provide a universal definition of an employee timetabling problem. Both the concepts of shifts and of tasks are included, whereby a shift may include several tasks. Employees are assigned to the shifts and assume tasks for which they are qualified. Since the task is valid for the duration of a complete shift, no sub-daily changes of tasks (or rather workstations) are made. Blöchlinger [6] introduces timetabling blocks (TTBs) with individual lengths. In this model employees may be assigned to several sequential TTBs, by which subdaily time intervals could be represented within a shift. Blöchlinger's work also considers tasks; however, a task is always fixed to a TTB. Essentially, our problem of the logistics service provider represents a combination of [20] (assignment of staff to tasks) and [6] (sub-daily time intervals), but with the assignment periods (shifts) of the employees already being set.

Staff scheduling is a hard optimisation problem. In [12] Garey and Johnson demonstrate that even simple versions of staff scheduling problems are NP-complete. Kragelund and Kabel [17] show the NP-hardness of the general employee timetabling problem. Moreover, Tien and Kamiyama prove in [32] that practical personnel scheduling problems are generally more complex than the TSP which is itself NP-hard. Thus, heuristic approaches appear justified for our application.

Apparently, there exists no off-the-shelf solution approach to the kind of detailed sub-daily staff planning problem considered here. Approaches based on particle swarm optimisation (PSO), the evolution strategy (ES), and multiple artificial agents for this application are outlined in the following section.

We now give a short general overview of particle swarm optimisation and the evolution strategy. For more details, the reader ist referred to [16] [11] for standard-PSO and [4] [5] for standard-ES. Thereafter, some research of others on agent-based scheduling is outlined to complete the picture of related work.

The basic principles of PSO were developed by Kennedy and Eberhart among others [15] [16]. Swarm members are assumed to be massless, collision-free particles that search for optima with the aid of a fitness function within a solution space. In this process each single particle together with its position embodies a solution to the problem [34]. While looking for the optimum, a particle does not simply orient itself using its own experience but also using the experience of its neighbours [11]. This means that the particles exchange information, which can then positively influence the development of the population in the social system as a whole [24].

Modifications of standard real-valued PSO exist for binary variables, where the speed of a particle is used as the probability for the change of the binary value [16]. This approach, however, has several limitations and was changed from binary to decimal variables in [35]. Another PSO-variant was developed for sequence planning tasks [31]. In 2007 Poli analysed the IEEE Xplore database for the thematic grouping of PSO applications [25]. Of approximately 1100 publications only one work is focused specifically on timetabling [8] which is related to our own application problem. In [8], the authors adjust PSO to the combinatorial domain. No longer is the position of a particle determined by its speed but rather by using permutation operators. In [7] university timetabling was also approached with PSO.

The evolution strategy (ES) was originally invented by Rechenberg and Schwefel [5] and soon applied to continuous parameter optimisation problems. Like genetic algorithms the evolution strategy belongs to the class of evolutionary algorithms that form broadly applicable metaheuristics, based on an abstraction of the processes of natural evolution [3] [4]. There is some work on the evolution strategy in combinatorial and discrete optimisation. Herdy [14] investigates discrete problems with some focus on neighbourhood sizes during mutation. Rudolph [28] develops an evolution strategy for integer programming by constructing a mutation distribution that fits this particular search space. Bäck [3] discusses mutation realized by random bit-flips in binary search spaces. Nissen [21] modifies the coding and the mutation operator of the evolution strategy to solve combinatorial quadratic assignment problems. Schindler et al. [30] apply the evolution strategy to combinatorial tree problems by using a random key representation which represents trees with real numbers. Schwefel and Beyer [5] present permutation operators for the evolution strategy in combinatorial ordering problems. Li et al. [19] develop a mixed-integer variant of the evolution strategy that can optimize different types of decision variables, including continuous, normal discrete, and ordinal discrete values. Nissen and Gold [22] propose an evolution strategy for a combinatorial network design problem that successfully utilises a repair heuristic and domain-specific mutations. However, continuous parameter optimisation is still the dominant field of application for the evolution strategy, as the operators of the standard form are particularly well adapted to this type of problem.

Next to these metaheuristics, there is also some related work on agent-based scheduling. Puppe et al. [27] present two concepts for artificial agents on scheduling in hospitals. In the resource-oriented view each resource or the associated organizational unit is represented as an agent. This concept is more applicable, when the problem is static. In the patient-oriented view, an agent is created for every patient examination, which is more adapted to dynamical problems.

Krempels [18] also creates a staff schedule by using agents. The agent approach is divided in several phases. Initially a planner agent creates a plan ignoring staff preferences. Thereafter, the planner tries to improve the plan by incorporating preferences. A knowledge tank stores all relevant aspects of the resources. In case of a conflict, an agent is created for each staff member, followed by a negotiation phase.

De Causmaecker et al. [9] make comments on negotiation schemes for course timetabling. Only necessary information should be exchanged among agents. Moreover, a negotiation process should not take exceedingly long. More recent agent-based approaches for scheduling are, for instance, presented in [1] and [36].

4 PSO Approach and Evolution Strategy

4.1 Problem Representation

To apply PSO and the evolution strategy, the sub-daily staff scheduling problem needs to be conveniently represented. A two-dimensional matrix is applied. Each particle in the swarm (for PSO) has an own matrix that determines its position. Also, each individual in the ES-population uses a matrix to represent its solution to the application problem. The rows of the matrix signify employees and the columns signify each time period of the length $l_t > 0$. To mark times in which an employee is not present due to his work-time model, a dummy workstation is introduced (in Table 1: workstation 0). For example, employee two is absent in the first period and then is assigned to workstation 2. Assignment changes can only be made to non-dummy workstations, so that no absent employee is included.

To lower the complexity the number of dimensions should be reduced. This can be realised via a suitable depiction of time. Within the planned day, time is viewed with a time-discrete model. An event point (at which a new time interval begins) occurs when the allocation requirement for one or more workstations or employee availability change. With this method, however, the periods are not equally long any more, so that their lengths need to be stored.

Table 1 Assignment of workstations in a matrix.

employee	**period**						
	1	2	3	4	5	6	...
1	1	1	1	1	1	1	
2	0	2	2	2	2	2	
3	0	1	1	2	2	2	
4	0	6	6	6	6	2	
5	3	3	2	2	0	0	
...							

4.2 Repair Heuristic

Both scheduling metaheuristics outlined in this paper employ an identical repair heuristic to reduce the total error points of a solution. This repair heuristic corrects constraint violations in the following order based on error point size:

- qualification: employees not qualified for the currently assigned workstation are given an apropriate assignment whilst ignoring under- or overstaffing
- no demand: employees currently assigned to a workstation with zero demand are given a different assignment (if possible) whilst simultaneously considering their qualification
- understaffing: if workstations are understaffed employees are reassigned from other workstations with overstaffing (if possible) also considering their qualification. Thus, simultaneously the problem of overstaffing is reduced.

4.3 PSO for this Application

The following pseudocode presents an overview of the implemented PSO. Here, pBest represents the best position found so far by the particle while gBest corresponds to the best position of all particles globally.

```
01: initialise the swarm
02: evaluate the particles of the swarm
03: determine pBest for each particle and gBest
04: loop
05:   for i = 1 to number of particles
06:     calculate new position // use the 4 alternative actions
07:     repair the particle
08:     evaluate the particle
09:     if f(new position)<f(pBest) then pBest=new position // new pBest
10:     if f(pBest)<f(gBest) then gBest=pBest // new gBest
11:   next i
12: until termination
```

At the start of PSO the initialisation of the particle position creates valid assignments w.r.t. the hard constraints by using information from the companys current full-day staff schedule. Therefore, valuable prior knowledge is not wasted. Based on this plan, improved solutions can now be determined that include plausible workstation changes.

In each iteration the new particle position is determined by traversing all dimensions of the particle and executing one of the following actions with predefined probability. The probability distribution was heuristically determined in prior tests. The behaviour of the PSO-heuristic is relatively insensitive to changes of p_1, p_3, and p_4. The optimal value for p_2 depends on the problem size (smaller probabilities for larger problems).

- No change (p_1=9.7%): The workstation already assigned remains.
- Random workstation (p_2=0.3%): A workstation is randomly determined and assigned. Only those assignments are made, for which the employee is qualified. The probability function is uniformly distributed.
- pBest workstation (p_3=30%): The corresponding workstation is assigned to the particle dimension from pBest. Through this, the individual PSO component is taken into account.
- gBest workstation (p_4=60%): The corresponding workstation is assigned to the particle dimension from gBest. gBest was found to work best as a neighbourhood topology for this type of application in [13]. By considering the best position of all particles, the swarm's experience is included in the position calculation.

Once created, a solution is repaired with the heuristic described above before it undergoes evaluation.

The characteristics of PSO have not been changed with these modifications. There are merely changes in the way to determine a new particle position, so that the calculation of the velocity is not needed. The current form of position determination makes it unnecessary to deal with dimension overruns. All other peculiarities of PSO regarding social or global behaviour remain. Even all neighbourhood topologies established as part of continuous parameter optimisation in standard-PSO remain and can be used without restrictions. In our implementation, PSO terminates after 400,000 inspected solutions. Alternatively, convergencebased termination criteria could be employed.

4.4 Evolution Strategy for this Application

The following pseudocode presents an overview of the implemented ES.

01: initialise the population with μ individuals
02: repair the μ individuals
03: evaluate the μ individuals
04: loop
05: copy and recombine parents to generate λ offspring
06: mutate the λ offspring
07: repair the λ offspring
08: evaluate the λ offspring
09: select $((\mu+\lambda)$ or $(\mu,\lambda))$ μ best individuals as new generation
10: until termination

The ES population is initialized with valid solutions w.r.t the hard problem constraints. Again, information from the company's current full-day staff schedule is used. We use the same initialisation as for PSO. (μ,λ)-selection (comma-selection) as well as $(\mu+\lambda)$-selection (plus-selection) are used as well as different population sizes. In plus-selection parents compete with their offspring and can, thus, survive to the next generation cycle. By contrast, comma-selection assumes that only offspring compete during the selection process. The best solution found during an experimental run is always stored and updated in a "golden cage". It represents the final solution of the run. Following suggestions in the literature [4] [5], the ratio μ/λ is set to 1/5 – 1/7 during the practical experiments.

Ten alternative recombination variants were evaluated in a pre-test. The best performance was achieved with a rather simple form that is based on the classical one-point crossover. The recombination of parents to create an offspring solution works as follows: A common crossover point is determined at random for all employees (rows) of a solution and the associated parts of the parents are exchanged (see fig. 1).

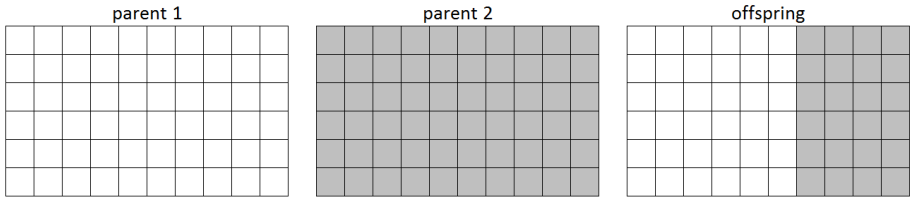


Fig. 1 Recombination operator employed.

Mutation is the main search operator employed in ES. In standard-ES mutation is performed using normally-distributed random variables so that small changes in a solution are more frequent than large changes. In [23] we developed a search operator that adheres quite closely to this classical form of mutation and produced fairly good results.

In this paper, a different approach to mutation is employed that takes the characteristics of the discrete search space better into account. It is based on the work of Rudolph [28]. He developed an approach to construct a mutation distribution for unbounded integer search spaces. The concept of maximum entropy is used to select a specific distribution from

numerous potential candidates. Rudolph tested his ideas on five nonlinear integer problems. Some adaptations were required for the staff scheduling problem, though. The search space in our problem domain is bounded and hard constraints must be considered. In short, the main differences to Rudolph's approach are as follows:

- dimension boundaries are introduced to account for the bounded search space
- mutation produces only changes that consider employee availability
- the assignment of workstations during mutation respects necessary qualifications
- the mutation intensity is increased to account for the high-dimensional search space

Before a solution is evaluated it is repaired using the same repair heuristic as was the case for PSO. The ES terminates when 400,000 solutions have been inspected to allow for a fair comparison with PSO.

4.5 Artificial Agents for this Application

The two metaheuristic approaches that are based on searching the solution space are contrasted with a constructive method that is based on a multitude of interacting artificial agents. Following the suggestion of Puppe et al. [27], resource-oriented agents are used for this static staff scheduling application. In our problem, constraints and preferences come from two directions. On one side is the employer who aims at reduced overall costs, a high service level, the consideration of qualifications in the schedule etc. On the other side there are the employees, that try to enforce their rights, such as legal regulations and the minimization of workstation rotations during the day. Consequently, following Krempels [18], our approach is structured in two phases associated with employer and employees.

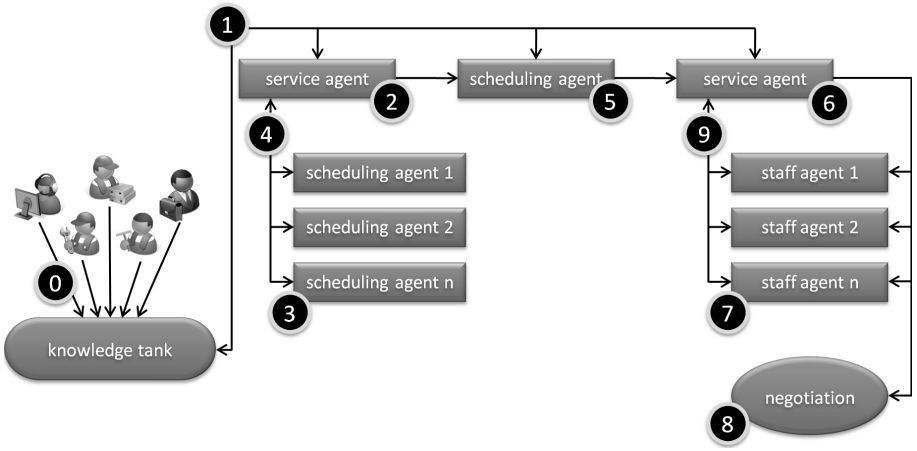


Fig. 2 Representation of the agent approach for the logistics problem.

Fig. 2 shows a schematic representation of our multi-agent approach, which also respects the recommendations in [9]. The individual steps, that finally construct a staff schedule, can be described as follows:

- First, the properties of existing resources, current demands and conditions of the problem space are stored in the knowledge tank (0). This information includes absence of

employees, required qualifications, error-point values for violations of restrictions, personnel requirements of each interval etc.

- The information in the knowledge tank is supplied (1) to three agents (2), (5) and (6).
- Before starting to plan, service agent (2) initialises the schedule by assigning all employees to a dummy workstation. This indicates, that these employees are not currently assigned to an actual workstation.
- Following that, service agent (2) ranks the nine workstations, with the highest priority going to workstations for which the least number of employees are qualified. Should the number of qualified staff for two or more workstations be identical, then the priorities are ordered at random.
- Scheduling agents (3) are sequentially initialised by the service agent (2), according to priority. Each scheduling agent (3) represents one of the nine workstations. Only one scheduling agent (3) exists at any time. The scheduling agent for which the fewest employees are qualified begins. He schedules qualified employees, who are present and have not yet been assigned. Over- and understaffing should be minimised as much as possible. The planning result of the first scheduling agent is passed (4) to service agent (2), which in turn gives feedback regarding the schedule to the knowledge tank (1). Then, service agent (2) initiates the next scheduling agent (3), which also attempts to cover its personnel demand as good as possible. During this process, previously assigned employees may not be deployed to subsequent workstations. Service agent (2) sequentially initiates scheduling agents (3) until all nine workstations have been processed.
- After an assignment plan was created, there could still be employees in some timeslots, who have not yet received an assignment. Switching employees to other workstations could result in better coverage of demand. The service agent (2) calls a scheduling agent (5), also connected (1) to the knowledge tank. Scheduling agent (5) finalises the schedule by deploying all workers, who are still unassigned, necessarily accepting overstaffing. Possible switches are again checked as to whether they would lead to better demand coverage and those that would are carried out.
- Assignment planning was done up to now from the point of view of the company. This occurred while neglecting employee needs – the reduction of the number of workstation rotations. For this reason, scheduling agent (5) initiates another service agent (6) in order to consider employee preferences. This service agent (6) is also connected to the knowledge tank (1).
- Service agent (6) examines each timeslot in the schedule and checks whether a workstation rotation occurs. If this is the case, all workers are identified for whom a negotiation could occur for this timeslot. They must be present in the timeslot and qualified for the switch. Service agent (6) simultaneously generates a staff agent (7) for each relevant employee. In contrast to the scheduling agents (3), more than one staff agent exists at the same time.
- Two staff agents (7) negotiate a workstation assignment switch (8) in the following way: The staff agent where the service agent (6) identified a workstation rotation sequentially asks the other staff agents for a swap. Each staff agent knows its current workstation assignment at times t, $t-1$ und $t+1$. The two communicating staff agents exchange only information about their assignments at time t. Without re-calculating the whole fitness function they can now decide, if a swap would reduce the overall error count of the schedule. If this is the case, they agree to swap and communicate (9) this to the service agent (6). Then, the swap is executed and all staff agents are deleted. If a swap would not reduce the error count, the process continues by asking the next staff agent in the queue. As a result, a swap may or may not occur for each staff agent where a

workstation rotation was identified, depending on the availability of a swap option that improves the overall error count of the schedule.
- In addition to the negotiation (8) between staff agents (7), a negotiation is also carried out between the service agent (6) and the staff agent, for which the workstation rotation was identified. The goal of this negotiation is not to execute a switch with another staff agent, but rather to carry out a switch at time t for the workstation at which the employee is working at times $t-1$ or $t+1$. This also helps reduce the number of workstation rotations. Service agent (6) only agrees to the switch, if the overall quality of staff assignments does not deteriorate. The result of the negotiation is either the assignment to a different workstation at time t (and thus the reduction of workstation rotations) or keeping the assignment as is. This decision is reported to service agent (6) and carried out.
- Service agent (6) repeats the last three steps up to the point where no further improvements occur.

5 Results and Discussion

The full-day manual staff schedule for the logistics service provider problem without sub-daily workstation changes results in 411,330 error points after an evaluation that included the penalties arising from the afore-mentioned constraints.

Table 2 Comparison (error points) of the different sub-daily scheduling heuristics, based on 30 independent runs each. Best results are bold and underlined.

heuristic	error		number of job changes	wrong qualifications in minutes	understaffing in minutes	overstaffing in minutes	
	mean	min				demand >0	demand =0
Manual Plan	411330	411330	0.0	1545.0	20130.0	14610.0	33795.0
Agents	51829	51801	1579.0	0.0	7365.0	28395.0	7245.0
PSO (10)	**51752**	**51736**	1502.2	0.0	7365.0	28395.0	7245.0
PSO (20)	51781	51763	1531.4	0.0	7365.0	28395.0	7245.0
PSO (100)	51826	51811	1575.8	0.0	7365.0	28395.0	7245.0
PSO (200)	51841	51817	1591.2	0.0	7365.0	28395.0	7245.0
ES (10,50)	51870	51842	1620.3	0.0	7365.0	28395.0	7245.0
ES (10+50)	51843	51816	1592.5	0.0	7365.0	28395.0	7245.0
ES (30,200)	51855	51835	1604.5	0.0	7365.0	28395.0	7245.0
ES (30+200)	51846	51820	1596.3	0.0	7365.0	28395.0	7245.0

Table 3 t-test results for pairwise comparison of heuristics.

H_1	T	df	significance H_0 (l-tailed)	mean difference	95% confidence intervall of differences	
					lower	upper
PSO(10) < ES(10+50)	-26.40	58.00	< 0.001	-90.67	-97.54	-83.79
PSO(10) < Agents	-23.57	47.17	< 0.001	-77.27	-83.86	-70.67
Agents < ES(10+50)	-3.26	58	$= 0.002$	-13.40	-21.64	-5.16

The results of the various scheduling approaches are shown in table 2. Thirty independent runs were conducted each time for each of the experiments to allow for statistical

testing. All test runs were conducted on a PC with an Intel 4 x 2.67 GHz processor and 4 GB of RAM. An individual run with the multi-agent approach takes approx. 1 sec of CPU-time. The runtime requirements for the PSO and ES approaches are much higher and in the order of 50 minutes per run (including the repair heuristic). This effort, however, is acceptable as there is sufficient time available for creating the schedule. Moreover, the required CPU-time could certainly be reduced, for instance through parallelization, but this was not the focus of our work.

All heuristics for sub-daily staff scheduling significantly outperform the manual full-day schedule in terms of total error points. This demonstrates the value of sub-daily scheduling as compared to today's standard staff scheduling approach which not only wastes resources but also demotivates personnel and deteriorates quality of service. Generally, the problems of understaffing and overstaffing for periods without demand are greatly reduced. On the other hand, all heuristics lead to more overstaffing in periods with $demand > 0$ as compared to the initial plan. This approach, however, is sensible because employees can still support each other instead of being idle when $demand = 0$.

The fact that all heuristics arrive at the same value for 'wrong qualifications', 'understaffing' and 'overstaffing in minutes' should be interpreted cautiously. PSO and ES find these values due to the use of the repair heuristic, which includes domain-specific knowledge, just as the agent approach is tailored to the problem at hand. Finding these values is, thus, not really an easy task.

However, the true complexity of the application lies in the additional requirement to also reduce the number of job rotations to a minimum. Each job rotation is punished with only one error point, according to the companies ranking of constraint violations as inquired through interviews with the management. Thus, the total error counts of individual schedules by different solution approaches are often quite close. But a schedule that includes many absurd job rotations will not have acceptance of the planners and the employees. Thus, even relatively small differences in the overall error count of distinct plans can be quite meaningful in practice.

Interestingly, the PSO heuristic provides the best results with a rather small swarm size of 10 particles, but also larger swarm sizes produce good results. Many steps are required to arrive at a good schedule. Thus, it seems preferable to track changes for more iterations as compared to richer knowledge (through larger swarm size) of the solution space in each iteration. This effect is less clear for the ES with solution repair. It was visible for the ES, though, when no solution repair was employed as in [23].

Apparently, the plus-selection has a slight advantage over the comma-selection for the ES on this problem instance, but this should not be generalized. The mutation scheme based on maximum entropy provides better results for the ES than a more traditional approach based on rounded Gaussian mutations as given in [23]. This result underlines the importance of adapting the mutation operator to fit the characteristics of the search space as well as possible.

PSO(10) and ES(10+50) provided the best mean error results in their respective groups. With 30 independent runs for each heuristic it is possible to test the statistical significance of the performance difference between both solution methods with a t-test (see table 3). A Levene-test revealed the homogeniety of variances (test level 5%) between both groups $(F = 3.55, p = 0.065)$. The corresponding t-test with a 95% confidence interval confirms the better performance of PSO(10) with a very high statistical significance ($p < 0.001$ for H_0). The result remains the same, if heterogeniety of variances is assumed. This success of PSO must be attributed to its operators since the coding of PSO and ES are identical. A second

reason concerns the fewer strategy parameters in our PSO-approach which are more easily adapted to the application domain.

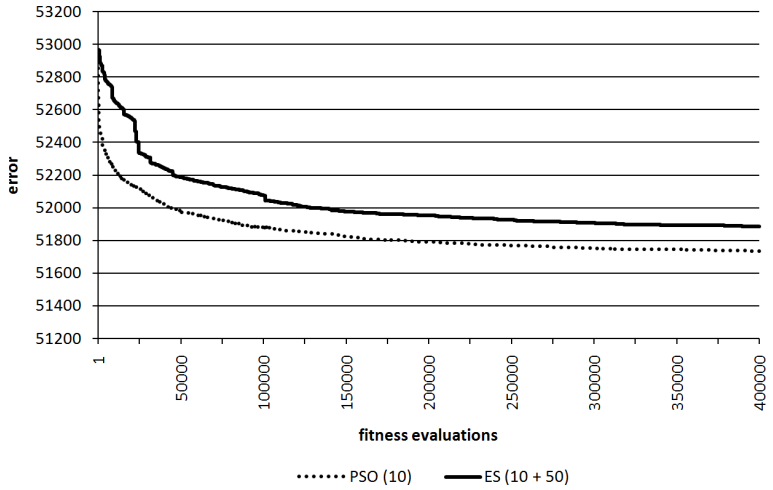


Fig. 3 Convergence chart for PSO(10) and ES(10+50).

Fig. 3 shows the convergence behaviour of best variants from PSO and ES in comparison. Not only does PSO generate the better final solution, but it also demonstrates a more rapid convergence towards good solutions. This is generally a desirable characteristic, particularly when the available time for an optimisation is rather limited (not the case here).

Notwithstanding the fact that the PSO heuristic was able to provide better results for this problem, it also has one technical advantage over ES. The PSO outlined in this paper only requires the varying of two parameters (swarm size and p2), which can both be easily set. ES, on the other hand, offers more parameterisation possibilities (selection pressure, recombination scheme, plus or comma selection etc.), resulting in greater heuristic complexity from a user's perspective.

The results from the multi-agent system are quite close to the schedules created by PSO and in fact better than those generated by ES. A t-test was conducted to compare the best parameterisation of PSO (swarm size 10) and the multi-agent approach (see Table 3). A Levene-test revealed the heterogeneity of variances (test level 5%) between both groups ($F = 6.585, p = 0.013$). The t-test with a 95% confidence interval confirms the better performance of PSO (10) with a very high statistical significance ($p < 0.001$ for H_0).

A corresponding t-test was conducted between the best parametrisation of ES (10+50) and the agent approach. The Levene-test here showed a homogeneity of variances (test level 5%) between both groups ($F = 0.049, p = 0.826$). The t-test confirms the better performance of the multiagent approach with a very high statistical significance ($p = 0.002$ for H_0).

An advantage of the multi-agent approach over the metaheuristics, alongside the low CPU-requirements, is the relative simplicity of its scheduling strategy. While it is hard for a staff planner to grasp what is really happening during optimisation with PSO or ES, the acceptance for the agent-derived solution is likely to be far higher, since the individual steps of the planning and negotiation procedure are relatively straightforward and familiar for

staff managers. The importance of this comprehensibility for the acceptance of the resulting schedule should not be underestimated.

The agent approach does not violate qualification constraints and over- as well as understaffing are reduced to the possible minimum as found by PSO and ES. It is only the number of sub-daily workstation rotations that is different in the solutions produced by the constructive multi-agent method. To achieve an improved solution quality, an extended rescheduling and swapping of assignments would have been required. It must consider more than two staff members in parallel as well as large parts of the planning horizon. This is beyond what is possible through one-to-one negotiation of a staff agent with the service agent or other staff agents. It can only be achieved with the aid of a central planning instance, that partly ignores the individual preferences of agents for a better overall result of the entire schedule. Such a central planning instance, however, is not in line with the distributed negotiation and decision scheme that is generally associated with multi-agent systems.

The different solutions approaches were also tested on the smaller problem sets, representing the individual days of the week. Table 4 shows the respective mean errors (based again on 30 runs) for each day. The relative performance is similar to the more complex week problem discussed before, supporting our previous conclusions.

Table 4 Mean results for individual days of the week problem (30 runs each). Best results are bold and underlined.

	Mo	Tu	We	Th	Fr	Sa	So
Manual Plan	85185	178260	91140	15465	11850	14850	14580
Agents	7727	5917	8183	8272	5528	8861	7337
PSO (10)	**7712**	**5900**	**8161**	**8248**	**5500**	**8838**	7330
PSO (20)	7726	5910	8171	8257	5508	8846	**7325**
PSO (100)	7725	5909	8170	8255	5508	8844	**7325**
ES (10,50)	7726	5910	8171	8257	5508	8846	**7325**
ES (10+50)	7725	5909	8170	8255	5508	8844	**7325**
ES (30,200)	7736	5918	8184	8262	5515	8857	7336
ES (30+200)	7737	5919	8183	8263	5514	8857	7335

6 Conclusion and Future Work

Sub-daily staff scheduling is a meaningful practical problem area. Using a complex, high-dimensional and highly constrained practical planning scenario from logistics, it was demonstrated that sub-daily planning with modern heuristic approaches produces far better results than traditional spreadsheet-based full day scheduling. Thus, sub-daily scheduling significantly increases the value contributions of individual staff members. This success of PSO and ES also contradicts Puppe et al. [27] who suggest that, in contrast to distributed multi-agent systems, centralized scheduling methods (in our case metaheuristics with a centrally supervised optimisation approach) are likely to fail due to the many constraints and complexity of the task.

Because PSO and ES in their traditional forms are not suitable for the planning problem at hand, the metaheuristics were adapted to the combinatorial domain without sacrificing their basic mechanisms. PSO outperformed different variants of the evolution strategy on this problem. The superior performance must be attributed to the operators and parameters of PSO since the coding of PSO and ES are identical.

A constructive heuristic, based on interacting agents, performed competitively with the ES and only slightly inferior to PSO. Based purely on solution quality, PSO should be favored when runtime is not a seriously limiting factor for optimisation. In our practical application this is the case.

The agent approach is vastly quicker in finding good solutions. This result suggests that artificial agents could be useful for real-time scheduling or re-scheduling tasks where runtime for the optimisation is usually very limited. This conclusion is in line with findings of other authors in the literature, such as in [1] for crew rescheduling.

Our agent approach also has benefits in terms of user-acceptance, since the generation of planning results is here more comprehensible than with metaheuristics. In future work we will take a closer look at other recent agent-based approaches for related problems such as the ones presented in [1] and [36].

In addition to the results presented in this paper, we have also experimented with Tabu Search (TS) as a well-known and not population-based local search metaheuristic. Although TS was tailored to the problem at hand, the results achieved so far are not competitive with the three heuristics outlined here. Moreover, TS appears to be strongly influenced by characteristics of the initial solution. Additionally, TS displays slower convergence and, thus, uses more fitness evaluations than PSO and ES to arrive at results of reasonable quality.

In our current research, similar scheduling problems from the domain of trade are investigated. Here, a particularly flexible form of demand-oriented personnel planning is gaining significance. The assumption of fixed shifts is given up and automatic generation of workingtime models and staff scheduling are done in parallel, which further increases the complexity of the task. The results achieved so far confirm the value of sub-daily staff scheduling. Moreover, metaheuristics demonstrate a more robust performance than a constructive approach when the application problem is slightly varied.

References

1. Abbink E.J.W., Mobach D.G.A., Fioole, P.J., Kroon, L.G., v.d.Heijden E.H.T., Wijngaards N.J.E.: Actor-Agent Application for Train Driver Rescheduling. In: Proceedings of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems - Vol. 1, 513–520 (2009)
2. ATOSS Software AG, FH Heidelberg (eds.): Standort Deutschland 2006. Zukunftssicherung durch intelligentes Personalmanagement, München (2006)
3. Bäck T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)
4. Bäck T., Fogel D.B., Michalewicz Z. (eds.): Handbook of Evolutionary Computation. Institute of Physics Publishing (1997)
5. Beyer H.G., Schwefel H.P.: Evolution strategies: a comprehensive introduction. In: Natural Computing 1: 3–52 (2002)
6. Blöchlinger I.: Modeling Staff Scheduling Problems. A Tutorial. In: European Journal of Operational Research 158: 533–542 (2004)
7. Brodersen O.B.: Eignung schwarmintelligenter Verfahren für die betriebliche Entscheidungsunterstützung. Cuvillier (2008)
8. Chu S.C., Chen Y.T., Ho J.H.: Timetable Scheduling Using Particle Swarm Optimization. In: Proceedings of the International Conference on Innovative Computing. Information and Control (ICICIC Beijing 2006) Vol. 3: 324–327 (2006)
9. De Causmaecker P., Ouelhadj D., Vanden Berghe G.: Agents in Timetabling Problems. In: Proc. of MISTA 2003, 67–71 (2003)
10. Ernst A.T., Jiang H., Krishnamoorthy M., Owens B., Sier D.: An Annotated Bibliography of Personnel Scheduling and Rostering. In: Annals of OR 127: 21–144 (2002)
11. Fukuyama Y.: Fundamentals of Particle Swarm Optimization Techniques. In: Lee K.Y., El-Sharkawi M.A. (eds.): Modern Heuristic Optimization Techniques with Applications to Power Systems, Wiley-IEEE Press, 24–51 (2003)

12. Garey M.R., Johnson D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness, Freeman (1979)
13. Günther M., Nissen V.: A Comparison of Neighbourhood Topologies for Staff Scheduling With Particle Swarm Optimisation. In: Mertsching B. et al. (eds.): Proc. of KI 2009, LNAI 5803, Springer, 185–192 (2009)
14. Herdy M.: Application of the 'Evolutionsstrategie' to Discrete Optimization Problems. In: Schwefel H.P., Männer R. (eds): Parallel Problem Solving from Nature, Springer, 188–192 (1990)
15. Kennedy J., Eberhart R.C.: Particle Swarm Optimization. In: Proc. of the IEEE Int. Conf. on Neural Networks, IEEE, 1942–1948 (1995)
16. Kennedy J., Eberhart R.C., Shi Y.: Swarm Intelligence, Kaufmann (2001)
17. Kragelund L., Kabel T.: Employee Timetabling. An Empirical Study, Master's Thesis, Department of Computer Science, University of Aarhus, Denmark (1998)
18. Krempels, K.H.: Lösen von Scheduling-Konflikten durch Verhandlungen zwischen Agenten. In: Sauer J. (ed.): Proc. of PuK 2002, 86–89 (2002)
19. Li R., Emmerich M.T.M., Bovenkamp E.G.P., Eggermont J., Bäck T., Dijkstra J., Reiber J.H.C.: Mixed Integer Evolution Strategies and Their Application to Intravascular Ultrasound Image Analysis. In: Rothlauf F. (ed.): Applications of Evolutionary Computation, LNCS 3907, Springer, 415–426 (2006)
20. Meisels A., Schaerf A.: Modelling and Solving Employee Timetabling. In: Annals of Mathematics and Artificial Intelligence 39: 41–59 (2003)
21. Nissen V.: Solving the Quadratic Assignment Problem with Clues from Nature. In: IEEE Transactions on Neural Networks 5 (1): 66–72 (1994)
22. Nissen V., Gold S.: Survivable Network Design with an Evolution Strategy. In: Yang A., Shan Y., Bui L.T. (eds.): Success in Evolutionary Computation. Studies in Computational Intelligence, Springer, 263–283 (2008)
23. Nissen V., Günther M.: Staff Scheduling with Particle Swarm Optimization and Evolution Strategies. In: Cotta C., Cowling P. (eds.): EvoCOP, LNCS 5482, Springer, 228–239 (2009)
24. Parsopoulos K.E., Vrahatis M.N.: Recent Approaches to Global Optimization Problems through Particle Swarm Optimization. In: Nat. Comp. 1: 235–306 (2002)
25. Poli R.: An Analysis of Publications on Particle Swarm Optimization, Report CSM-469, Dep. of Computer Science, University of Essex, England (2007)
26. Proudfoot Consulting: Global Productivity Report, Atlanta (2008)
27. Puppe F., Klügl F., Herrler R., Kirn S., Heine C.: Konzeption einer flexiblen Agentenkomponente für Schedulingaufgaben im Krankenhausumfeld. In: Proc. of 2. Koll. "Intelligente Softwareagenten und betriebswirtschaftliche Anwendungsszenarien" (2000)
28. Rudolph G.: An Evolutionary Algorithm for Integer Programming. In: Davidor Y., Schwefel H.P., Männer R. (eds.): PPSN III, LNCS 866, Springer, 139–148 (1994)
29. Scherf B.: Wirtschaftliche Nutzenaspekte der Personaleinsatzplanung. In: Fank M., Scherf B. (eds.): Handbuch Personaleinsatzplanung, Datakontext, 55–83 (2005)
30. Schindler B., Rothlauf F., Pesch E.: Evolution strategies, Network Random Keys, and the One-Max Tree Problem. In: Applications of Evolutionary Computing: EvoWorkshops 2002, LNCS 2279, Springer, 29–40 (2002)
31. Tasgetiren M.F., Sevkli M., Liang Y.C., Gencyilmaz G.: Particle Swarm Optimization Algorithm for Single Machine total Weighted Tardiness Problem. In: Proceedings of the CEC 2004. IEEE, 1412–1419 (2004)
32. Tien J., Kamiyama A.: On Manpower Scheduling Algorithms. In: SIAM Rev. 24 (3): 275–287 (1982)
33. Vanden Berghe G.: An Advanced Model and Novel Meta-heuristic Solution Methods to Personnel Scheduling in Healthcare, Thesis, University of Gent (2002)
34. Veeramachaneni K.: Optimization Using Particle Swarm with Near Neighbor Interactions. In: GECCO-2003, LNCS 2723, Springer, 110–121 (2003)
35. Veeramachaneni K., Osadciw L., Kamath G.: Probabilistically Driven Particle Swarms for Optimization of Multi-valued Discrete Problems: Design and Analysis. In: Proceedings of the IEEE SIS 2007, Honolulu, 141–149 (2007)
36. Wauters T, Verbeeck K, Vanden Berghe G., de Causmaecker P.: A Multi-Agent Learning Approach for the Multi-Mode Resource-Constrained Project Scheduling Problem. Paper presented at the 2nd Int. Workshop on Optimisation in Multi-Agent Systems (OptMas), Budapest AAMAS (2009)
37. Sub-Daily Staff Scheduling Data Sets and Benchmarks, http://www.tu-ilmenau.de/fakww/2608+M54099f70862.0.html

The Bi-Objective Master Physician Scheduling Problem

Aldy Gunawan • Hoong Chuin Lau

Abstract Physician scheduling is the assignment of physicians to perform different duties in the hospital timetable. In this paper, the goals are to satisfy as many physicians' preferences and duty requirements as possible while ensuring optimum usage of available resources. We present a mathematical programming model to represent the problem as a bi-objective optimization problem. Three different methods based on ε–Constraint Method, Weighted-Sum Method and Hill-Climbing algorithm are proposed. These methods were tested on a real case from the Surgery Department of a large local government hospital, as well as on randomly generated problem instances. The strengths and weaknesses of the proposed methods are also discussed. Finally, a summary is given together with suggestions for future research.

Keywords: *master physician scheduling problem, preferences, bi-objective optimization, mathematical programming.*

1 Introduction

Personnel scheduling is defined as the process of constructing optimized work schedules for staff (Topaloglu, 2009). A literature review of applications, models and algorithms in personnel scheduling has been provided by Ernst et al. (2004). The personnel scheduling problem includes a wide variety of applications such as airlines, railways, manufacturing and health care systems. In this paper, the scheduling of physicians in a hospital is addressed.

Brandeau et al. (2004) provided a more recent collection of Operations Research applications in health care, with particular emphasis on health care delivery. To our knowledge, research on physician scheduling focuses primarily on a single type of duty, such as the emergency room (e.g. Vassilacopoulos, 1985; Beaulieu et al., 2000; Carter and Lapierre, 2001; Gendreau et al., 2007; Puente, et al., 2009), the operating room (e.g. Testi et al., 2007; Burke and Riise, 2008; Beliën et al., 2009; Roland et al., 2009), the physiotherapy and rehabilitation services (Ogulata et al., 2008).

In this paper, our problem, termed the **Master Physician Scheduling Problem**, is the tactical planning problem of assigning physician activities to the time slots over a time horizon incorporating a large number of rostering and resource constraints together with complex

Aldy Gunawan
School of Information Systems, Singapore Management University
80 Stamford Road, Singapore 178902
E-mail: aldygunawan@smu.edu.sg

Hoong Chuin Lau
School of Information Systems, Singapore Management University
80 Stamford Road, Singapore 178902
E-mail: hclau@smu.edu.sg

physician preferences. The main objectives are to satisfy as many physicians' preferences and duty requirements as possible while ensuring optimum usage of available resources such as clinics and operating theatres.

The major contributions/highlights of this paper are as follows:

(1) We take a physician-centric approach to solving this problem, since physician retention is the most critical issue faced by hospital administrations worldwide.

(2) We formulate the problem as a bi-objective optimization problem and solve the problem by different methods: ε–Constraint Method, Weighted-Sum Method and Hill-Climbing Algorithm.

The organization of the paper is as follows. Section 2 gives some literature review. Section 3 gives a detailed description of the master physician scheduling problem. In Section 4, we propose a bi-objective mathematical programming model along with the description of notation and variables, constraints and objective functions. Section 5 discusses three different methods used to solve the problem. Section 6 makes a computational analysis of the model with a real case from the Surgery Department of a large local government hospital, as well as on randomly generated problem instances. Finally, we provide some concluding perspectives and directions for future research in Section 7.

2 Literature Review

There have been a number of review papers in the area of personnel scheduling and rostering research, as in the works of Aggarwal (1982), Burke et al. (2004), Ernst et al. (2004). Much of the research on personnel scheduling in health care has been devoted to the case of nurse scheduling problem (e.g. Burke et al., 2004; Ernst et al, 2004; Bard and Purnomo, 2005; Beliën and Demeulemeester, 2005; Petrovic and Berghe, 2008). On the other hand, little work has been done on the physician scheduling problem. Carter and Lapierre (2001) provide the fundamental differences between physicians and nurses scheduling problems. Unlike nurse rostering problems, in physician scheduling, maximizing satisfaction only matters, as physician retention is the most critical issue faced by hospital administrations. In addition, while nurse schedules must adhere to collective union agreements or written rules, physician schedules are more driven by personal preferences and with no formal scheduling rules.

Physician and nurse scheduling problems are typically multi-objective by nature. One approach for handling multi-objective optimization problem is to formulate the objectives as soft constraints and define the global objective function as the total deviations in the soft constraints (Beaulieu, et al., 2000; Topaloglu, 2006, 2009; Burke et al., 2009). Another way to solve a multi-objective problem is to apply the Weighted-Sum method that combines the objectives into a single scalar value (Beaulieu et al., 2000, Carter and Lapierre, 2001; Blöchliger, 2004; Topaloglu, 2006; Beliën et al., 2009; Puente et al., 2009; Topaloglu, 2009). Yet another method that has also been considered is the sequential method (Topaloglu, 2009). In this method, objectives are sorted in descending order of importance and optimized in an iterative procedure. Another most commonly

used method is goal programming since it allows simultaneous solution of multiple objectives (Ozkarahan, 2000; Ogulata and Erol, 2003; Topaloglu, 2006; White et al., 2006).

Burke et al. (2007) and Burke et al. (2009) presented a Pareto-based optimization technique based on a Simulated Annealing algorithm to address nurse scheduling problems in the real world. One of the latest papers about physician rostering problem is presented by Puente et al. (2009). The problem consists in designing timetables for the physicians at the Emergency Department in a hospital.

3 Problem Definition

This paper focuses on a physician scheduling problem for the Surgery Department of a large government hospital in Singapore. The problem (termed the Master Physician Scheduling Problem) is to assign different physician duties (or activities) to the defined time slots over a time horizon incorporating a large number of constraints and complex physician preferences. For simplicity, we assume the time horizon to be one work week (Mon-Fri), further partitioned into 5 days and 2 shifts (AM and PM).

The work mode combines shifts and duties. Physicians may specify their respective *ideal schedule* in terms of the duties they like to perform on their preferred days and shifts, as well as shifts-off or days off. An actual schedule is generated by taking the physicians' preferences together with resource capacity and rostering constraints into consideration (Figure 1).

Due to conflicting constraints, the ideal schedules might not be fully satisfied in the actual schedule (see Figure 1 for illustration). That may occur in two possible scenarios:

- Some duties have to be scheduled on different shifts or days – which we term *non-ideal scheduled duties* (e.g. Physician 2 Tuesday duties).
- Some duties simply cannot be scheduled due to resource constraints – which we term *unscheduled duties* (e.g. Physician 1 Friday PM duty).

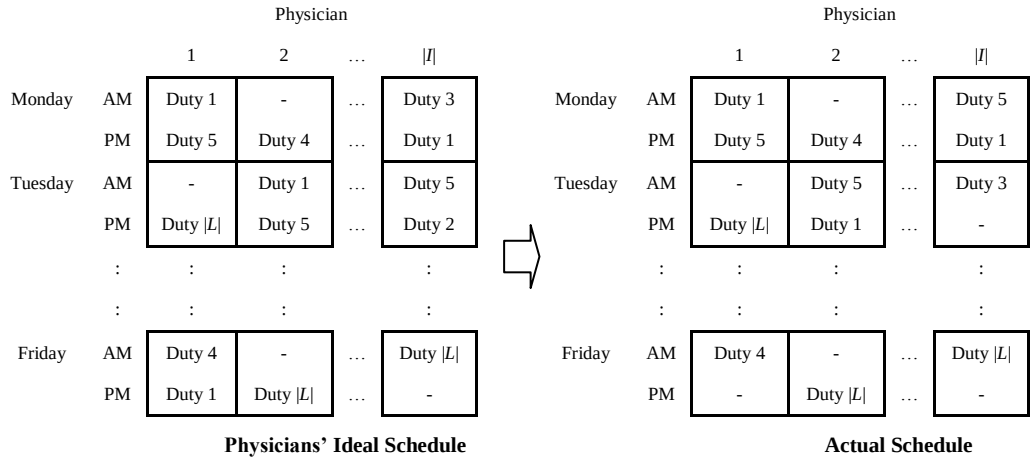


Figure 1. Example of the master physician scheduling problem

The master physician scheduling problem is a highly constrained resource allocation problem. The constraints imposed are categorizes into two different types: hard and soft

constraints. Our goal is to meet the hard constraints while aiming at a high-quality result with respect to soft constraints. The hard constraints in our problem are as follows:

- *H*1: No physician can perform more than one duty in any shift.
- *H*2: The number of resources (e.g. operating theatres, clinics) needed cannot exceed their respective capacities at any time. For simplicity, we assume that each type of activity does not share its resources with another type of activities – for example, operating theatres and clinics are used to perform surgery and out-patient duties, respectively.
- *H*3: The number of activities allocated to each physician cannot exceed his contractual commitments, and do not conflict with his external commitments. In this paper, we assume external commitments take the form of physicians' request for shifts-off or days-off, and hence no duty should be assigned to these requests.

The master physician scheduling problem incorporates both physician preferences and ergonomic constraints, optimizing on two objectives - maximizing the number of ideal schedules and minimizing the number of unscheduled duties. These objectives are related to the following soft constraints:

- *S*1: Duties should be scheduled with respect to the *ideal schedule*.
- *S*2: For some heavy duties, such as surgery and endoscopy duties, that could not be scheduled with respect to the ideal schedule, we try to reschedule these duties with respect to the ergonomic constraints:
  - If a physician is assigned to a heavy duty in the morning shift, then he cannot be assigned to another type of heavy duty in the afternoon shift *on the same day*. However, it is possible to assign *the same* type of heavy duties in consecutive shifts on the same day.
  - Similarly, a physician cannot also be assigned to another type of heavy duty in the morning shift on a particular day if he has been assigned to a heavy duty in the afternoon shift on the previous day.

4 Mathematical Programming Model

The following notation is required to formulate the mathematical programming model.

Parameters

I = Set of physicians, $i \in \{1,2,\ldots,|I|\}$

J = Set of days, $j \in \{1,2,\ldots,|J|\}$

K = Set of shifts per day, $k \in \{1,2,\ldots,|K|\}$

L = Set of duties, $l \in \{1,2,\ldots,|L|\}$

L^H = $\{ l \in L : l = \textit{heavy}$ duty$\}$

PRA = $\{(i,j,k) \in I \times J \times K : (i,j,k) =$ physician i requests not being assigned on day j shift $k\}$

R_l = number of resources required to perform duty l $(l \in L)$

C_{jkl} = number of resources available for duty l on day j shift k $(j \in J, k \in K, l \in L)$

(i.e. resource capacity)

A_{il} = number of duty *l* requested by physician *i* in a weekly schedule $(i \in I, l \in L)$

F_{ijkl} = 1 if physician *i* requests duty *l* on day *j* shift *k* (*ideal schedule*), 0 otherwise $(i \in I, j \in J, k \in K, l \in L)$

Decision and auxiliary variables

X_{ijkl} = 1 if physician *i* is assigned to duty *l* on day *j* shift *k* with respect to the ideal schedule, 0 otherwise

Y_{ijkl} = 1 if physician *i* is assigned to duty *l* on day *j* shift *k* with respect to the ergonomic constraints, 0 otherwise

U_i = number of unscheduled duties of physician *i*

N_i = number of non-ideal scheduled duties of physician *i*

S_i = number of ideal scheduled duties of physician *i*

We consider the problem that optimizes physician ideal schedules on one hand, and on the other, improves the quality of duty transition on non-ideal scheduled slots through ergonomic constraints. More precisely, we are concerned with the bi-objective problem of maximizing the number of ideal scheduled duties (1) and minimizing the number of unscheduled duties under ergonomic constraints (2).

Maximize $Z_1 = \sum_{i \in I} S_i$ (1)

Minimize $Z_2 = \sum_{i \in I} U_i$ (2)

subject to:

$R_l \times \sum_{i \in I} (X_{ijkl} + Y_{ijkl}) \leq C_{jkl}$ $j \in J, k \in K, l \in L$ (3)

$X_{ijkl} + Y_{ijkl} \leq 1$ $i \in I, j \in J, k \in K, l \in L$ (4)

$\sum_{j \in J} \sum_{k \in K} (X_{ijkl} + Y_{ijkl}) \leq A_{il}$ $i \in I, l \in L$ (5)

$\sum_{l \in L} (X_{ijkl} + Y_{ijkl}) \leq 1$ $i \in I, j \in J, k \in K$ (6)

$\sum_{l \in L} (X_{ijkl} + Y_{ijkl}) = 0$ $(i, j, k) \in PRA$ (7)

$X_{ijkl} \leq F_{ijkl}$ $i \in I, j \in J, k \in K, l \in L$ (8)

$U_i = \sum_{l \in L} A_{il} - \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} (X_{ijkl} + Y_{ijkl})$ $i \in I$ (9)

$S_i = \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} X_{ijkl}$ $i \in I$ (10)

$N_i = \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} Y_{ijkl}$ $i \in I$ (11)

$Y_{ijkl_1} + X_{ij(k+1)l_2} + Y_{ij(k+1)l_2} \leq 1 \quad i \in I, j \in J, k \in \{1,2,\ldots,|K|-1\}, l_1 \& l_2 \in L^H (l_1 \neq l_2)$ (12)

$Y_{ij|K|l_1} + X_{i(j+1)1l_2} + Y_{i(j+1)1l_2} \leq 1 \quad i \in I, j \in \{1,2,\ldots,|J|-1\}, l_1 \& l_2 \in L^H (l_1 \neq l_2)$ (13)

$$Y_{ij|K|l_1} + X_{ij1l_2} + Y_{ij1l_2} \le 1 \qquad i \in I, j \in J, l_1 \& l_2 \in L^H (l_1 \ne l_2) \tag{14}$$

$$Y_{ij1l_1} + X_{i(j-1)|K|l_2} + Y_{i(j-1)|K|l_2} \le 1 \qquad i \in I, j \in \{2,3,...,|J|\}, l_1 \& l_2 \in L^H (l_1 \ne l_2) \tag{15}$$

$$X_{ijkl}, Y_{ijkl} \in \{0,1\} \qquad i \in I, j \in J, k \in K, l \in L \tag{16}$$

$$U_i, N_i, S_i \in Z^+ \qquad i \in I \tag{17}$$

Constraint (3) ensures that the total number of resources required does not exceed total number of available resources per shift (the resource capacity constraint). Note that R_l is set to zero for activities without limited number of resources available. (4) ensures that a duty is scheduled as either an ideal or a non-ideal duty. (5) represents the number of duties allocated to each physician cannot exceed his contractual commitments. (6) ensures that each physician cannot be assigned more than one duty in any shift, while (7) ensures that no duty would be assigned to a physician during any shifts-off or days-off requested. Duties represented by X_{ijkl} have to be scheduled with respect to the *ideal schedule* (constraint (8)). Constraints (9), (10) and (11) calculate the number of unscheduled duties, ideal scheduled duties and non-ideal scheduled duties, respectively. The details of ergonomic constraints are represented by (12) – (16). Finally, (16) imposes the 0-1 restrictions for the decision variables X_{ijkl} and Y_{ijkl} while (17) is the nonnegative integrality constraint for the decision variables U_i, N_i and S_i.

In the following section, three different approaches are proposed to solve the bi-objective physician scheduling problem: one based on ε–Constraint approach that obtains a single solution, and the others based on Weighted-Sum Method and Hill-Climbing Algorithm that obtains non-dominated or Pareto-optimal solutions.

5 Proposed Methods

5.1 ε–Constraint Method

The ε–Constraint Method was suggested by Haimes et al. (1971). In this method, the bi-objective problem is reformulated by just keeping one of the objective functions and restricting the other objective function within user-specified value. Here, we decide to restrict the number of unscheduled duties to be less than or equal to the values obtained by solving another model proposed by Gunawan and Lau (2009) (denote by U_i^*). Therefore, the model only focused on minimizing the number of unscheduled duties with respect to ergonomic constraints. The modified problem is as follows:

[**ε–Constraint Model**]

Maximize $Z_1 = \sum_{i \in I} S_i$ (18)

subject to:

constraints (3) – (17)

$$U_i \le U_i^* \qquad i \in I \qquad (19)$$

5.2 Weighted-Sum Method

The Weighted-Sum Method is the simplest approach and commonly used to solve the multiple-objective optimization problem. It formulates the problem as a classical multi-objective weighted-sum model that combines two objectives into a single objective by multiplying each objective with a user-defined weight. The weight of each objective is usually chosen in proportion to the objective's relative importance in the problem.

[**Weighted-Sum Model**]

Minimize $Z = W_1 \times \left(-\sum_{i \in I} S_i\right) + W_2 \times \left(\sum_{i \in I} U_i\right)$ (20)

subject to: constraints (3) – (17)

Note that in Weighted-Sum Model, the original objective function Z_1 is transformed into a minimization objective function. The advantage of the Weighted-Sum method is that it guarantees finding Pareto-optimal solutions for convex optimization problems, which can be inferred from Deb (2003) Theorem 3.1.1:

Corollary: The solution to the Weighted-Sum Model is not Pareto-optimal iff either W_1 or W_2 is set to zero.

Algorithm

(1) Set $W_1 = 1$
(2) **Repeat**
(3) Set $W_2 = 1 - W_1$
(4) Solve the Weighted-Sum Model optimally (using mathematical programming)
(5) $W_1 = W_1 - 0.1$
(6) **Until** $W_1 < 0$
(7) For all solutions generated by the above, let M denote the subset of Pareto-optimal solutions
(8) **For** a pre-set number of iterations **do** the following
(9) Let M_1 and M_2 ($\in M$) with the lowest and the second lowest total number of unscheduled duties, respectively
(10) Set $W'_1 = W_1$ of solution M_1 and $W'_2 = W_2$ of solution M_1
(11) Set $W''_1 = W_1$ of solution M_2 and $W''_2 = W_2$ of solution M_2
(12) Calculate new weight values, denoted as W^*_1 and W^*_2, as follows:
$W^*_1 = (W'_1 + W''_1)/2$
$W^*_2 = 1 - W^*_1$
(13) Solve the Weighted-Sum Model with $W_1 = W^*_1$ and $W_2 = W^*_2$
(14) **If** the solution obtained is a new Pareto-optimal solution
(15) **Then** update M
(16) **Else if** the solution obtained and M_1 are the same
(17) Set the solution obtained as M_1 and Update M
(18) **Else if** the solution obtained and M_2 are the same
(19) Set the solution obtained as M_2 and Update M

Figure 2. Algorithm to obtain Pareto-optimal solutions

In this paper, instead of using a single set of weight values, several different sets of weight values would be used to efficiently generate a set of Pareto-optimal solutions. First, a constant k

number of solutions with different values of W_1 uniformly distributed between [0, 1] are generated. Since not all Pareto-optimal solutions may be discovered by the initial set of weight values, we introduce an adaptive exploration on the neighborhood of weight values using linear interpolation, i.e. we examine two different Pareto-optimal solutions to derive weight values for obtaining other possible optimal solutions. The detail of the algorithm is presented in Figure 2.

5.3 Hill-Climbing Algorithm

In this section, we turn to a Hill Climbing Algorithm to generate a set of non-dominated solutions. The initial solution is generated by setting one of the weight values to 1. Next, a set M of *potentially non-dominated solutions* would be generated. This set is updated whenever a new non-dominated solution $\mathbf{x}'$ is generated. This updating process consists of two possible actions:

(1) Adding $\mathbf{x}'$ to M if there is no other solution $\mathbf{v} \in M$ such that $\mathbf{v}$ dominates $\mathbf{x}'$

(2) Removing all solutions from set M which are dominated by $\mathbf{x}'$

The Hill-Climbing Algorithm will terminate when either there is no unscheduled duties or it reaches a pre-set number of iterations. The algorithm is given as follows.

Hill-Climbing Algorithm
(1) Generate a starting solution $\mathbf{x} \in D$, where D is the set of feasible solutions
(2) $M := \emptyset$
(3) Update M of *potentially efficient solutions* with $\mathbf{x}$
(4) Repeat
(5) Select one solution $\mathbf{x} \in M$
(6) Construct a new solution $\mathbf{x}' \in V(\mathbf{x})$, where $V(\mathbf{x}) \subseteq D$ is the neighborhood of solution $\mathbf{x}$
(7) **If** a new solution $\mathbf{x}'$ exists
(8) **If** $\mathbf{x}'$ is a non-dominated solution then
(9) Update M
(10) **Until** the stop conditions are satisfied

Figure 3. Hill-Climbing Algorithm

Our proposed neighborhood structure is in essence a kind of ejection chain move involving either one or two physicians and the pool of hitherto unscheduled duties. From the initial solution generated, the *Unscheduled_Pool* contains the list of physicians with the respective number of unscheduled duties. A physician (say physician *i*) and one of his unscheduled duty (say *Duty*1) is selected randomly from the *Unscheduled_Pool* and the aim is to insert it into the schedule, thereby decreasing the total number of unscheduled duties by 1. To do so, one of his scheduled duties (say *Duty*2) at say *slot*2 needs to be reallocated to another timeslot say *slot*1.

Note that each time as a duty is moved to another timeslot, it must satisfy either one of the two following conditions:

Condition1: the duty is allocated to a timeslot that follows the physician's ideal schedule. The net effect is that the total number of ideal scheduled duties either remains the same or increases by 1.

Condition2: the duty is allocated to a timeslot that does *not* follow the physician ideal schedule. In this case, we need to ensure that the ergonomic constraint is not violated. The net effect is that the total number of ideal scheduled duties either remains the same or decreases by 1.

In considering the relocation of *Duty*1 to *slot*2, two possible scenarios are possible:

(1) **Scenario 1**: If there is resource available at *slot*2 to perform *Duty*1 (Figure 4), the move can be performed.

(2) **Scenario 2**: If no resource is available *slot*2 for *Duty*1 (Figure 5), then another physician *j*, who is performing the *same* duty (i.e. *Duty*1) at *slot*2 will be selected (if any) and we apply an ejection chain strategy to swap out the *Duty*1 of physician *j* so as to free up the resource needed.

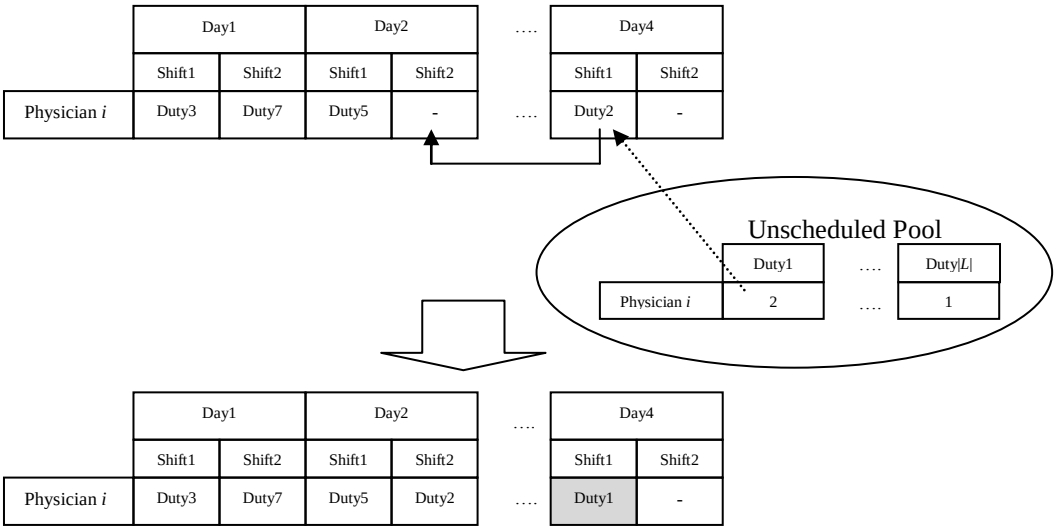


Figure 4. Illustration of Scenario 1

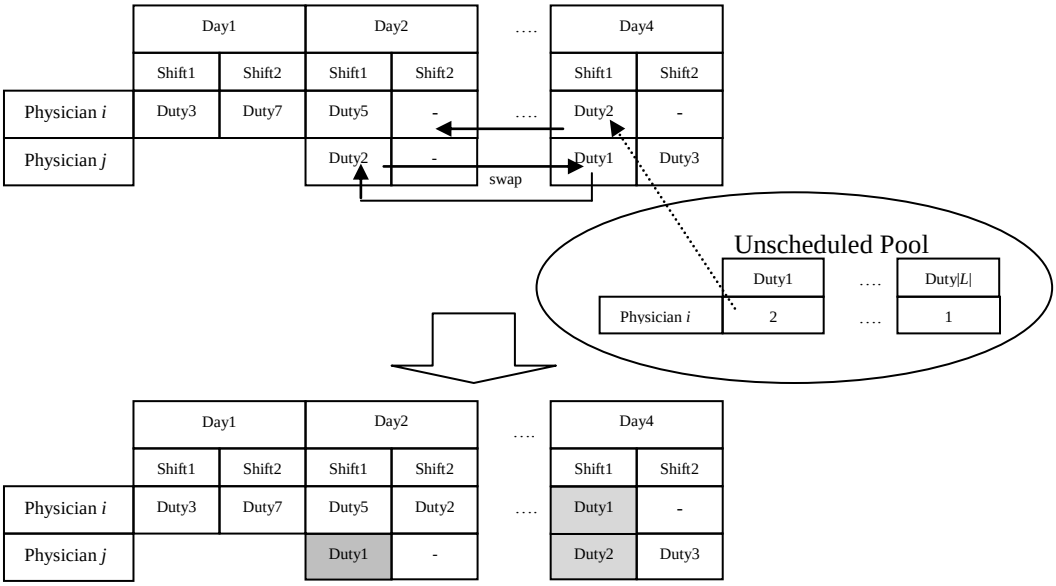


Figure 5. Illustration of Scenario 2

The pseudo-code for generating moves for this neighborhood is as shown in Figure 6.

```
(1)  Select physician i from Unscheduled_Pool randomly
(2)  Find an empty timeslot randomly, slot1
(3)  By considering all scheduled duties of physician i, find one possible time slot2 such that the duty
     allocated at slot2 can be reassigned to slot1
(4)  If it can be rescheduled at slot1,
(5)     Find an unscheduled duty of physician i, Duty1
(6)     If the resource capacity at slot2 for Duty1 is greater than 0
(7)        Evaluate whether Duty1 can be allocated to slot2
(8)        If there is no constraint violation, generate a new possible solution x'
(9)     Else if the resource capacity at slot2 for Duty1 is equal to 0
(10)       Evaluate whether Duty1 can be allocated to slot2
(11)       If there is no conflict,
(12)          Find a physician j who has the same duty scheduled, Duty1, at slot2
(13)          Apply an ejection chain strategy to physician j, by ensuring that all constraints are satisfied
(14)          If there is no constraint violation, generate a new possible solution x'
```

Figure 6. Neighborhood Move

6 Computational Results

To evaluate the performance of the proposed methods, computational experiments were done on 6 different random problem sets and a real case from the Surgery Department of a large local government hospital. The 6 sets problem sets were generated with varying values of the parameter – total percentage of *heavy* duties assigned to physicians (last column of Table 1). For each problem set, we also generate several problem instances with different values of number of resources available in every shift (Table 2). The details about how problem instances were generated are summarized in Gunawan and Lau (2009).

Table 1. Characteristics of problem instances

Problem Set	Number of physicians	Number of shifts per day	Number of days	Number of duties	Number of *heavy* duties	Number of duties with limited capacity	Total percentage of *heavy* duties*
Case study	15	2	5	9	3	3	73%
Random 1	20	2	5	7	3	3	20%
Random 2	20	2	5	7	3	3	30%
Random 3	20	2	5	7	3	3	40%
Random 4	20	2	5	7	3	3	50%
Random 5	20	2	5	7	3	3	60%
Random 6	20	2	5	7	3	3	70%

$* = \left(\sum_{i \in I} \sum_{l \in L^H} A_{il} / |I| \times |J| \times |K|\right) \times 100\%$

In the following sub-sections, we report a suite of computational results and analysis obtained from the proposed methods. The mathematical programming models (ε–Constraint and Weighted-Sum Models) were implemented using ILOG OPL Studio 5.5 and the proposed algorithm (Hill Climbing Algorithm) was coded in C++. All codes are executed on a Intel (R) Core (TM)^2 Duo CPU 2.33GHz with 1.96GB RAM that runs Microsoft Windows XP.

Table 2. Examples of varying values of C_{jkl} (Random 1 and Random 2 instances)

Problem Set	Instances	L		
		Duty 1	Duty 2	Duty 3
Random 1		15	28	22
	Random 1a	3	6	4
	Random 1b	3	5	4
	Random 1c	3	4	4
	Random 1d	3	3	4
	Random 1e	3	3	3
	Random 1f	2	3	3
	Random 1g	1	2	2
Random 2		21	46	32
	Random 2a	4	10	5
	Random 2b	4	9	5
	Random 2c	4	8	5
	Random 2d	4	7	5
	Random 2e	4	6	5
	Random 2f	4	5	5
	Random 2g	4	5	4
	Random 2h	3	5	4
	Random 2i	2	4	3

6.1 Results from ε–Constraint Method

As described in Section 5.1, the physician scheduling problem is reformulated by keeping one objective and restricting the other one within a specified value. In this paper, we restrict the number of unscheduled duties within the number of unscheduled duties generated by another model proposed by Gunawan and Lau (2009).

In Gunawan and Lau (2009), the ergonomic constraints are imposed to all scheduled duties. On the other hand, in this paper, duties are scheduled with respect to either of two criteria: the number of scheduled duties with respect to the physicians' ideal schedules has to be satisfied as many as possible, while non-ideal scheduled duties cannot violate ergonomic constraints.

Table 3. Computational results of ε–Constraint Model

Problem Instances	Number of unscheduled duties	Number of scheduled duties		Percentage of unscheduled duties	Percentage of scheduled duties	
		Ideal	Non-ideal		Ideal	Non-ideal
Case study	8	135	7	5.3	90.0	4.7
Random 1a	0	196	4	0.0	98.0	2.0
Random 1b	0	192	8	0.0	96.0	4.0
Random 1c	0	192	8	0.0	96.0	4.0
Random 1d	4	186	10	2.0	93.0	5.0
Random 1e	5	181	14	2.5	90.5	7.0
Random 1f	5	180	15	2.5	90.0	7.5
Random 1g	10	173	17	5.0	86.5	8.5
Random 2a	0	196	4	0.0	98.0	2.0
Random 2b	0	196	4	0.0	98.0	2.0
Random 2c	0	196	4	0.0	98.0	2.0
Random 2d	0	194	6	0.0	97.0	3.0
Random 2e	0	194	6	0.0	97.0	3.0
Random 2f	3	186	11	1.5	93.0	5.5
Random 2g	3	186	11	1.5	93.0	5.5
Random 2h	3	183	14	1.5	91.5	7.0
Random 2i	10	174	16	5.0	87.0	8.0

Table 3 summarizes the results obtained for the real case study, as well as Random 1 and 2 instances. In general, we found that the number of unscheduled duties is relative small compared with the number of ideal scheduled duties (less than or equal to 5.3%). By using this method,

different optimal solutions can be found by setting different U_i^* values. Take note however that it is possible that infeasible solutions would be obtained.

The following table summarizes the average percentages of all our problem sets. It can be observed that the average percentage of ideal scheduled duties is at least 86%, and only Random 5 has the average percentage of non-ideal scheduled duties which is more than 10%.

Table 4. Summary of computational results of ε–Constraint Model

Problem Set	Number of instances	Average percentage of unscheduled duties	Average percentage of scheduled duties	
			Ideal	Non-ideal
Case study	1	5.3	90.0	4.7
Random 1	7	1.7	92.9	5.4
Random 2	9	1.1	94.7	4.2
Random 3	9	1.8	91.2	6.9
Random 4	11	1.1	89.6	9.2
Random 5	13	1.1	86.7	12.2
Random 6	15	2.6	89.5	7.9

6.2 Results from Weighted-Sum Method

In Section 5.2, we proposed an algorithm to generate several possible sets of weight values in order to obtain set of Pareto-optimal solutions. It is started by generating 10 different sets of weight values that uniformly distributed within [0, 1].

In the next step, we set the number of iterations to 5 iterations. This step is applied for further finding of other possible Pareto-optimal solutions. By using linear interpolation, we focus on exploring neighborhoods of the solutions with the lowest values of the total number of unscheduled duties since we view that unscheduled duties as bad compared to non-ideal scheduled duties.

In general, the value of W_1 should be less than 0.5 in order to obtain the lowest number of unscheduled duties. We also found that the higher the percentage of heavy duties, the lower the value of W_1 should be set. It could be due to the difficulty to assign unscheduled heavy duties with respect to ergonomic constraints. That's why we need to give higher importance/value for W_2.

Table 5 represents the results obtained by the proposed algorithm. Here, we only present two representative instances 1g and 6l for illustration purposes. Figure 7 represents the Pareto-optimal solutions obtained for Random 1 and 2 instances.

In general, we observe that the more we increase the weight value of the first objective (W_1), the less we get the number of non-ideal scheduled duties (see Table 5 for illustration). At the same time, the number of unscheduled duties would also be increased since the number of unscheduled duties becomes less important with the decreased weight value of the second objective (W_2). This method could guarantee finding solutions on the Pareto-optimal set. However, we also found that different weight values need not necessarily lead to Pareto-optimal solutions and some sets of weight values might lead to the same solution.

Table 5. Computational results of instances 1g and 6l

Random 1g					Random 6l				
Weight		Number of Scheduled duties		Number of Unscheduled duties	Weight		Number of Scheduled duties		Number of Unscheduled duties
W_1	W_2	Ideal	Non-Ideal		W_1	W_2	Ideal	Non-Ideal	
1.0	0.0	183	0	17	1.0	0.0	181	0	19
0.9	0.1	183	3	14	0.9	0.1	181	9	10
0.8	0.2	183	3	14	0.8	0.2	181	9	10
0.7	0.3	183	3	14	0.7	0.3	181	9	10
0.6	0.4	183	3	14	0.6	0.4	181	9	10
0.5	0.5	181	7	12	0.5	0.5	181	9	10
0.4	0.6	179	11	10	0.4	0.6	179	13	8
0.3	0.7	179	11	10	0.3	0.7	173	22	5
0.2	0.8	179	11	10	0.2	0.8	169	27	4
0.1	0.9	179	11	10	0.1	0.9	160	38	2
0.0*	1.0	31	51	139	0.0*	1.0	113	85	2
0.45	0.55	179	11	10	0.15	0.85	160	38	2
0.475	0.525	179	11	10	0.175	0.825	165	32	3
0.4875	0.5125	179	11	10	0.1625	0.8375	160	38	2
0.49375	0.50625	179	11	10	016875	0.83125	165	32	3
0.496875	0.503125	179	11	10	0.165625	0.834375	160	38	2

* Non Pareto-optimal solution

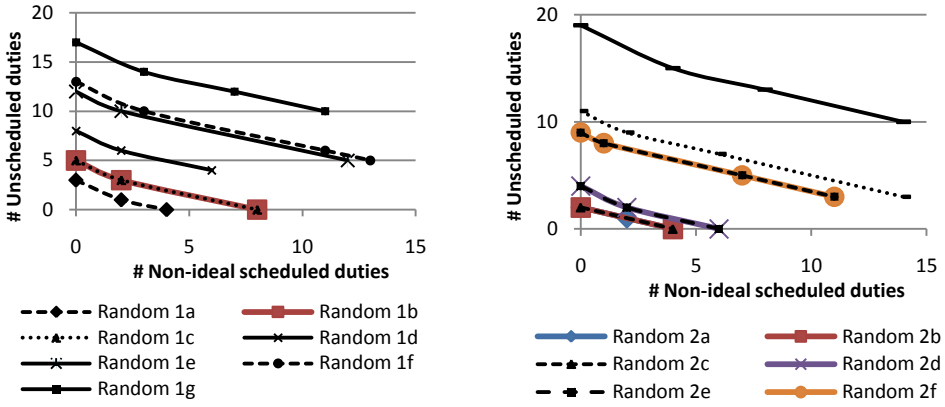


Figure 7. Pareto-optimal solutions of Random 1 and 2 problem sets

The proposed algorithm is also tested to the real case study (Table 6).The value of W_1 should be within [0.9, 1.0] in order to obtain the lowest number of unscheduled duties. The result of the real case study problem by the ε–Constraint and the Weighted-Sum Methods and the actual allocation generated manually by the hospital are also compared.

The number of ideal scheduled duties obtained by the Weighted-Sum Model is significantly higher than that of the manual allocation. Although the number of unscheduled duties obtained by both ε–Constraint Model and Weighted-Sum Model are slightly worse than the number of unscheduled duties via manual allocation, the number of non-ideal scheduled duties is better than that of the manual allocation. One of possible reason is in the manual allocation, the administrator allocates non-ideal scheduled duties to any time slots/shifts without considering the ergonomic constraints. In the manual allocation, there are also two physicians who have to cancel their days-off or shifts-off for other duties. This outcome is very undesirable since they might have external commitments that cannot be delayed or cancelled.

Table 6. Comparison between the manual allocation and model solutions on a real case

	Manual allocation	ε–Constraint Model	Weighted-Sum Model
Number of unscheduled duties	5	8	8
Number of non-ideal scheduled duties	10	7	2
Number of ideal scheduled duties	135	135	140

6.3 Results from Hill-Climbing Algorithm

In this experiment, the number of iterations for Hill Climbing is set to 200 for each test instance. Note that the number of Pareto-optimal solutions obtained by the Weighted-Sum Method is small. For instance, for problem instances Random 1 (i.e. 1a to 1g), the number of Pareto-optimal solutions generated is between 3 and 4, compared with the Hill-Climbing Algorithm which provides up to 10 non-dominated solutions (see Table 7). Figure 8 represents results obtained by the Hill-Climbing Algorithm for some of the representative instances.

Table 7. The number of solutions generated

Problem Set	The range of the number of solutions generated	
	Weighted-Sum Method	Hill-Climbing Algorithm
Case Study	1	2
Random 1	[3,4]	[3,10]
Random 2	[2,4]	[3,11]
Random 3	[3,5]	[2,12]
Random 4	[4,5]	[4,12]
Random 5	[4,5]	[4,16]
Random 6	[4,7]	[4,11]

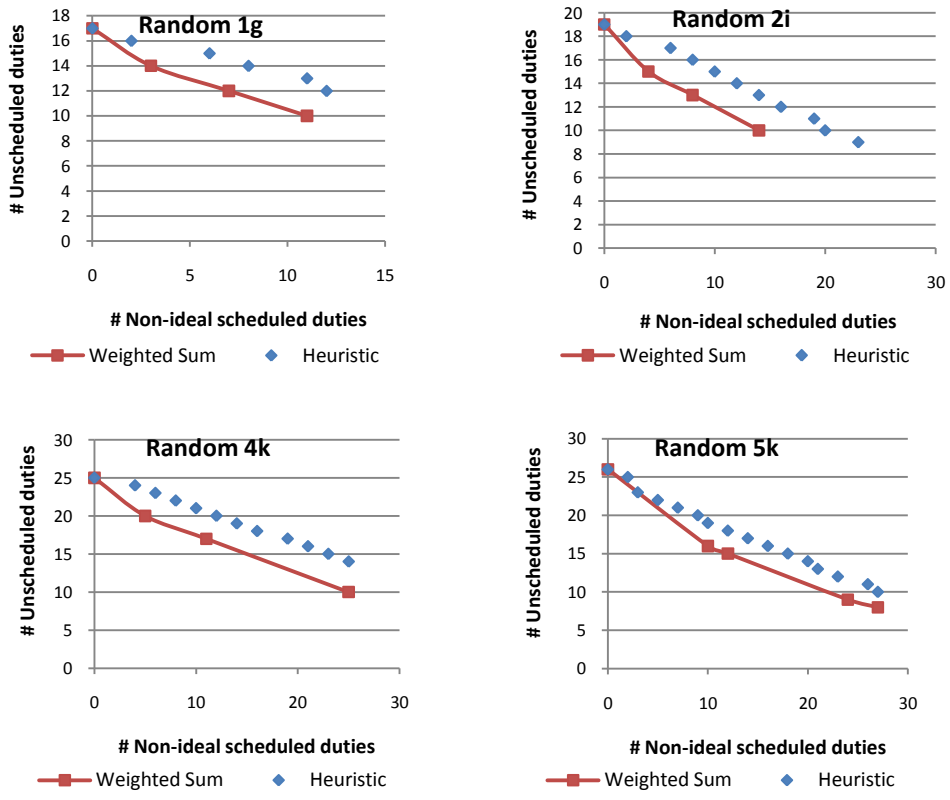


Figure 8. Non-dominated solutions of Hill-Climbing Method

As observed by Burke et al., 2009, one issue in comparing the algorithms in multi-objective problems is that there is no systematic criterion to measure the performance of each algorithm. In Burke et al. (2009), a number of objective functions were considered, and their approach was convert these objectives into goals, and the aim was to minimize the deviations (i.e. the percentage of total number of violations in the solution with respect to the total number of constraints).

In this paper, we choose to measure the deviation of our heuristic approach from Pareto optimality directly. Let the results obtained by the Hill-Climbing and Weighted-Sum Methods be denoted as Sets H and W with sizes of n_H and n_W, respectively. In order to compare and measure the closeness between a solution $\mathbf{x} \in H$ and a solution $\mathbf{y} \in W$, we propose the following formula:

$$dist(\mathbf{x},\mathbf{y}) = \left[\frac{|Z_1(\mathbf{y}) - Z_1(\mathbf{x})|}{Z_1(\mathbf{y})} + \frac{|Z_2(\mathbf{y}) - Z_2(\mathbf{x})|}{Z_2(\mathbf{y})} \right] \Big/ 2 \tag{21}$$

For a particular solution **x**, we calculate n_W different values of *dist*(**x**,**y**) and choose the solution **y** which yields the minimum *dist*(**x**,**y**) value (ties broken arbitrarily). The fitness value of a solution **x** is calculated as follows:

$$Fitness(\mathbf{x}) = \frac{1}{dist(\mathbf{x},\mathbf{y}) + 1} \tag{22}$$

Note that this is a normalized value that falls between 0 and 1, where a value 1 means perfect fit, and tends to 0 as the distance increases.

For each problem instance, we will have n_H different values of *dist*(**x**,**y**). For example, for Random 1g (see Figure 8), six different non-dominated solutions were obtained by the Hill-Climbing Algorithm. The average fitness value associated with a given problem instance is then calculated as follows:

$$Average\ Fitness = \frac{\sum_{x \in H} Fitness(\mathbf{x})}{n_H} \tag{23}$$

Table 8 lists the distances obtained for representative instances Random 1g, 3i, 4k and 5k. We observe that the Hill-Climbing Algorithm produces non-dominated solutions with the fitness values greater than 0.93. Although the results obtained by the Hill-Climbing Method might not be Pareto-optimal solutions, we found that the number of non-dominated solutions generated is more than that of the Weighted-Sum Method. For future research, these non-dominated solutions can be considered as starting points/initial solutions that would be further improved in order to obtain Pareto-optimal solutions.

Table 8. Comparison between the Hill-Climbing Algorithm and the Weighted-Sum Method

Problem Instances	Number of solutions generated by Weighted-Sum Method	Number of solutions generated by Hill Climbing Algorithm	Average Fitness
Random 1g	4	6	0.974
Random 2i	4	11	0.962
Random 4k	4	12	0.954
Random 5k	5	16	0.937

Table 9 summarizes the statistical descriptions of the entire results for all problem sets. The grand mean of average fitness values is above 0.9 which is considered high. Some instances in Random 2 and 3 have the values of 1. The **Grand Mean** column refers to the means of the average fitness values of the respective problem sets.

Table 9. Summary of average fitness values

Problem Set	Number of instances	Grand Mean	Std dev	Minimum	Maximum
Case Study	1	0.96	0.04	0.93	0.99
Random 1	7	0.94	0.03	0.90	0.97
Random 2	9	0.95	0.03	0.92	1.00
Random 3	9	0.96	0.03	0.92	1.00
Random 4	11	0.94	0.01	0.92	0.96
Random 5	13	0.96	0.02	0.93	1.00
Random 6	15	0.94	0.02	0.92	0.97

7 Conclusion

In this paper, we introduce the master physician scheduling problem considering two different objectives simultaneously. Three different multi-objective methods have been proposed. These approaches were tested on a real case from the Surgery Department of a large local government hospital, as well as on randomly generated problem instances. We observe that the objectives were better satisfied compared against the manual allocation.

In terms of future research, there are several potential areas for investigation. An interesting research direction would be to apply other methods, such as Multi-Objective Simulated Annealing, Multi-Objective Tabu Search, and to develop other neighborhood structures in an attempt to improve the solutions. In the same way, we can also consider other constraints, such as fairness constraints, which commonly seen in other hospitals (Gendreau et al., 2007). Another systematic criterion to measure the performance of an algorithm can be considered as future work. We notice that some distance values of the Hill-Climbing Method's solutions might be large. It is probably due to the limitation of the Weighted-Sum Method in generating all possible Pareto-optimal solutions. The application of the ε–Constraint Method is rather limited in this paper; for example, we can consider applying this method to retrieve the complete Pareto-optimal solutions. The main idea is to construct a sequence of ε-Constraint Model based on a progressive modification of U_i^* values (equation (19)) (Deb, 2003; Bérubé et al., 2009).

Acknowledgements We like to thank the Department of Surgery, Tan Tock Seng Hospital (Singapore) for providing valuable comments and test situations.

References

1. Aggarwal, S. (1982). A focused review of scheduling in services. *European Journal of Operational Research, 9*(*2*), 114-121.
2. Bard, J. F., & Purnomo, H. W. (2005). Preference scheduling for nurses using column generation. *European Journal of Operational Research*, *164*, 510-534.

3. Beaulieu, H., Ferland, J. A., Gendron, B., & Philippe, M. (2000). A mathematical programming approach for scheduling physicians in the emergency room. *Health Care Management Science*, *3*, 193-200.
4. Beliën, J., & Demeulemeester, E. (2005). Integrating nurse and surgery scheduling. In Proceedings of the 2nd Multidisciplinary International Scheduling Conference: Theory and Applications 2005, New York, USA, 18-21 July 2005, 408-409.
5. Beliën, J., Demeulemeester, E., & Cardoen, B. (2009). A decision support system for cyclic master surgery scheduling with multiple objectives. *Journal of Scheduling*, *12*, 147-161.
6. Bérubé, J.-F, Gendreau, M., & Potvin, J.-Y. (2009). An exact ε-constraint method for bi-objective combinatorial optimization problems: application to the traveling salesman problem with profits. *European Journal of Operational Research*, *194*, 39-50.
7. Blöchliger, I. (2004). Modeling staff scheduling problems. A tutorial. *European Journal of Operational Research*, *158*, 533-542.
8. Brandeau, M.L., Sainfort, F., & Pierskalla, W.P. (2004). *Operations research and healthcare: A handbook of methods and applications*. Dordrecht: Kluwer Academic.
9. Burke, E. K., & Riise, A. (2008). Surgery allocation and scheduling. In Proceedings of the 7th International Conference of Practice and Theory of Automated Timetabling 2008, Montreal, Canada, 18-22 August 2008.
10. Burke, E. K., De Causmaecker, P., Vanden Berghe, G., & Van Landeghem H. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, *7*(*6*), 441-499.
11. Burke, E. K., Li, J., & Qu, R. (2009). A Pareto-based search methodology for multi-objective nurse scheduling. *Annals of Operations Research*, DOI: 10.1007/s10479-009-0590-8, published online.
12. Burke, E. K., Li, J., Petrovic, S. & Qu, R. (2007). A new Pareto-optimality based metaheuristic approach to the multi-objective nurse scheduling problem. Technical Report, School of Computer Science and IT, University of Nottingham.
13. Carter, M. W., & Lapierre, S. D. (2001). Scheduling emergency room physicians. *Health Care Management Science*, *4*, 347-360.
14. Deb, K. (2003). Multi-objective optimization using evolutionary algorithms. Wiley & Sons, Chichester, New York.
15. Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, *153*, 3-27.
16. Gendreau, M., Ferland, J., Gendron, B., Hail, N., Jaumard, B., Lapierre, S., Pesant, G., & Soriano, P. (2007). Physician scheduling in emergency rooms. In Burke, E. K., & Rudová, H. (eds.), The Practice and Theory of Automated Timetabling VI (PATAT'06, Selected papers). *Lecture Notes in Computer Science*, *3867*, 53-66, Springer, Heidelberg.
17. Gunawan, A., & Lau, H.C. (2009). Master physician scheduling problem. In Proceedings of the 4th Multidisciplinary International Scheduling Conference 2009, Dublin, Ireland, 10 – 12 August 2009.

18. Haimes, Y.Y., Lasdon, L.S., & Wismer, D.A. (1971). On a bicriteriona formulation of the problems of integrated system identification and system optimization, *IEEE Trans. Syst., Man Cybernet*, *1*, 296-297.
19. Ogulata, S. N., & Erol, R. (2003). A hierarchical multiple criteria mathematical programming approach for scheduling general surgery operations in large hospitals. *Journal of Medical Systems*, *27*(*3*), 259-270.
20. Ogulata, S. N., Koyuncu, M., & Karakas, E. (2008). Personnel and patient scheduling in the high demanded hospital services: a case study in the physiotherapy service. *Journal of Medical Systems*, *32*(*3*), 221-228.
21. Ozkarahan, I. (2000). Allocation of surgeries to operating rooms by goal programming. *Journal of Medical Systems*, *24*(*6*), 339-378.
22. Petrovic, S., & Vanden Berghe, G. (2008). Comparison of algorithms for nurse rostering problems. In Proceedings of the 7th International Conference of Practice and Theory of Automated Timetabling 2008, Montreal, Canada, 18 – 22 August 2008.
23. Puente, J., Gómez, A., Fernández, I., Priore, P. (2009). Medical doctor rostering problem in a hospital emergency department by means of genetic algorithm. *Computers & Industrial Engineering*, *56*, 1232-1242.
24. Roland, B., Di Martinelly, C., Riane, F. & Pochet, Y. (2009). Scheduling an operating theatre under human resource constraints. *Computers & Industrial Engineering*, article in press, corrected proof, available online, 14 January 2009.
25. Testi, A., Tanfani, E., & Torre, G. (2007). A three-phase approach for operating theatre schedules. *Health Care Management Science*, *10*, 163-172.
26. Topaloglu, S. (2006). A multi-objective programming model for scheduling emergency medicine residents. *Computers & Industrial Engineering*, *51*, 375-388.
27. Topaloglu, S. (2009). A shift scheduling model for employees with different seniority levels and an application in healthcare. *European Journal of Operational Research*, *198*, 943-957.
28. Vassilacopoulos, G. (1985). Allocating doctors to shifts in an accident and emergency department. *Journal of the Operational Research Society*, *36*, 517-523.
29. White, C.A., Nano, E., Nguyen-Ngoc, D.H, White, & G.M. (2006). An evaluation of certain heuristic optimization algorithms in scheduling medical doctors and medical students. In Proceedings of the 6th International Conference of Practice and Theory of Automated Timetabling 2006, Brno, The Czech Republic, 30 August – 1 September 2006.

Combining VNDS with Soft Global Constraints Filtering for Solving NRPs

J-P.Métivier · P. Boizumault · S. Loudni

Abstract Nurse Rostering Problems (NRPs) consist of generating rosters where required shifts are assigned to nurses over a scheduling period satisfying a number of constraints. In [25], we have shown how soft global constraints can be used to model NRPs in a concise and elegant way. In this paper we go one step further by proposing new neighborhood heuristics for VNS/LDS+CP. Experiments show that, despite its genericity and flexibility, our approach supplies excellent results on small and middle size problems and very promising results on large scale problems.

Keywords NRP · Constraint Programming · VNS · VNDS · Soft Global Constraints.

1 Introduction

Due to their complexity and importance in real world modern hospitals, Nurse Rostering Problems (NRPs) have been extensively studied in both Operational Research (OR) and Artificial Intelligence (AI) for more than 40 years [5,13]. Most NRPs in real world are NP-hard [20] and are particularly challenging as a large set of different rules and specific nurse preferences need to be satisfied to warrant high quality rosters for nurses in practice. Other wide ranges of heterogeneous and specific constraints usually make the problem over-constrained and hard to solve efficiently [1,32].

NRPs consist of generating rosters where required shifts are assigned to nurses over a scheduling period satisfying a number of constraints [5,10]. These constraints are usually defined by regulations, working practices and preferences of nurses and are usually categorised into two groups: hard constraints and soft constraints (with their violation costs).

Jean-Philippe Métivier
GREYC (CNRS - UMR 6072) – Université de Caen, Campus II – Boulevard du Maréchal Juin, 14000 Caen Cedex, FRANCE
Tel.: + 33 (0)2 31 56 74 84
Fax: + 33 (0)2 31 56 73 30
E-mail: Jean-Philippe.Metivier@info.unicaen.fr

Patrice Boizumaut and Samir Loudni
GREYC (CNRS - UMR 6072) – Université de Caen, Campus II – Boulevard du Maréchal Juin, 14000 Caen Cedex, FRANCE

VNS/LDS+CP [23] is a generic local search method based on VNDS (Variable Neighborhood Decomposition Search [15]) for solving over-constrained problems. Neighborhoods are obtained by unfixing a part of the current solution according to a neighborhood heuristic. Then the exploration of the search space related to unfixed part of the current solution is performed using LDS (Limited Discrepancy Search [16]) combined with Constraint Propagation (Filtering).

Global constraints are often key elements in successfully modelling and solving real-life problems due to their efficient filtering. Global constraints are particularly well suited for modelling (hard) NRP constraints [3,35]. More recently, soft global constraints proposed by [24,31,33,37] enable to quantify the violation while keeping the efficiency of their filtering. In [25], we have shown how soft global constraints can be used to model NRPs in a concise and elegant way. In this paper we go one step further by proposing new neighborhood heuristics for VNS/LDS+CP. Such heuristics provide results better than those described in [25]. Experiments show that, despite its genericity and flexibility, our approach supplies excellent results on small and middle size problems and very promising results on large scale problems.

Section 2 gives a synthetic overview of NRPs. Section 3 describes VNS/LDS+CP and reviews neighborhood heuristics already proposed for solving NRPs using VNS. We performed experiments over different instances selected to be representative of the diversity and the size of NRPs (Section 4). For each selected instance, we compare (Section 5) quality of solutions and computing times for our method with the best known ad'hoc method for solving it [18]. Finally we conclude and draw some further works.

2 Nurse Rostering Problems

2.1 An overview of NRPs

NRPs consist of generating rosters where required shifts are assigned to nurses over a scheduling period (planning horizon) satisfying a number of constraints [5,13]. These constraints are usually defined by regulations, working practices and nurses preference. Constraints are usually categorised into two groups: hard and soft ones.

Hard constraints must be satisfied in order to obtain feasible solutions for use in practice. A common hard constraint is to assign all shifts required to the limited number of nurses.

Soft constraints are not mandatory but are desired to be satisfied as much as possible. The violations of soft constraints in the roster are used to evaluate the quality of solutions. A common soft constraint in NRPs is to generate rosters with a balanced workload so that human resources are used efficiently.

Shift types are hospital duties which usually have a well-defined start and end time. Many nurse rostering problems are concerned with the three traditional shifts Morning, (7:00–15:00), Evening (15:00–23:00), and Night (23:00–7:00). Nurses can possess different skills and cover can be defined for each skill.

Different kinds of constraints can be imposed on the work planning of a nurse:

(i) Shift constraints set the minimal and/or maximal number of nurses of certain skill level working in each shift as well as within each group during the planning period.

(ii) Pattern constraints ensure a certain level of quality for the plannings produced and may be specified either globally for the staff or only for certain individuals. Typical requirements are:

- *patterns of shifts* (i.e. minimal and/or maximal total number of particular sequences of shifts) between any two types of shifts in the planning period (e.g., at least one day-off per week),
- *length of stretches* of shifts of identical type to avoid working too few or too many days in a row on a certain shift (e.g. working more than 4 consecutive day shifts is not permited),
- *patterns of stretches* such as *forward rotation* (going from day shifts to evening shifts to night shifts to day shifts again),
- and *patterns of stretches* of a *given length* that ask for at least so many consecutive shifts of a certain type right after shifts of another type (e.g., there must be a day-off before and after working for three consecutive night shifts).

(iii) Workload constraints are used to model the requirements of min/max total number of hours and/or shifts of specific type worked by each nurse in the planning period.

2.2 Example: `Valouxis` Instance

For the `Valouxis` instance [36], 16 nurses must be planned over a period of 4 weeks. Three shifts are considered: M (Morning), E (Evening) and N (Night). O (Off) will represent repose. The following hard and soft constraints must be enforced. Fig. 1 describes a planning of cost 60 for this instance.

1. **Hard constraints:**

(H1) From Monday to Friday, M, E and N shifts require respectively (4,4,2) nurses.
(H2) For weekend, M, E and N shifts require respectively (3,3,2) nurses.

2. **Soft constraints:**

(S1) For each nurse, the number of M shifts should be within the range [5..8]. Any deviation δ is penalised by a cost $\delta \times 1000$.
(S2) For each nurse, the number of E shifts should be within the range [5..8]. Any deviation δ is penalised by a cost $\delta \times 1000$.
(S3) For each nurse, the number of N shifts should be within the range [2..4]. Any deviation δ is penalised by a cost $\delta \times 1000$.
(S4) Each nurse must have at least 10 days Off. Any shortage δ generates a cost $\delta \times 1000$.
(S5) Each nurse must have at most 13 days Off. Any excess δ generates a cost $\delta \times 100$.
(S6) Over a period of 4 weeks, each nurse must have at least 1 free Sunday. Any violation of this rule is penalised by a cost 1000.
(S7) Each nurse should not work more than 3 consecutive N shifts. Any excess δ generates a cost $\delta \times 1000$.
(S8) Shift changes must be performed respecting the order: M, E, N. Any violation of this rule is penalised by a cost 1000.
(S9) Each isolated working day is penalised by a cost 1000.
(S10) Each isolated day off is penalised by a cost 1000.
(S11) Each nurse should work 4 consecutive days. Any excess δ generates a cost $\delta \times 1000$. Each period of 2 (resp. 3) consecutive working days is respectively penalized by a cost 40 (resp. 20).

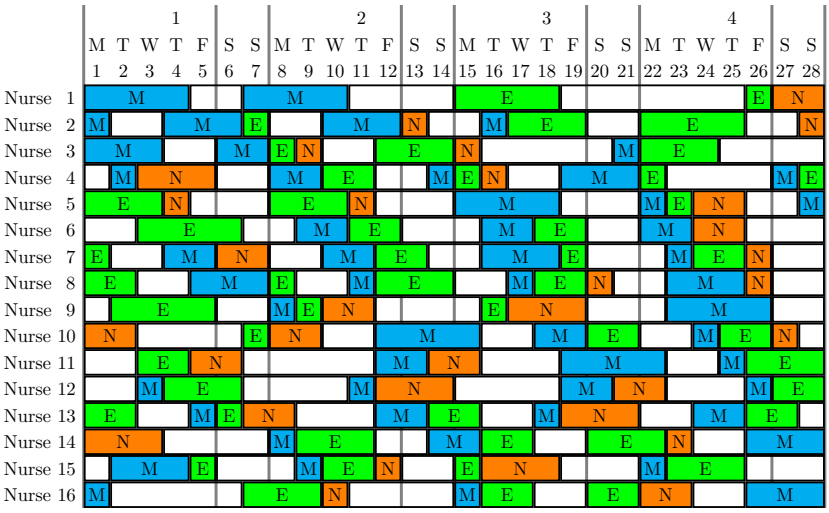


Fig. 1 A solution of cost 60 for the `Valouxis` instance.

3 Solving NRPs

3.1 VNS/LDS+CP

A wide range of approaches and techniques have been proposed for solving NRPs. These include ad'hoc OR methods (by means of mathematical programming with preprocessing steps to reduce the problem size), constructive heuristics and local search methods combining OR techniques to find an initial solution (see [5,13] for a comprehensive review). Of those techniques that have been applied to NRPs, metaheuristics dealing with large-scale neighborhoods (2-opt, swap and interchange of large portions of nurse plannings, ...) such as Variable Neighbourhood Search (VNS) seem to be well suited and very effective.

VNS is a metaheuristic which systematically exploits the idea of large neighborhood change, both in descent to local minima and in escape from the valleys which contain them [27]. Variable Neighborhood Decomposition Search (VNDS) [15] extends basic VNS within a successive approximations method. For a solution of size n, all but k variables are fixed, and VNDS solves a sub-problem in the search space defined by the k unfixed variables.

3.1.1 General overview

VNS/LDS+CP [23] is a generic local search method based on VNDS, where neighborhoods are obtained by unfixing a part of the current solution according to a neighborhood heuristic. Then the exploration of the search space related to the unfixed part of the current solution is performed using a Limited Discrepancy Search (LDS [16]) combined with Filtering in order to benefit from the efficiency of soft global constraints filtering (See Algorithm 1).

Algorithm 1: Pseudo-code for VNS/LDS+CP.

```
   function VNS/LDS+CP(𝒳, 𝒞, k_init, k_max, δ_max)
1  begin
2      s ← genInitialSol(𝒳)
3      k ← k_init
4      while (k < k_max) ∧ (not timeout) do
5          𝒳_unaffected ← H_neighbor(𝒩_k, s)
6          𝒜 ← s\{(x_i = a) s.t. x_i ∈ 𝒳_unaffected}
7          s' ← NaryLDS(𝒜, 𝒳_unaffected, δ_max, 𝒱(s), s)
8          if 𝒱(s') < 𝒱(s) then
9              s ← s'
10             k ← k_init
11         else k ← k + 1
12     return s
```

Unlike an usual VNS scheme, our approach offers two main advantages: first, by focusing efforts on improving only a part of the solution, we restrict the size of the search space and intensify search to improve the current solution; second, even if the exploration of (very) large neighborhoods requires a too expensive effort, the use of LDS allows to efficiently explore parts of the search space.

Algorithm 1 shows the general pseudo-code of VNS/LDS+CP, with k_{init} (resp. k_{max}) the minimal (resp. maximal) number of variables to be unassigned and δ_{max} the maximal number of discrepancies allowed for LDS. A subset of k variables (k is the *dimension* of the neighborhood) is selected by the neighborhood heuristic $\texttt{H}_{neighbor}$ in $\mathcal{N}_k$ (set of all subsets of k variables among $\mathcal{X}$) (line 5). A partial assignment $\mathcal{A}$ is generated from the current solution s by unassigning the k selected variables ; the $(n-k)$ non-selected variables keep their current value in s (line 6). Then, unassigned variables are rebuilt by a partial tree search LDS, combined with constraint propagation based on filtering of global constraints. If a solution of better quality s' is found in the neighborhood of s (line 8), then s' becomes the current solution and k is reset to k_{init} (lines 9-10). Otherwise, we look for improvements in the subspace where $(k+1)$ variables will be unassigned (line 11). The algorithm stops when it reaches the maximal dimension size allowed or the timeout (line 4).

3.1.2 LDS+CP

LDS is a tree search method introduced by Harvey and Ginsberg [16] allowing to iteratively solve binary CSPs. Let H be a heuristic that is trusted. The main idea of LDS is to follow H when exploring the search tree, and to consider that H may make mistakes a small number (δ) of times. Thus, δ *discrepancies* are allowed during search. For a given maximal number δ_{max} of discrepancies, LDS explores the tree in an iterative way with an increasing number of *discrepancies* (from $\delta = 0$ to $\delta = \delta_{max}$). Depending on the value of δ_{max}, LDS is either a partial or a complete tree search. In [23], LDS has been extended to n-ary optimization problems, and only performs the last iteration (for $\delta = \delta_{max}$).

Our *variable ordering* for LDS first selects the variable having the lowest ratio domain cardinality divided by its degree (`Dom/Deg`). Our *value ordering* (`BestFirst`) selects the values according to the increasing order of their violation costs. We re-use information gained from the filtering of soft global constraints to determine the

violation cost for a value. Finally, Constraint Propagation is performed using soft global constraints filtering (see [17,25]).

3.2 Neighborhood Heuristics: Related works

Neighborhood heuristics are crucial since they select parts of the search space to explore in order to find solutions of better quality. However, designing efficient neighborhood heuristics is a hard task and requires a great deal of expertise. Morever, as quoted in [8], few neighborhood heuristics have been designed for NRPs. In this subsection, we review these neighborhood heuristics and describe the context in which they have been used.

(i) (VNS). In [6], three neighborhood heuristics based on swapping large parts of nurse plannings have been proposed and used in a VNS scheme:

- *Shuffle neighborhood* considers different swaps between the worst nurse planning and any other nurse planning.
- *Greedy Shuffle neighborhood* considers swaps between any two nurse plannings.
- *Core Shuffle neighborhood* considers two consecutive swaps between any two nurse plannings at a time (see [6] for more details).

(ii) (VNS+HO). A hybrid method combining VNS with a heuristic ordering (HO) has been proposed in [7]. The aim of the heuristic ordering is to sort all the shifts by their estimated difficulty for assigning them or how likely they are to cause high penalties. First, an initial planning is built using the heuristic ordering. Second, in order to improve the initial planning, a VNS is performed, followed by a repair phase. This phase selects the worst individual plannings, unassigns their shifts and reassigns them using the heuristic ordering. This process is repeated until a stopping criterion is reached. Two kinds of neighborhoods heuristics have been proposed:

- *One-shift Swap*: re-assigning a shift to another nurse working on the same day.
- *Two-shift Swap*: swapping a pair of shifts assigned to two nurses working on the same day.

(iii) (VNS+CP). A 2-steps hybrid Constraint Programming approach has been proposed in [32]. First, a constraint satisfaction model is used to generate weekly plannings of high quality satisfying a subset of shift sequence constraints. An iterative forward search is then used to combine them in order to build feasible solutions over the whole scheduling period (4 weeks). Second, VNS combined with the neighborhood heuristics described in (i) is used to quickly improve obtained solutions.

(iv) (VNS+IP). VNS has been used as a postprocessing step in [10] to make refinements on solutions found by an Integer Program (IP). Proposed neighborhood heuristics are based on swapping groups of consecutive shifts and are very close to the *Greedy Shuffle neighborhood* heuristic decribed in (i).

(v) (LNS). More recently, a LNS (Large Neighborhood Search [34]) scheme has been used to tackle NRPs [17]. It proceeds by selecting fragments of nurse plannings to be unassigned and then rebuilding them using F iltering. Such a use of LNS can be considered as an instance of VNDS. Three neighborhood heuristics have been proposed in [17]:

(a) *Sliding window with a fixed length*: Nurse plannings are selected over a sliding window (i.e. covering fixed days of the roster of all nurses) of one week.

(b) *Sliding window with an overlap*: It is a refinement of heuristic (a) by selecting variables involved in the pattern constraints for which a part of variables is on the boundary of the sliding window, whereas another part is outside the sliding window.
(c) *Detecting regions of low quality*: Instead of selecting all nurse plannings like for heuristics (a) and (b), only nurse plannings of low quality are considered. Moreover, PGLNS [30] is used to determine the size of the sliding window and the set of variables to be unassigned according to the information gained by filtering (see [30] and [17] for more details).

3.3 Neighborhood Heuristics: Our proposal

Neighborhood heuristics based on swap cannot be combined with our VNDS approach which requires an unassigning step and a rebuilding step. Moreover, we haven't used heuristics (a), (b) and (c) described Section 3.2 for two main reasons. First, selecting all nurse plannings is only effective for small problems. For large problems, as neighborhoods size can quickly grow, the exploration of (very) large neighborhoods may require a too expensive effort. Second, as a lot of soft global constraints are stated over the whole planning of a nurse, unassigning only the subset of variables that appear in the sliding window will not lead the rebuilding step to find a new solution of better quality. Indeed, the more the variables are linked, the more opportunities for the rebuilding step to minimize violations.

All variables related to a nurse planning will be together unassigned. For our approach, k will represent the number of nurse plannings to be unassigned (and not the number of variables to be unassigned as depicted in general Algorithm 1). We have considered the following three heuristics:

- `rand` randomly selects nurse plannings,
- `maxV` selects nurse plannings having high violation costs,
- `dilution` combines the two previous heuristics. Among the k nurse plannings to be unassigned, half of them are selected using `maxV`, and the other ones will be chosen randomly. The idea is to mix *intensification* phases (by considering nurse plannings with high violation cost) with *diversification* phases (by considering nurse plannings randomly in order to escape from local minima).

4 Experimental protocol

The ASAP site (Automated Scheduling, optimization And Planning) of University of Nottingham (`http://www.cs.nott.ac.uk/~tec/NRP/`) records a large and various set of NRPs instances as well as the methods used to solve them.

We performed experiments over different instances we selected in order to be representative of the diversity and the size of NRPs (see Table 1). For each instance, **we always compare our approach with the best methods** for solving it [18]. As experiments have been run on various machines, we will report, for each instance, the original CPU time and the processor. For all instances, except the first three ones where the processor is too old to be normalised (they are noted in italic Table 1), CPU times will be normalised[1] and denoted `CPUN`.

[1] For a machine κ times slower than ours, reported CPU times will be divided by κ.

Instances	$\|I\|\times\|J\|$	$\|D\|$	Optimum	Ad hoc methods			VNS/LDS+CP	
				Algo.	**Cost**	**Time(s)**	**Cost**	**Time(s)**
Ozkarahan	14×7	3	0⋆	[29]	-	-	**0**	1
Millar	8×14	3	0⋆	Network	2550	*500*	**0**	1
				TS+B&B	**0**	1		
Musa	11×14	2	175⋆	[28]	199	*28*	**175**	39
LLR	26×7	4	301⋆	TS+CP	366	16	312	275
BCV-5.4.1	4×28	5	48⋆	Hybrid TS	**48**	5	**48**	1
				VDS	**48**	128		
Valouxis	16×28	4	20⋆	VDS	60	32450	60	6570
				SS+VDS	100	6000		
Azaiez	13×28	3	0⋆	(0,1)-LGP	**0**	150	**0**	233
GPOST_A	8×28	3	5⋆	SS+VDS	9	6457	8	474
				MIP [14]	**5**	1285		
GPOST_B	8×28	3	3⋆	SS+VDS	5	5932	4	989
				2-Phases	**3**	14		
				MIP [14]	**3**	441		
ORTEC_01	16×31	5	270⋆	GA [7]	775	3600	355	6818
					681	86400		
				VNS+HO [7]	706	3085		
					541	37020		
				VNS+IP [10]	460	2571		
				VDS	355	14359		
					280	51420		
				MIP [14]	**270**	120		
Ikegami 3Shift-DATA1	25×30	4	2⋆	TS+B&B	6	111060	63	671

Table 1 Best results for Ad hoc methods vs best results for VNS/LDS+CP.

Some methods include a pre-treatment. As CPU times for this step are not given in papers, reported CPU times concern in fact the second step. In our approach, we use LDS, combined with filtering of soft global constraints, to generate the initial solution. So, reported CPU times for our method always **include the computing time for obtaining the initial solution**.

Benchmarks we considered (see Table 1) represent a wide variety of NRPs with non-trivial properties which are derived from real world complex instances. They are significantly different from each other by the number of nurses (ranging from 4 to 26), the number of shift types (ranging from 2 to 5), the duration of the planning period (ranging from 7 to 31 days) and the constraints to be verified: Shift constraints, Pattern constraints and Workload constraints (see Section 2.1). Finally, they may also differ by the number of personal requests and preferences.

Each instance has been solved by VNS/LDS+CP using neighborhood heuristics `rand`, `maxV` and `dilution`. k_{min} has been set to 2 and k_{max} to 66% of the total number of nurses. Timeout has been set according to the size of each instance. For heuristics `rand` and `dilution`, a set of 10 runs per instance has been performed. VNS/LDS+CP has been implemented in C++. Experiments have been performed under Linux on a 2.8 GHz P4 processor, with 1GB RAM.

5 Experimental results

5.1 Comparing with ad hoc methods

5.1.1 `Ozkarahan` *instance [29]*

We find the optimum in less than 1*s. using* `maxV`*.*

5.1.2 `Millar` *instance (2 methods)*

- *B1) Network programming* [26]: All feasible weekly shift patterns of length at most 4 days are generated. Then, an acyclic graph is defined, where nodes are the stretches, while arcs represent feasible transitions between stretches. Costs are associated to the transitions in order to reflect their desirability. The model is solved using CPLEX.
- *B2) TS+B&B* [19]: Nurse constraints are used to produce all feasible shift patterns for the whole scheduling period for each nurse (independently from shift constraints). Best combinations of these shift patterns are found using mathematical programming and Tabu Search.

With B1, a solution of cost 2,550 is found after 500 s. on an IBM RISC6000/340. With B2, a solution of cost 0 is obtained in 1 s. on a 1GHz Intel P3 processor. *We find the optimum in less than* 1 *s. using* `maxV`*.*

5.1.3 `Musa` *instance [28]*

A solution of cost 199 is found in 28 s. on UNIVAC-1100. *We find the optimum (cost 175) in* 39 *s. using* `maxV`

5.1.4 `LLR instance`

A hybrid AI approach (TS+CP), which combines Constraint Propagation and Tabu Search is used in [22]. First, a relaxed problem which only includes hard constraints is solved as a CSP. Second, adjustments with local search and tabu search is then applied to improve the solution. A solution of cost 366 is found after 96 s. on a PC/P-545MHz (`CPUN` 16 s.). *With* `rand`*, we obtain (on average) a solution of cost* 316.1 *after* 600 *s. The best solution (over the 10 runs) has a cost* 312 *(*275 *s.). The first solution (cost* 363*) is obtained in less than* 1 *s.*

5.1.5 `BCV-5.4.1` *instance (2 methods)*

All the results are obtained on a same machine (2.66GHz Intel P4 processor). Hybrid Tabu search [4] is the best of the 2 methods for this instance. The optimum is found in 5 s. (`CPUN` 5 s.). *With* `dilution`*, we obtain the optimum after* 1 *s.*

5.1.6 `Valouxis` *instance*

This instance [36] is described Section 2.2. In [8], Variable Depth Search (VDS) obtains a solution of cost 60 (3 workstretches of length 3) after 23,175 s. on a 2.66GHz Intel Core2 Duo processor (`CPUN` 32,450 s.). VDS works by chaining together single swaps of shifts among nurse plannings. Several heuristics are used to select the swaps to be chained in order to escape from local optima. In [9], VDS has also been used as an improvement method in the Scatter Search (a population based optimisation method). On this instance, (SS+VDS) obtains a solution of cost 100 in 4,000 s. on a 2.83GHz Intel Core2 (`CPUN` 6,000 s.).

We obtain a solution of cost 60 *(*3 *workstretches of length 3) after* 6,570 *s. using* `rand` *(see Figure 1).*

5.1.7 `Azaiez` *instance*

An optimal solution is provided with the (0,1)-Linear Goal Programming method [2] after 600 s. on a PC/P-700MHz (`CPUN` 150 s.). `rand` *(resp.* `maxV`*) finds the optimum in* 233*s. (resp.* 1,050 *s.).*

5.1.8 `GPOST` *(2 instances)*

The first instance, `GPOST_A`, has an optimal solution of cost 5. The optimum has been found in 1,320 s. using MIP (Mixed Integer Programming) on a P4 2.66GHz (`CPUN` 1,285 s.) This approach [14] takes advantage of the structure of the problem in order to derive new pattern rules to allocate particular shifts e.g. Night shifts. Such propagations drastically reduce the size of the search space. On this instance, (SS+VDS) [9] obtains a solution of cost 9 in 4,305 s. (`CPUN` 6,457 s.).

We find a solution of cost 8 *in* 474 *s. using* `dilution`.

The second instance, `GPOST_B`, is a relaxed version of `GPOST_A` where nurse requests have been removed. For this instance, three approaches have been proposed:

- the same MIP approach [14] finds an optimal solution in 420 s. (`CPUN` 441 s.).
- a 2-steps method [18]. First, all feasible plannings are enumerated for each nurse. Then, the final planning is generated using CPLEX. This method obtains an optimal solution in 8 s. on a 2.83GHz Intel Core2 Duo processor (`CPUN` 14 s.) without taking into account the time used in the first step.
- (SS+VDS) [9] obtains a solution of cost 5 in 3,955 s. (`CPUN` 5,932 s.).

We find a solution of cost 4 *in* 989 *s. using* `rand`.

5.1.9 `Ikegami-3shift-DATA1` *instance*

Experiments have been performed on a P3 1GHz. TS+B&B [19] finds a solution of cost 10 after 543 mns (`CPUN` 194 mns) with a timeout of 24h and a solution of cost 6 after 5,783 mns (`CPUN` 1,851 mns) with a timeout of 100h. `maxV` *provides a solution of cost* 63 *(where all unsatisfied constraints are of weight* 1*) after* 671 *s. with a timeout of* 1*h.*

Contrary to other instances, nurse constraints are hard ones and shift constraints are soft ones for `Ikegami`. So our neighborhood heuristics which unassign whole nurse

Instance	Opt.	First Sol.	rand			maxV		dilution			timeout
			best	time	avg.	best	time	best	time	avg.	
Millar	0	4800	0	2	0	0	1	0	1	0	300
BCV-5.4.1	48	69	48	5	48.8	48	202	48	1	48.6	300
LLR	301	363	312	275	316.1	337	385	315	440	321.3	600
Valouxis	20	37240	60	6570	132	160	3780	60	7160	102	7200
GPOST_A	5	7876	8	654	11.4	14	1252	8	474	11	1800
GPOST_B	3	7362	4	989	8.5	1365	44	5	1701	8.1	1800

Table 2 Comparing heuristics `rand`, `maxV` and `dilution` on several instances.

plannings are irrelevant. If the timeout is increased, the solution quality is improved but it is not enough to bring the optimum. As it is more efficient to unassign variables related to soft constraints than hard ones, one may consider that basic heuristics unassigning shift constraints would be efficient. But it is not the case as it is very difficult to obtain a first solution: the number of nurse constraints is greater than the number of shift ones.

5.1.10 First results for ORTEC_01

The ORTEC_01 instance is a benchmark from ORTEC's Harmony software, an international consultancy company in planning, optimization and decision support solutions. This instance is a large and difficult one. Several approaches have been used to solve it:

- The MIP approach [14] finds an optimal solution in 120 s. (CPUN 120 s.).
- (VNS+HO) [7] finds a solution of cost 706 in 1 h. on a P4 2.4GHz (CPUN 3,085 s.). The same method finds a solution of cost 541 in 12 h. (CPUN 617 mns).
- (VNS+IP) [10] finds a solution of cost 460 in 3,000 s. on a P4 2.4GHz (CPUN 2,571 s.).
- VDS finds a solution of cost 355 in 16,755 s. (CPUN 14,359 s.) and a solution of cost 280 in 60,000 s. (CPUN 51,420 s.) on a P4 2.4GHz.

For a timeout set to $7,200$ s., *we find a solution of cost* 355 *in* 6,818 *s. using* `rand`*, and a solution of cost* 375 *in* 4,231 *s. using* `dilution`*.* More experiments have to be performed to confirm and improve these promising results.

5.2 Comparing our neighborhood heuristics

Table 2 compares the results produced by our neighborhood heuristics (i.e. `rand`, `maxV` and `dilution`) on different instances. For each instance, the cost of the best solution found, its computation time and average solutions over 10 runs are reported. The cost of the first solution we obtained is also recorded in the third column. We can draw some remarks:

- On average, `dilution` outperforms both `rand` and `maxV`, except for LLR, where `rand` is the best one. Indeed, as two consecutive days off get a penalty of 5 and as there are two nurses which require one week day off in their planning leading to a higher violation cost (i.e. 30), heuristics `maxV` and `dilution` will almost select these two nurses, while `rand` will enable to escape from such *local optima*.

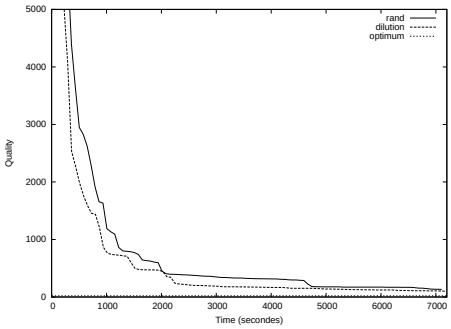


Fig. 2 `Valouxis` Instance, optimum=20

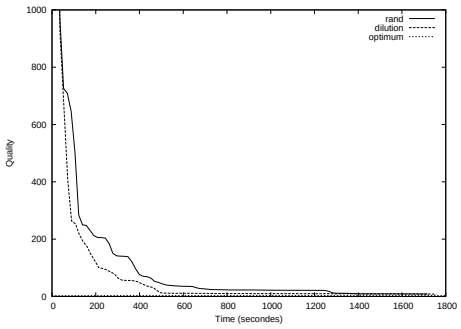


Fig. 3 `GPOST_B` Instance, optimum=3

– For the best reported results, `dilution` and `rand` perform similarly, both in solution quality and computing time. Indeed, when k becomes sufficiently large, the two heuristics tend to have very similar neighborhoods.
– `maxV` is the less effective heuristic. This is probably due to its deterministic criterion, which leads the heuristic to be stucked in local minima. Focussing only on the worst nurse plannings will rarely improve the quality of the overall planning. So, using some randomness enables *diversification*.

The performance profile of a method describes the evolution of the quality of obtained solutions as a function of computation times. Fig. 2 and Fig. 3 depict the performance profiles of VNS/LDS+CP for `Valouxis` and `GPOST_B` instances. As `maxV` is the less effective heuristic, (average) results are only reported for `dilution` and `rand`.

On `Valouxis` instance (see Fig. 2), `dilution` enables to quickly improve the quality of the solution during the search. At the beginning, the performance profile of `dilution` is very close to that of `rand`. But after a few seconds of computation (60 s.) `dilution` always provides solutions of better quality, thus clearly outperforming `rand`. This behavior can be explained by the fact that `dilution` benefits from information provided by `MaxV` tto improve nurse plannings having a high violation cost, but without selecting them all the time.

On `GPOST_B` instance (see Fig. 3), the same conclusions can be drawn: first, solution quality improvements are larger at the early stages of computation (property of *diminishing returns*), particularly during the time interval of $[0 \ldots 1{,}000$ s.]. Second, the two curves show a decelerating phase leading to a quasi-plateau.

6 Conclusion

For each instance, we have compared our method with the best ad hoc method for solving it [18]. Despite its genericity and flexibility, our method has obtained:

– solutions of better quality and better computing times for `Ozkarahan`, `Millar`, `Musa`, `LLR`, `BCV-5.4.1`, and `Valouxis` ;
– solutions of equal quality with computing times close to those for `BCV541` and `Azaiez`,
– *very promising solution quality* on large scale instances as `GPOST_A`, `GPOST_B` or `ORTEC_01`.

For large instances as `ORTEC` or `Montreal`, or very specific ones as `Ikegami`, performances of our method could be greatly improved by i) using neighborhood heuristics especially designed for NRPs, and ii) reducing the lack of communication between soft global constraints by extending arc consistency for soft binary constraints [11, 12, 21]. In 2009, [21] has shown for the first time that dedicated cost function filtering techniques can also be used to define Global Cost Functions leading to important speedups compared to the use of global constraints with cost variables.

References

1. H. Meyer auf'm Hofe. Solving rostering tasks as constraint optimisation. In *PATAT'00*, LNCS 2079, pages 191–212, 2001.
2. M. Azaiez and S. Al Sharif. A 0-1 goal programming model for nurse scheduling. *Computers and Operations Research*, 32(3):491–507, 2005.
3. S. Bourdais, P. Galinier, and G. Pesant. HIBISCUS: A constraint programming application to staff scheduling in health care. In *CP'03*, volume 2833 of *LNCS*, pages 153–167, 2003.
4. E. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In *2nd Asia-Pacific Conference on Simulated Evolution and Learning*, volume 1585 of *LNCS*, pages 187–194, 1999.
5. E. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
6. E. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Variable neighborhood search for nurse rostering problems. In *Metaheuristics: computer decision-making*, pages 153–172, Norwell, MA, USA, 2004. Kluwer Academic Publishers.
7. E. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008.
8. E. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A time predefined variable depth search for nurse rostering, TR-2007-6, University of Nottingham, 2007.
9. E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A scatter search methodology for the nurse rosetering problem. *Journal of the Operationnal Research Society*, pages 1–13, 2009.
10. E. K. Burke, Ji. Li, and R. Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484–493, 2010.
11. M. Cooper, S. de Givry, M. Sanchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for Weighted CSP. In *AAAI'08*, 2008.
12. M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.
13. A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *EJOR*, 153(1):3–27, 2004.
14. C. A. Glass and R. A. Knight. The nurse rostering problem: A critical appraisal of the problem structure. *EJOR*, 202(2):379–389, 2010.
15. P. Hansen, N. Mladenovic, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.
16. W. Harvey and M. Ginsberg. Limited Discrepancy Search. In *IJCAI'95*, pages 607–614, 1995.
17. F. He and R. Qu. Constraint-directed local search to nurse rostering problems. In *6th International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS 2009), CP'09*, pages 1–12, 2009.
18. *http://www.cs.nott.ac.uk/˜tec/NRP/*.
19. A. Ikegami and A. Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.
20. R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
21. J. Lee and K. Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In *IJCAI'09*, pages 559–565, 2009.

22. H. Li, A. Lim, and B. Rodrigues. A hybrid AI approach for nurse rostering problem. In *SAC*, pages 730–735, 2003.
23. S. Loudni and P. Boizumault. Combining VNS with constraint programming for solving anytime optimization problems. *EJOR*, 191(3):705–735, 2008.
24. J.-P. Métivier, P. Boizumault, and S. Loudni. Softening Gcc and Regular with preferences. In *24th annual ACM Symposium on Applied Computing*, pages 1392–1396, University of Hawaii at Manoa, USA, March 2009.
25. J-P. Métivier, P. Boizumault, and S. Loudni. Solving nurse rostering problems using soft global constraints. In *15th Int. Conf. on Principles and Practice of Constraint Programming*, volume 5732 of *LNCS*, pages 73–87, Lisbon, Portugal, 2009.
26. H. Millar and M. Kiragu. Cyclic and non-cyclic sheduling of 12h shift nurses by network programming. *EJOR*, 104(1):582–592, 1996.
27. N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & OR*, 24(11):1097–1100, 1997.
28. A. Musa and U. Saxena. Scheduling nurses using goal-programming techniques. In *IIE transactions*, volume 16, pages 216–221, 1984.
29. I. Ozkarahan. The zero-one goal programming model of a flexible nurse scheduling support system. In *Int. Industrial Engineering Conference*, pages 436–441, 1989.
30. L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In *Proceedings of CP'04, LNCS 3258*, pages 468–481, 2004.
31. T. Petit, J-C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *CP'01*, volume 2239 of *LNCS*, pages 451–463, 2001.
32. R. Qu and F. He. A hybrid constraint programming approach for nurse rostering problems. In *28th SGAI International Conference on A.I.*, pages 211–224, 2008.
33. J-C. Régin, T. Petit, C. Bessière, and J-F. Puget. An original constraint based approach for solving over-constrained problems. In *CP'00*, volume 1894 of *LNCS*, pages 543–548, 2000.
34. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP'98*, volume 1520 of *LNCS*, pages 417–431, 1998.
35. H. Simonis. Models for global constraint applications. *Constraints*, 12(1):63–92, 2007.
36. C. Valouxis and E. Housos. Hybrid optimization techniques for the workshift and rest assignment of nursing personnel. *A.I. in Medicine*, 20(2):155–175, 2000.
37. W. van Hoeve, G. Pesant, and L-M. Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4-5):347–373, 2006.

An efficient and robust approach to generate high quality solutions for the Traveling Tournament Problem

Douglas Moody, Graham Kendall and Amotz Bar-Noy

City University of New York Graduate Center and School of Computer Science (Moody,Bar-Noy), University of Nottingham, UK (Kendall)

dmoody@citytech.cuny.edu, gxk@cs.nott.ac.uk, amotz@sci.brooklyn.cuny.edu

The Traveling Tournament Problem (TTP) describes a typical sports scheduling challenge. The TTP, which is based on the U.S. Major League Baseball (MLB), has specific instances with results available on the web. Several approaches have been proposed since the problem's creation. The "best" of these solutions use extensive resources in local search activities to find high quality solutions. We propose a tiling method that can produce a good quality solution, using a fraction of the resources documented in other approaches. Our solution can also be expanded to handle the additional real-world scheduling requirements, including unbalanced schedules within the MLB.

1 Introduction

The Traveling Tournament Problem (TTP) is a double round robin tournament to be played by n teams over $(2n-2)$ periods or weeks, where each team plays in every period (we do not consider the "mirrored" version of the problem). The three constraints of the TTP are:

1) Maximum "**Road Trip**" of three games: each team can play at most three consecutive games away from the team's home site before playing again at the home site. For teams beginning the season with an away game, it is assumed travel for that game would begin at the home site. Likewise, teams ending the season with an away game, would require to travel from that opponent's location to the team's home site to complete the season.

2) Maximum "**Home stand**" of three games: each team can play at most three consecutive games at its home site.

3) Repeater Rule: a team cannot play an opponent away in time period k and then home in time period $k+1$, or vice versa.

The TTP seeks to minimize the distance traveled by each team. A distance matrix is available to define the distance between each team. This calculation is used for each road trip, with the total being the accumulated distance for all teams. The selection of opponents and their order within

the road trip is critical, while home stands have no bearing on the distance calculation. Efficient road trips across all teams are more likely to yield good quality solutions to the TTP.

The TTP has served as a benchmark problem for sports scheduling over the past decade. Easton et al. [6] defines the problem, and the latest results are available at [14]. However, the TTP definition notes its simplification of the actual Major League Baseball. This simplification eliminates some key complexities of the MLB, notably:

Existence of unbalanced schedules – Teams in MLB play other teams within their division more frequently than teams in other divisions.

Unequal home and away games – Due to Inter-league play, teams do not play an equal number of home and away games against all teams. It also introduces constraints of teams from the same city playing on the same day. Kendall [7] notes this constraint in scheduling the English football league.

These two simplifications enable many solution approaches to take advantage of round-robin tournament scheduling for the TTP. The solutions that depend on a mirrored approach would not be able to produce an unbalanced schedule. Other approaches use simple tournament generators that also would not handle an unbalanced schedule. We propose an approach that can compete with existing TTP tournament based solutions sets, and also be extendable to handle the real MLB problem.

2 Related Work

The TTP has spawned a variety of methodologies to find good quality solutions. Kendall et al. [8] surveys this broad array of approaches. The work to date can be viewed in two phases. The first phase consists of traditional approaches including simulated annealing by Anagnostopoulos [2], Lims' [11] work on integer programming, the constraint programming approach by Leong [10] , and tabu search proposed by Di Gaspero and Schaerf [5]. Also special heuristics have been developed as used by Shen and Zhang [13]. The second phase focuses on the use of parallel processing and server farms. Van Hentenryck and Vergados [16], and Araujo [1] use a parallel implementation of existing algorithms to improve on published results.

The majority of the above methods have focused almost exclusively on local search operations to produce the best solutions. The initial neighborhood for the local search is usually generated at random, and is not necessarily feasible. Hence the starting "neighborhood" may be extremely far from the optimal solution, in terms of a distance measured by the number of "swaps", which are defined as moves between neighborhoods.

Our approach seeks to create an initial neighborhood that is a feasible solution closer to the optimal solution than random initial solutions. We use a tiling method, proposed by Kingston [9] for course scheduling. Our tiling method creates an initial solution that has minimal hard con-

straint violations, and contains a core set of game assignments for a high quality solution. The second phase of local search is used to remove constraint violations, and tune the solution.

The concept of modeling the problem using away trips was also considered by Trick [15]. A new variable, representing a road trip is introduced into the model. The road trip variable is constructed as game assignments are made. This variable affects the value of the objective function, and hence the road trips are modified to reduce the objective function. Hence many road trips are considered for a given team during the running of the model. This approach differs from our approach, as we begin the scheduling process with existing road trip (tile) formations. During our scheduling phase, we do not re-formulate a tile, based upon other assignments. We may break a tile into its component parts, but these parts are not reassembled into another tile. Our approach is based on a fixed set of tiles.

3 Proposed Approach

Our approach is a two phase approach involving tiling, followed by a local search phase. The tiling phase creates an initial solution, which may not be feasible. The second phase (a local search) removes the hard constraint violations and improves the quality of the solution.

3.1 Tiling

We model the road trips of the TTP as "tiles". Each tile contains "blocks", which represent individual games. A road trip of three opponents is considered as one tile, with three blocks. A teams' schedule can be thought of as a series of tiles, with home games as spacers between the tiles. Figure 1 shows the scheduling grid and tiles for Team 1 and Team 2.

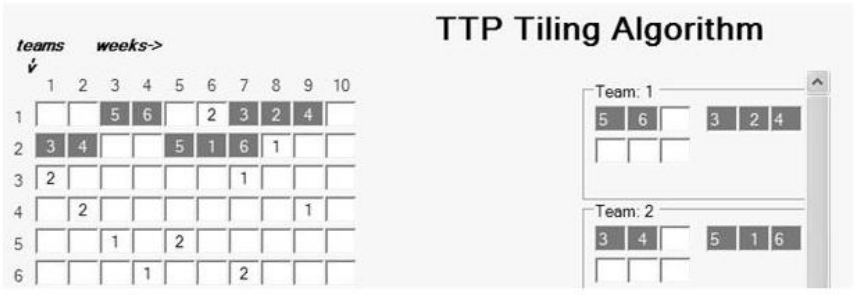


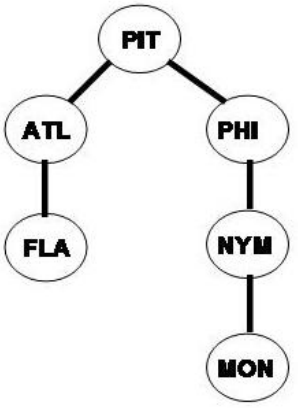
Fig. 1. Tile Placement for Teams 1 and 2 (shaded cells indicate an away game)

For each team a set of tiles is created that seeks to minimize the distance traveled for a particular team, without taking into account any constraints involving other teams. These tiles are placed in a scheduling grid of n rows representing teams and $(2n\text{-}2)$ columns representing weeks.

As tiles are placed, other cells of the grid are filled in to maintain schedule consistency with the tile placement. When no more tiles can be

placed, the tiles are broken into their component blocks, and placed subject to TTP constraints. If all blocks cannot be placed, the consecutive home and away constraint is relaxed, allowing all blocks to be placed albeit in an infeasible solution.

The creation of the tiles is carried out on a team by team basis. For each team a minimum spanning tree (MST) is created by using the Prim [4] algorithm. The algorithm begins with the selection of a root node, in our approach this is a team. All distance edges in the distance matrix, defined in the problem, are searched for the smallest edge. Each edge has a vertex of a team in the tree, and a vertex of a team not in the tree. This edge is then added between the two teams. For the first branch, we are finding the nearest opponent of the root team. The second edge is the nearest team to the root team, or its first opponent. Edges that are added to an opponent will suggest that the two vertices or teams will be on the same road trip or tile. Two edges having the same root node as a vertex, suggest that the two opponents of the root team will be on different road trips. Figure 2 presents an example MST for the team Pittsburgh (PIT) in the NL6 problem documented in [14].



DISTANCE MATRIX

Home Team \ Away Team	ATL	NYM	PHI	MON	FLA	PIT
ATL	0	745	665	929	605	521
NYM	745	0	80	337	1090	315
PHI	665	80	0	380	1020	257
MON	929	337	380	0	1380	408
FLA	605	1090	1020	1380	0	1010
PIT	521	315	257	408	1010	0

Example: Distance from Team NYM to Team PHI is 80 miles

Fig. 2. Minimum Spanning Tree (MST) for PIT in NL6 and the accompanying distance matrix.

Figure 2 suggests that the team PIT should have 2 road trips or tiles – one with ATL and FLA, and the other with PHI, followed by NYM and MON. If the team completes these two road trips, the team may have travelled the optimal minimum distance. This does not imply the league overall will have optimal minimal distance, but rather just this team.

We use a tree collapsing algorithm to create tiles from the tree structure. Separate trees are created with each tree having the root node of a team. The collapsing approach merges child nodes into their parent node. When the parent has three teams, a tile is created. When the parent must decide among its children, a greedy method is used, to pick the best set of three teams from the parent and children. Figure 3 provides a sample collapsing process.

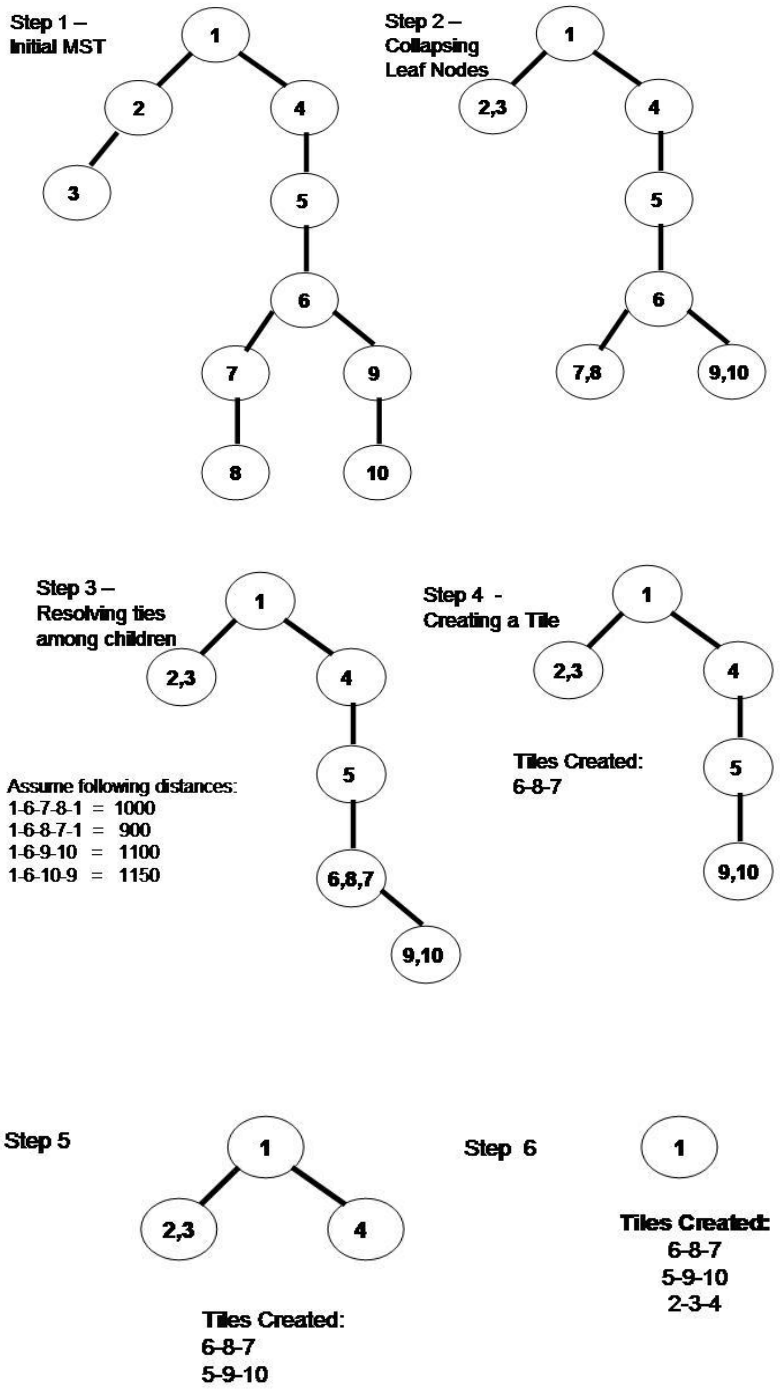


Fig. 3. Creation of Tiles through collapsing of the tree

The three team tile creation process creates *ceiling((n-1)/3)*n* tiles. Note that all tiles have three blocks, with the possible exception of the last tiles created for the team. At the root node, if both children have a weight of two nodes, two tiles of two blocks are created for each child.

After tile creation, each tile is given a cost. This cost is calculated to capture some measurement of the impact on the objective function of breaking up the tile. The cost is the sum of a round-trip of the home team to each opponent within the tile, minus the actual distance to be traveled between teams within the tile. This cost is then used to prioritize the order that tiles will be placed in the schedule.

We select the tile with the highest cost. We place this tile on the first available space on the schedule without violating any constraints. We start with week one, and move forward checking all constraints for the three games represented in the tile. If the tile cannot be placed in week 1, we rotate the tile, by changing the order of the first and last teams in the road trip. This rotation does not alter the distance associated with the tile. The tile is moved through the schedule week by week until a placement can be found.

The tile placement process is continued for each tile in cost priority order. When no more tiles can be placed we break all tiles into individual games. The games are then placed in the schedule, starting from the first week of the schedule. We relax the maximum away games and home stand rules as well as the repeater rule constraints at this point to allow for placement of all games. When a game cannot be placed, given these relaxations, previous assignments are backtracked. If the backtracking of the individual games does not produce a feasible schedule, then the last place tile is backtracked.

One noteworthy aspect of our approach is that tiles are never broken and then formed into new tiles, referred to as reconfiguration. This process involves breaking two tiles in their component blocks, and regrouping the blocks into two new tiles.

3.2 Local Search

The local search phase of our approach is designed to remove any hard constraint violations and improve the quality of the schedule. When it is possible to place all tiles, a near optimal solution based on the quality of our tile set can be generated. The breaking of the tiles is where our initial solution degrades. Hence, the games placed singularly are the root of all hard constraint violations, and higher than optimal travel distances. Our local search seeks to reach the best local minimum of our initial solution, after performing a variety of "swaps". Each swap moves a set of games within the schedule while maintaining, or improving, feasibility and the objective function of the solution.

We employ the following sets of swaps during our local search phase:

Home / Away Swap – We swap two games, involving the same two teams, where each team is home in one game and away in the other game. The swap is done between two weeks.

Round Swap – All games for two weeks are moved between the two weeks.

Partial Round Swap - The partial round swap described in [2], moves a set of connected games between two weeks. We begin this operation by selecting two games in different weeks. The teams in these games create our swap set. We then add teams playing these two teams, in either week, to the swap-set. The process continues until all teams, in both weeks, are in the swap set or its complement. We then move all games involving teams in the swap set between weeks. We can also choose to move all teams in the complement of the swap set between the weeks.

The local search phase is an iterative process to optimize the impact of the above swap actions. We look at each swap in the order above, and analyze the schedule improvement, if any, of each possible swap. The swap with the largest improvement is performed to reach a new schedule. If no improvement can be found, we move to the next type of swap in the list above. The same process is used until no improvement can be found. After performing all possible swaps of the last type (partial round), we have reached a local minimum and stop the local search.

4 Results

Our two phase approach of tiling and local search enables us to produce a high quality solution with minimal resources. We only use a small fraction of the resources consumed by the best known approaches. Our platform is a 2.13 Ghz Dell laptop, using a .NET software application.

We use three approaches for comparison, presented in Table A. The first approach, described by Van Hentenryck and Vergados [16] is a parallel processing approach, carried out on clusters of dual-processor blade servers. They use a simulated annealing based Traveling Tournament approach first proposed by Anagnostopoulos [2]. The second approach by Di Gaspero and Schaerf [5] uses tabu search. The final comparison is with Araujo et al. [1], using parallel processing. This approach is also a two phase approach, with a random initial neighborhood construction, followed by a greedy search phase to reach a local minimum to produce a mirrored solution. The local search phase is iterated after a perbutation of the initial neighborhood.

Instance	Tiling Results	Van Hentenryck and Vergados Results [16]	Di Gaspero and Schaerf Results [5]	Araujo et al. (GRASP) Results [1]	Average % Difference from Tiling
NL 16	317,764	**267,194**	279,465	285,614	14.54%
NFL 16	266,231	**235,930**	238,581	N/A	12.21%
NFL 18	339,822	**296,638**	N/A	299,134	14.08%
NFL 20	406,463	**332041**	352947	359,748	16.71%
NFL 22	482,374	**412,812**	439,626	418,086	13.90%
NFL 24	544,354	**463,657**	499017	465,491	14.34%
CON16	354	N/A	**328**	342	5.67%
CON18	466	N/A	**418**	432	9.64%
CON20	568	520	**521**	522	9.02%

Table A: Results Comparison (best solutions in **bold**)

Instance	Tiling Time	Van Hentenryck and Vergados Time – Best [15]	Di Gaspero and Schaerf Time – [5]
NL 16	38	1,815	51,022.4
NFL 16	35	2,220	N/A
NFL 18	105	3,120	N/A
NFL 20	135	6,750	N/A
NFL 22	150	8,100	N/A
NFL 24	320	5,490	N/A
CON16	18	N/A	19,665
CON18	22	N/A	33,979
CON20	23	N/A	46,579

Table B: Time Comparison in seconds

We choose instances with a higher number of teams and the constrained constant distance instances of the problem, for comparison. The instances with fewer teams have been addressed by a variety of algorithms, whereas instances with 16 teams or more have been addressed successfully by only the above approaches.

Table A compares the results and resources of our tiling approach with the best-known approaches. Our tiling approach comes withinn 10-16% of the objective function for the TTP for all approaches. The tiling approach produces slightly better results where the number of games to be played by each team is divisible by three. This situation allows all games for a team to be placed in 3-team tiles. Hence our 16 and 22 team instances are slightly better in comparison with the other approaches.

For the constrained instances, our approach is even closer than the higher NFL instances. The construction and costing of the tiles is not important, since all distances are constant. Hence each tile for a team has the same cost, when the tile is broken. Our 16 team instance, dictating 5

tiles of 3 teams per tile, provides our best result. One factor in this result is that the games for each team can be constructed into a set of tiles with 3 games each. This enables more tiles to be placed in the initial neighborhood, increasing the quality of the initial solution.

We use only a few seconds compared to the substantive time used in the other approaches. Table B compares our times with the published times of the approaches with comparable metrics.

The key difference between our tiling approach and the other approaches is the initial neighborhood. Our initial neighborhood is built based upon tiles, which are high quality partial solutions. Hence our approach saves the time used by others in the local search to develop a high quality initial solution. We follow our tiling phase with an efficient greedy approach to quickly develop a solution relatively close to the best known solutions.

The most illuminating result is the NL16 figure for Araujo et al. in Table A. The result of 285,614 was achieved by running the GRASP algorithm in sequential mode outlined in [1] for 5 days. This algorithm is similar to our local search, because of its greedy nature. Both algorithms explore all possible swaps of a given category and perform the swap with the greatest improvement. When no improvements can be made, the local minimum is reached. Since the local search phases are similar, the key difference between the approaches is the construction of the initial neighborhood. The results in the table show the results of using GRASP on neighborhoods created, over a 5 day time span. After each local minimum is reached, random swaps are done to create a new neighborhood. Hence large numbers of initial neighborhoods are created over the 5 day processing period. The best of these neighborhoods led to only an 8.5% improvement of creating one initial neighborhood solution through tiling.

5 Conclusion and further work

Our approach indicates substantial resource savings in finding good quality solutions for the TTP. Our initial neighborhood, along with a minimal local search phase can produce high quality solutions. In our future work we will compare solution results among the best approaches, to understand the distance of our initial neighborhood from the best known solution sets. This will enable us to identify the transformations needed to move from our initial neighborhood more directly to the best solutions.

We also plan to expand the TTP to handle a complete Major League Baseball schedule, which was the original motivation for the problem. Albeit the actual MLB schedule has a wide variety of real-world constraints, we look to expand the TTP by implementing the actual unbalanced team schedules and inter-league play of the MLB master schedule. These instances will greatly increase the complexity of the schedule due to its unbalanced nature.

References

1. Araújo, A., Boeres, M.C., Rebello, V.E., Ribeiro, C.C., Urrutia, S.., Exploring Grid Implementations of Parallel Cooperative Metaheuristics, A Case Study for the Mirrored Traveling Tournament Problem. Chapter 16, Metaheuristics Volume 39, US:Springer, 2007.
2. Anagnostopoulos A., Michel L., Van Hentenryck P., Vergados Y. A simulated annealing approach to the traveling tournament problem. Journal of Scheduling 2006;9:pp. 177–93.
3. Bar-Noy, A. and Moody, D. "A Tiling Approach for Fast Implementation of the Traveling Tournament Problem," Practice and Theory of Automated Timetabling (PATAT06, Brno, August 2006), Conference Proceedings, pp. 351-358.
4. Cormen, Thomas H. , Leiserson, Charles E., Rivest Ronald L., Stein, Clifford, Introduction to Algorithms. pp. 570-573.Boston: McGraw-Hill, 2001.
5. Di Gaspero L, Schaerf A. A composite-neighborhood tabu search approach to the traveling tournament problem. Journal of Heuristics 2007;13:pp. 189–207.
6. Easton K., Nemhauser G.L., Trick M.A. The travelling tournament problem: description and benchmarks. In: Walsh T, editor. Principles and practice of constraint programming. Lecture notes in computer science, vol. 2239. Berlin: Springer; 2001. pp. 580–5.
7. Kendall G. Scheduling English football fixtures over holiday periods. *Journal of the Operational Research Society 2008*; 59:743–55.
8. Kendall G., Knust S., Ribeiro C. C. and Urrutia S. Scheduling in Sports: An Annotated Bibliography. Computers & Operations Research (2009), 37: pp.1-19
9. Kingston, J. A tiling algorithm for high school timetabling, Proceedings Practice and Theory of Automated Timetabling (PATAT04, Pittsburgh, August 2004, USA).,Conference Proceedings, pp. 208-225.
10. Leong, G. (2003). Constraint programming for the traveling tournament problem.
    www.comp.nus.edu.sg/henz/students/gan_tiaw_leong.pdf
11. Lim, A., Zhang, X. (2003) Integer programming and simulated annealing for scheduling sports competition on multiple venues, Proceedings of the Fifth Metaheuristics International Conference (MIC 2003).
12. Ribeiro CC, Urrutia S. Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research 2007;179:pp. 775–87.
13. Shen, H., Zhang, H. (2004) *Greedy Big Steps as a Meta-Heuristic for Combinatorial Search*. The University of Iowa AR Reading Group, Spring 2004
14. Trick, M.A., Challenge Traveling tournament instances. Online document at http://mat.gsia.cmu.edu/TOURN/.
15. Trick MA., Formulations and Reformulations in Integer Programming. Lecture Notes In Computer Science; Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Volume 3524/2005 pp: 366-379. Springer 2005.
16. Van Hentenryck, P. and Vergados Y., "Population-Based Simulated Annealing for Traveling Tournaments", AAAI - National Conference on Artificial Intelligence, 2007

Youth Sports Leagues Scheduling

Douglas Moody, Graham Kendall and Amotz Bar-Noy

City University of New York Graduate Center and School of Computer Science (Moody,Bar-Noy), University of Nottingham, UK (Kendall)

dmoody@citytech.cuny.edu, gxk@cs.nott.ac.uk, amotz@sci.brooklyn.cuny.edu

1 Introduction

Youth sports are administered by governing bodies that determine sportsmanship rules, promote the sport, and organize youth participation. Organizations within these bodies may be towns, high schools, sports clubs with international affiliations (e.g., FIFA -Federation International de Football Federation), and religious groups. Each of the organizations sponsor teams in leagues, and provides a venue or fixture. For example youth leagues in the United States include: junior soccer leagues, Little League baseball, inter-scholastic high school basketball, and the Catholic Youth organization (CYO). Youth sports leagues are played worldwide. For example, Little League Baseball is played in 72 countries worldwide within 7,170 leagues, comprising over 2 million players [7].

A Youth Sports League (YSL) consists of divisions (see Appendix A for terms we use in this paper), which are sets of teams grouped by age, gender, and/or level of play. The number of teams in a division can vary, ranging from 4 to 20 teams. Each participant registers with the league to play the same number of games, regardless of division. The schedule for a division is often a round robin tournament followed by additional games against selected opponents from the division in order to meet each team's required number of games. This type of schedule is referred to as "unbalanced" since a team may play one opponent once more than another. The sharing of the organization's venue by its sponsored teams creates a dependency between the division schedules. Two of an organization's teams, possibly from different divisions, cannot host a game at the same time. Hence, the administrator must consider the schedule of all divisions, when creating the master league schedule.

The scheduling of youth sports leagues differs from the professional sports league problem, widely studied in scheduling literature and surveyed by Kendall et al. [6]. Professional sports involve a balanced schedule, with guaranteed availability of the venue. Youth sports leagues play unbalanced schedules, and teams from all divisions must share a venue. A YSL venue can support several games a day, whereas professional sports teams' venues typically only host one game in a day. This venue sharing creates a schedule dependency among all divisions. A professional league with 4 divisions comprising 12 teams, can be viewed as four separate and distinct round-robin tournaments. The same league structure in the YSL must be viewed as one schedule with 48 teams play-

ing an unbalanced schedule. Real world instances of youth sports can include 400 divisions involving 3500 teams, for example the Long Island CYO youth basketball.

In the following section, we will informally define the youth sports league problem presenting its hard and soft constraints, and the objective function. The subsequent section discusses related research in this area, followed by our tiling approach to address the problem. The last section discusses the availability of real-world problem instances and related future work.

2 Problem Definition

The YSL involves scheduling multiple divisions, each containing teams sponsored by a common set of member organizations, across a set of venues. All games are intra-divisional. A season consists of a specific number of games to be played by all teams in the league, regardless of the number of teams in a particular division. If the number of games in a season is 12, then teams in a five team division and an eight team division will all play exactly 12 games. Some divisions are required to play an unbalanced round-robin tournament, where a team will play various opponents a different number of times.

Each organization makes a venue available to the league. The venue has a daily capacity or number of games that can be played on one day. A league may allow games to be played on more than one day of the week, particularly weekends. A season consists of a set of consecutive weeks, to be played on a given number of days per week. A venue's season capacity is the number of games per day that can be hosted by the venue, multiplied by the available days in the season. Divisions also have an associated referee level, indicating the minimum level of the referee's certification. This level is referred to as the referee category. A referee must have the proper level of certification to officiate the game. Referees may choose to officiate games at a lower level than their certification based upon their personal preference. This preference may stem from a goal of providing more on-field rule instruction to the younger age groups. However, in this instance the referee would be assigned a lower level category for the season. Each referee is assigned a category based upon their certification and personal requests. Often, a referee is assigned two games at a venue on a given day. The referee prefers to have the games follow each other to minimize his or her time at the venue. Hence it is desirable to schedule games, requiring the same referee level in succession. A venue's daily schedule should have an even number of games for each referee category to support this concept. This will enable the scheduler to schedule for each referee. A referee may officiate games consecutively from different divisions, if the divisions' referee category for the consecutive games are identical.

The schedules for the individual divisions are combined into a master schedule. The master schedule must address all venue sharing and referee assignment constraints.

A master schedule is to be created for all divisions such that:

1. Teams must play exactly *g* games within *w* weeks, where *g* is the league wide number of games per team, and *w* is the number of weeks of play during the season.
2. Each division will play a multiple round-robin schedule. For divisions requiring unbalanced schedules, teams may play an opponent only one additional time than other opponents.
3. Teams can play multiple games in a week, but only one game per day.
4. All teams from the organization will play at their organization's venue, when designated as the home team.
5. Each venue has a fixed capacity of *s* games per day. This value differs by venue.

1.

The quality of the schedule is evaluated by calculating the number of penalty points within the schedule. Schedules with lower penalty points, are of higher quality. The following section lists the penalty points, also referred to as soft constraints.

1. A team playing more than one game in a week (4 penalty points per game). Teams may play only one game per day, but are permitted to play on multiple days. For example, a team having games on both weekend days would be assigned 4 penalty points.
2. A venue hosting an odd number of games with the same referee category during one day (2 penalty points per referee category). Venues with an odd number of games within the same referee category will prohibit a referee from officiating successive games at the venue on that day.
3. A team playing home (or away) games in consecutive weeks (1 penalty point per week). Teams prefer to alternate home and away games.

The YSL solution must be a feasible solution satisfying all the hard constraints. The objective function of the YSL is to minimize the penalty points of the soft constraints above. Regin [10] shows that for a round-robin tournament, it is not possible to eliminate all breaks, defined as consecutive home or away games in the schedule. Hence, the third soft constraint renders a schedule of zero penalty points for the YSL impossible.

3 Example YSL Instance

Our example problem instance involves three towns, referred to as A, B, C, each sponsoring one or more boys and girls teams, within the league. Table A shows the input parameters necessary for the YSL in our sample league. The boys division contains two teams from town A, and one team from B and C. The girls division has three teams from B, two teams from A and one team from C. Each town maintains one venue with a capacity of two games per day, on two days each week. The season should contain 8 games per team, over a 7 week season. Both divisions have the same referee category. Note that the boys division is a double round-robin tour-

nament followed by additional games causing an unbalanced schedule. The girls division has a single round robin tournament followed by additional games resulting in an unbalanced schedule. The two divisions share a venue provided by each town.

Table A provides the set of inputs needed to define a YSL problem instance. Each row contains an input parameter, or a set of information about the league's structure.

Input Parameter	**Value**	**Definition**
G	8	Nubmer of games to be played by all teams
W	7	Number of weeks in the season
Y	2	Number of days per week for play
V	3	Number of venues
D	{b,g}	Values identifying each division.
R	L1	Referee categories
T*v*	A(b-2,g-2) B(b-1,g-3) C(b-1,g-1)	Organizations and number of teams sponsored by division. For each organization, the number of teams per division identified in ***D*** is provided. The value is specified by Organization, with the division-team number for each team in parentheses
C*v*	{2,2,2}	Capacity for each venue. All venues can host 2 games per day.

Table A – Sample problem input parameters and data

Table B provides the rounds necessary to schedule the master schedule for this youth sports league. A team is referred to by its town (A,B or C), followed by the sequence number of the team within the town. The team reference of "A2" would indicate this team is the second team sponsored by town A. For the Boys division, the first round shown (A1-B1, A2-C2) has been placed in week 2 of the master schedule. Each team plays an unbalanced schedule. Team A1, in the boys division, plays B1 and C1 three times while only playing A2, twice. Each box of the master schedule represents a venue's games for a day, whose maximum is two in our example instance. In weeks, where only one day is used by a venue, the choice of the day is not consequential.

The penalty point calculations are shown at the bottom of Table C. Since the number of games in a season is less than the number of elapsed weeks, it is guaranteed that penalty points will be earned for each team due to the soft constraint of playing multiple games in a week. Since both divisions have the same referee category, venues hosting an even number of games, will not incur a penalty point for non consecutive games by referee category. For example, in week 2, A2-C1 was scheduled on the same day as A4-C2 to avoid having an odd number of games on each day.

Boys	**A1-B1** **A2-C1** **-2-**	**A1-A2** **B1-C1** **-4-**	**C1-A1** **B1-A2** **-1-**	**B1-A1** **C1-A2** **-7-**	**A2-A1** **C1-B1** **-7-**	**A1-C1** **A2-B1** **-3-**	**A1-B1** **A2-C1** **-5-**	**B1-A2** **C1-A1** **-6-**
Girls	*A3-A4* *B3-B4* *B2-C2* *-7-*	*B4-A3* *B2-B3* *C2-A4* *-2-*	*A3-B3* *A4-B2* *C2-B4* *-3-*	*B2-A3* *A4-B4* *B3-C2* *-6-*	*B4-B2* *B3-A4* *A3-C2* *-1-*	*A4-A3* *B4-B3* *C2-B2* *-4-*	*A3-B4* *B3-B2* *A4-C2* *-2-*	*B4-A3* *B2-A4* *C2-B3* *-5-*

Table B – Tiles (Rounds) for the example league divisions

Venue /Week	1		2		3		4		5		6		7	
A	**C1-A1** **B1-A2**	*B3-A4*	*B4-A3* *C2-A4*				**A1-A2** *A4-A3*		*B4-A3* *B2-A4*		**B1-A2** **C1-A1**	*B2-A3*	**B1-A1** **C1-A2**	**A2-A1** *A3-A4*
B	*B4-B2*		**A1-B1** *B2-B3*	*A3-B4* *B3-B2*	*A3-B3* *A4-B2*	**A2-B1** *C2-B4*	*B4-B3* *C2-B2*		**A1-B1** *C2-B3*			*A4-B4*	**C1-B1**	*B3-B4*
C		*A3-C2*		**A2-C1** *A4-C2*	**A1-C1**		**B1-C1**		**A2-C1**		*B3-C2*			*B2-C2*

Penalty Points Total = 95 points

1) Multiple Games in a week: all Boys Teams week 7, all Girls Teams week 2 10 occurrences (40 points)
2) Days with odd number of Games in Referee Category: 12 days (24 points)
3) Playing Consecutive Home or Away games A1(4),A2(2),A3(3),A4(3), B1(1),B2(4),B3(3),B4(4), C1(4),C2(3) = (31 points)

Table C – Sample Problem scheduling grid

4 Related Work

A great deal of research in sports scheduling has been aimed at professional sports leagues. Kendall et al. [6] provides a broad survey of various sports leagues and their scheduling challenges. Leagues are often structured as single or double round-robin tournaments. The scheduling challenges revolve around the quality of the schedule, to achieve certain objectives. Norhona et al. [8], Ribeiro and Urrutia [11], consider fairness in South American football leagues, Kendall [5] seeks to minimize travel distances in the English Football league, and Rasmussen [9], Goossens and Spieksma in [4] consider various venue availability constraints. The instances in these problems can be reduced to tournament schedules of 20 to 24 teams, with consideration for some unique constraints and minimizing travel distance. Only Kendall [5] introduces a relationship between different divisions due to the pairing requirement (sets of venues that cannot host a game on the same day). In all these professional sports problems, a venue can be used for only one game in a day.

YSL are most closely associated with the Travelling Tournament Problem (TTP) described by Easton et al. [3]. The TTP and YSL share constraints, with the following exceptions:

1. The YSL may schedule multiples games per week, on separate days.
2. A YSL team is not required to play in every week.
3. Teams share home site venues, which support several games per day.
4. A YSL division may play an unbalanced tournament schedule.

These differences have a profound effect on the instance size, within real-world applications. The TTP instances documented on the problem definition web site [12] involve a maximum of 32 teams, albeit there is no real-world example for that size of tournament. The YSL has common occurrences of leagues with 50 divisions, comprising over 500 teams. The CYO of Westchester-Putnam, near New York City, has 68 divisions, comprising 582 teams sponsored by 58 parishes.

5 Proposed Approach

Our approach is a two phase approach based upon "tiling". Each tile represents a round of games for a division, requiring a total number of tiles equaling the number of divisions multiplied by the games per season. Each tile is one round of a tournament for a division. The tiles are placed in the master schedule in a greedy fashion. Each tile is placed in a week that results in the least number of penalty points being added to the objective function. Tiling continues until a tile cannot be placed without causing a hard-constraint violation. At this point, the venue capacity constraint is relaxed, and the remaining tiles are placed in the schedule. These final placements continue to use greedy approach to minimize total penalty points.

Once all tiles are placed, we analyze each game's contribution to the penalty points. Games violating the venue capacity hard constraint are temporarily assigned an artificially large penalty point value to prioritize them, within our local search phase. The schedule is now ready for the second phase, a local search. This phase consists of a set of swaps to reduce the current penalty points of the schedule. The following sections discuss the steps of these two phases in more detail.

5.1 Tiling Phase

For each division, we use a single round-robin tournament to produce n-1 rounds of opponent pairings, where n is the number of teams within a given division. We use one of a variety of single round-robin tournament method generators discussed in [2]. The YSL requires a number of teams, g, for each team to play. Hence, g / (n-1) is the number of round-robin tournaments needed for each division, The YSL will frequently have an unbalanced schedule. This occurs when g / (n-1) is not an integer. These divisions require using a partial round-robin tournament to complete the season. When divisions require more than a round-robin tournament, a mirrored round-robin tournament is used for the second tournament. We may use a round-robin tournament definition and its mirror several times within a division's schedule. Table D provides examples of divisions with unbalanced schedules:

Teams in the Division	Games per season	Full Round-Robin Tournament usage	Additional Tournament rounds
10	9	1 Round-Robin	None
6	12	1 Round Robin 1 Mirrored Round-Robin	2 rounds from round-robin
7	20	2 Round-Robin 1 Mirrored Round-Robin	2 rounds from Mirrored round-robin

Table D – Unbalanced Schedule in terms of tournaments

Each round of a tournament for a division is a tile. We apply a cost to each tile based upon the percentage of venue capacity used by each game. Tiles that contain several games with home teams in highly constrained venues will have a higher cost. A highly constrained venue is a venue where the usage rate ((number of teams/2) / daily slots), over the season, is higher than other venues. For example a venue sponsoring 6 teams, which can host 4 games a week, has a 75% usage rate. The rate suggests that the venue will be using an average 75% of its capacity each week. A venue usage rate over 100% indicates that a feasible schedule is not possible due to venue capacity.

Tiles with the highest costs are the first tiles to be placed in the schedule. For each week the tiles placement cost is calculated by determining the number of penalty points created by the tile's assignment for a given week. The week with the smallest placement cost, is chosen for the tile's placement. Tiling continues until the remaining tiles cannot be placed without causing a hard constraint violation of venue capacity.

The venue capacity constraint in relaxed to allow "over booking" of a venue. An artificial penalty point value of 9999 is used to enable the tile placement to continue. This temporary penalty assignment will highlight these games during the remaining tile placements and our local search phase.

5.2 Local Search Phase

The second phase involves a local search and seeks to remove hard constraint violations and reduce the number of penalty points in the existing schedule. The swaps in the YSL are similar to those discussed by Anagnostopoulos et al. [1] for solving the TTP. All swaps are done between teams in the same division. Each swap maintains a feasible schedule for each division. The swaps are:

Home / Away Swap – We swap two games, involving the same two teams, where each team is home in one game and away in the other. The swap is done between two weeks. The impact of the swap is a reduction in the number of games for one venue and an increase for the other venue. This swap is only effective in the YSL for teams playing their home games at different venues.

Round Swap – All games for two rounds of a division schedule are moved between two weeks. All venues hosting a game in either week are affected by the swap.

Partial Round Swap - The partial round swap moves a set of connected games between two weeks. We begin this operation by selecting two games in different weeks. The teams in these games create our swap set. We then add teams that are playing a team in the team-swap set to that set. The process continues until all teams, in both weeks, are in the swap set or its complement. We then move all games involving teams in the swap set between weeks. We can also choose to move all teams in the complement of the swap set between weeks.

We perform the local search in a structured fashion to reach the local minima. In the first step, we consider all games at the venue during the week that has a hard constraint violation, indicated by the artificially high penalty points. We calculate the reduction in penalty points, if any; of performing a Home/Away swap for each game. We also calculate the reduction in performing a round swap for each round. The most beneficial swap is chosen and executed.

In the second step, we analyze all games, involved in producing penalty points. Each possible home and away swap, and each possible round swap for each game is analyzed for its potential reduction in penalty points. All partial round swaps are identified for every round. The partial round swap considers both the swap set described above, and its complement set. The most beneficial swap among the three types of swaps is chosen for execution. The swap is performed and the resulting Master Schedule will have a lower penalty point total. If no improvement is found, processing is halted as the schedule has reached its local minimum. Figure 1 below presents the algorithm as pseudo-code.

1. *Tile creation*
    a. For each division generate a round-robin tournament and its mirrored image
    b. Create a tile for each round of the table until G tiles are generated, alternating the tournament table with its mirror.
    c. Assign a cost to each tile based upon the venue capacity usage of each game.
2. *Tile Placement*
    a. For each tile sorted by cost, find all weeks where the tile may be scheduled with no hard constraint violations. If no week exists, relax the venue capacity constraint and assign artificially high penalty point value for that week.
    b. Place all games in the tile within the week causing the fewest penalty points.
3. *Local Search: Home / Away swap*
    a. Set S′ to be the schedule of swapping home and away assignments for two games with the same opponents.
    b. Consider all games with the same opponents and find S′ with the minimum penalty points.

c. If S′ reduces penalty points, then swap home and away assignments for the games and continue step 3.

4. *Local Search: Round swap*
   a. Set S′ to be the schedule of swapping all games in one week with all games in another week.
   b. For each pair of weeks, calculate S′
   c. Find the S′ that minimizes the number of penalty points
   d. If S′ improves the schedule, swap rounds and repeat step 4.
5. *Local Search: Partial Round swap*
   a. For each pair of weeks, create sets of opponents who play each other in week 1 and / or week 2.
   b. Set S′ to the schedule of swapping each set of teams between the weeks.
   c. Find the S′ that minimizes the number of penalty points
   d. If S′ improves the schedule, swap rounds and repeat step 5.

Figure 1 – Tiling by Round Algorithm

6. **Results**

We will present our results at the conference. The results will include solutions to the sample instances shown in Table A, as well as the results from real-world instances of youth sports leagues, comparing the schedules actually used with our approach.

7. Future Work

In this paper, we have described a sports scheduling problem widely faced by thousands of organizations. This problem has not been reported before in the scientific literature. A set of professional-sports scheduling problems, including the TTP, have similarities to the YSL. However, key differences arise such as: sharing venues among several teams, significantly larger problem instances, and unbalanced schedules. These add real-world complexities that will challenge current approaches. We look to document problem instances and results for the community similar to the TTP [12]. We are currently working with several youth leagues to help in defining these world problem instances.

References

1. Anagnostopoulos A., Michel L., Van Hentenryck P., Vergados Y. A simulated annealing approach to the traveling tournament problem. Journal of Scheduling 2006;9:177–93.
2. Dinitz J.,E.Lamken, and Wallis, W.D. Scheduling a tournament. Handbook of Combinatorial Designs, pages 578-584. CRC Press, 1995.

3. Easton K., Nemhauser G.L., Trick M.A. The travelling tournament problem: description and benchmarks. In: Walsh T, editor. Principles and practice of constraint programming. Lecture notes in computer science, vol. 2239. Berlin: Springer; 2001. p. 580–5.
4. Goossens D, Spieksma F. Scheduling the Belgian soccer league. Interfaces 2009; 39:109–18.
5. Kendall G. Scheduling English football fixtures over holiday periods. Journal of the Operational Research Society 2008; 59:743–55.
6. Kendall G., Knust S., Ribeiro C. C. and Urrutia S. Scheduling in Sports: An Annotated Bibliography. Computers & Operations Research (2009), 37: 1-19
7. Little League Home website. Little League history. Online documentation at: http://www.littleleague.org/Learn_More/about/historyandmission/aroundtheworld.htm
8. Noronha T.F., Ribeiro C.C., Duran G., Souyris S., Weintraub A. A branch-and-cut algorithm for scheduling the highly-constrained Chilean soccer tournament. Practice and theory of automated timetabling VI. Lecture notes in computer science, vol. 3867. Berlin: Springer; 2007. pp. 174–186.
9. Rasmussen R.V. Scheduling a triple round robin tournament for the best Danish soccer league. European Journal of Operational Research 2008; 185:795–810.
10. Regin J.C. Minimization of the number of breaks in sports scheduling problems using constraint programming. DIMACS series in discrete mathematics and theoretical computer science, vol. 57, 2001. p. 115–30.
11. Ribeiro C.C., Urrutia S. Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. Practice and theory of automated timetabling VI. Lecture notes in computer science, vol. 3867. Berlin: Springer; 2007. p. 147–57.
12. Trick, M.A., Challenge Traveling tournament instances. Online document at http://mat.gsia.cmu.edu/TOURN/.

Appendix A. Glossary

Capacity: The number of games that can be played at a given venue on any day. The capacity may vary by venue.

Division: A set of teams where every team in the division plays against every other team a set number of times. Subsets of these teams may play one additional game amongst them to reach the required number of games.

Game: A sports contest between two teams.

League: A set of divisions, comprised of teams, which share a common set of venues.

Organization: An entity that maintains a venue and sponsors teams across many divisions.

Round-robin tournament: A round-robin tournament involves *n* teams playing *n*-1 games. Each game is played against a different opponent in the tournament. Also, every team plays a game in each round.

Slot: A time period during a given day to be used to play a game at a venue. The YSL schedules a fixed number of slots per day at each venue, depending upon the venue's daily capacity. The actual clock time of the slot is not considered in the problem.

Unbalanced Schedule: A schedule where a team will play one opponent more often than another opponent. The YSL requires unbalanced schedules in divisions where the required number of games in a season is not a multiple of the number of n 1, for a division.

Venue: A physical location for a game to played.

A Novel Event Insertion Heuristic for Creating Feasible Course Timetables

Moritz Mühlenthaler · **Rolf Wanka**

Abstract We propose a novel event insertion heuristic for finding feasible solutions for instances of the University Course Timetabling Problem (UCTP). We introduce and apply a new neighbourhood structure on partial timetables that permits to approach a feasible timetable. The key insight is that an event can be inserted in a time slot if all the events conflicting with it are moved to other time slots. In order to prevent our event insertion heuristic from running into local optima, a simple perturbation strategy is employed additionally. Our experimental results show that our event insertion heuristic yields superior results compared to other state-of-the-art feasible solution generation algorithms for a large number of corresponding benchmark instances.

Keywords Course Timetabling Problems · Feasible Solution Generation · Kempe Move

1 Introduction

The task of creating course timetables such that certain constraints are satisfied occurs periodically in all sorts of educational institutions such as high schools and universities. Typically the constraints to be considered are divided into hard and soft constraints. Hard constraints are "must haves," i. e., timetables which violate any of the hard constraints are considered infeasible. On the other hand, soft constraints are "nice to have," i. e., timetables with fewer soft constraint violations are more convenient for staff and students, and are thus preferred. The University Course Timetabling Problem (UCTP) is an NP-hard combinatorial optimisation problem in the setting of a university with the objective of finding a feasible timetable with minimal soft constraint violations.

In this paper, we propose a novel heuristic approach for finding feasible timetables for UCTP instances that is based on a new sophisticated neighbourhood structure for timetables. The research yielding the proposed heuristic was motivated by the authors' university administration, who posed the following question: "Do we have to rent additional rooms for

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

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

a temporarily increased course load, or is there *any* way we can get away with the rooms we have?" No specific soft constraints are imposed in this particular case, the generation of a feasible timetable is sufficient. But feasible timetable generation is also an important part of solving UCTPs when soft constraints are involved. Since hard and soft constraints are typically considered in distinct phases of the optimisation process, a feasible solution is required before even considering the soft constraint violations.

The proposed event insertion heuristic called *Kempe insertion heuristic* is built around a novel neighbourhood structure for timetables, which is closely related to the distance to feasibility. The key feature of this neighbourhood structure is that each move in the neighbourhood brings a timetable closer to feasibility. The experimental results presented in Section 4 show that our Kempe insertion heuristic outperforms current state-of-the-art feasible solution generation algorithms with respect to solution quality and computation time for the small and medium benchmark instances. This means, for such instances, solvers can spend more time on minimising soft constraint violations and thus potentially find better timetables with the same amount of computation time. For the large instances, our results are on par with the currently best performing algorithm by Tuga *et al.* [13], the HSA (hybrid simulated annealing) algorithm. For a considerable number of instances, we only need a fraction of CPU time compared to HSA. The advantage of our approach is that only *one single* sophisticated neighbourhood structure is used whereas HSA uses three different, but simple neighbourhood structures.

The remainder of this paper is organised as follows. In Section 2, the basic definitions concerning the distance to feasibility of timetables and Kempe moves are reviewed. In Section 3, the event insertion heuristic is discussed and we detail how to construct the neighbourhood structure. In Section 4, the performance of the Kempe insertion heuristic is evaluated based on the timetables generated for the 60 benchmark instances proposed by Lewis et al. in [5]. It is compared to the results of the HSA algorithm [13] and the Grouping Genetic Algorithm and the Heuristic Search Algorithm, both from [4].

2 Preliminaries

In this section we give the necessary definitions of structures and operations used by the Kempe insertion heuristic. First, we give a more formal definition of UCTPs, which were informally described above and define the notion of a *partial timetable*. We also give a short review of the *Kempe move*, which is a popular technique in educational timetabling for moving events in a timetable. The Kempe move is the main ingredient of the neighbourhood structure used by our new Kempe insertion heuristic.

2.1 Problem Definition

A UCTP instance I consists of the following data: A set $E = \{c_0, c_1, \ldots\}$ of events (or courses), a set $T = \{t_1, t_2, \ldots\}$ of time slots, and a set $R = \{r_0, r_1, \ldots\}$ of rooms. Additionally, we are given two relations $C \subseteq E \times E$ and $S \subseteq E \times R$. We say that events c and c' are in conflict if $(c, c') \in C$. We say that a room r is suitable for an event c if $(c, r) \in S$.

A *feasible timetable* for a UCTP instance I is an assignment $\tau : E \to R \times T$ of events to $(\textit{room}, \textit{time slot})$-pairs called *resources* such that each of the following *hard constraints* is satisfied:

1. Each event is assigned to a suitable resource.

2. No conflicting events occur in the same time slot.
3. No room is double-booked.

This definition is the basis of our proposed algorithm. It is consistent with the UCTP formulations for the benchmarking instances for feasible timetable generation from [5]. As indicated above, no soft constraints are to be considered for our purpose of finding a feasible timetable.

A *partial timetable* is an assignment of events to resources such that hard constraints 2 and 3 are satisfied. Additionally, a relaxed version of hard constraint 1 is imposed: we require that all assigned resources are suitable for the respective events, but not all events have to be assigned to a resource. Building on the notion of a partial timetable, feasible timetable generation can be turned into an optimisation problem for which the single objective is to minimise the number of events not assigned to a resource. Clearly, if no events remain unassigned, we have found a feasible timetable. The *distance to feasibility* of a partial timetable is the number of events which have not been assigned to a resource.

2.2 The Classical Kempe Move

The *Kempe move* [9] is a technique for moving events between time slots of a timetable. It has been applied with great success in a wide range of feasible timetable generation and soft constraint optimisation algorithms for educational timetabling [1, 7, 8, 12, 13]. The Kempe move is the basic ingredient of the neighbourhood construction in our new Kempe insertion heuristic proposed in the next section. The idea behind using Kempe moves for feasible solution generation is that as long as a few simple requirements are met, applying a Kempe move to a partial timetable results again in a partial timetable. As we will see below, the resulting timetable has the same distance to feasibility as the original timetable. Hence, no additional hard constraint violations are introduced by performing such Kempe moves. For other definitions of the distance to feasibility (see for example [4, 6, 13]), similar properties can be established.

A Kempe move is based on the identification of connected components in a bipartite graph. Let τ be a partial timetable for a UCTP instance I and let

$$E_t = \{c \mid c \in E, \exists r \in R : \tau(c) = (r,t)\}$$

be the events scheduled in time slot t. Now we consider two timeslots s and t along with the bipartite graph $G_{s,t}$ whose nodes are $E_s \cup E_t$ and whose edges are induced by the conflict relation of I. Without loss of generality, let e be any node in E_s called the *trigger event* and $D = D_s \cup D_t$ be the connected component of e with $D_s \subseteq E_s$ and $D_t \subseteq E_t$. None of the events in D_s can be moved to the time slot t as long as the events D_t are present in t without introducing conflicts that would violate the partiality requirements of τ. However, when the events in D_s are swapped with those in D_t, no such conflicts are introduced. A Kempe move between time slots s and t triggered by e swaps the events in D_s with those in D_t and hence, no conflicting events occur in E_s or in E_t after the Kempe move.

As an example, let us consider two timeslots s and t, which are populated by the events $\{c_i\}_{0 \le i < 12}$ as shown in Figure 1. Any two events connected by an edge are in conflict. The connected component of the trigger event c_0 is $\{c_0, c_3\} \cup \{c_6, c_7\}$, so the Kempe move triggered by c_0 exchanges c_0 and c_3 with c_6 and c_7. As indicated in Figure 1, the room assignment may need to be rearranged when exchanging events between timeslots. If, for example, event c_7 can only be scheduled in room 1, c_1 needs to be moved to a different

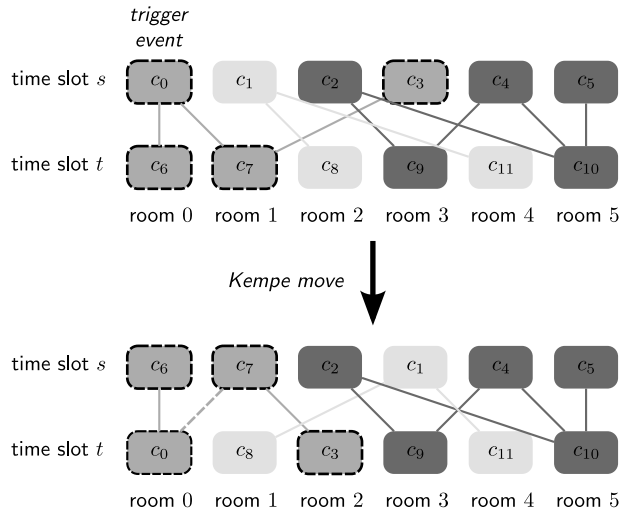


Fig. 1 A Kempe move between time slots s and t which is triggered by the event c_0. Conflicting events are connected by edges.

room. The task of assigning events to suitable rooms for a particular timeslot t can be stated in terms of a maximum cardinality bipartite matching problem, which can be solved, for instance, by the augmenting paths algorithm given in [11] in time $O(\min\{|E_t|,|R|\} \cdot |A|)$, where $|A|$ is the number of suitable room/time slot combinations. Please note that the Kempe move can also be used for soft constraint optimisation with only a slight modification: if the cost of assigning a particular event to a particular room is known, a minimum weight bipartite matching algorithm can be used to determine a room assignment with minimal cost for a particular time slot. See [7] for a UCTP solver using this technique.

For the purpose of finding feasible timetables, we have to consider the following if we want to avoid introducing additional hard constraint violations: when reassigning rooms for the events in a time slot t, each event has to be assigned to a suitable room, i. e., the cardinality of the matching has to be equal to $|E_t|$. If this is the case, we say that a Kempe move is *admissible*. If we perform an admissible Kempe move on a partial timetable, the result is again a partial timetable and its distance to feasibility, as defined in Section 2.1, is not altered by the Kempe move.

3 Feasible Timetable Generation

The new algorithm for generating feasible course timetables consists of two consecutive phases: In the first phase, a simple sequential heuristic, similar to the one used in [13], is employed for quickly creating a partial timetable with as many events assigned to suitable resources as possible. Typically for harder instances, not all events can be assigned to suitable resources by the sequential heuristic. Hence, in the second phase, we try to assign feasible resources to the remaining events using the proposed new *Kempe insertion heuristic* (see Section 3.2). This heuristic uses a computationally heavier neighbourhood exploration scheme based on the Kempe move in order to free suitable resources for the remaining events and bring the timetable gradually closer to feasibility.

3.1 The Sequential Heuristic

Sequential event insertion heuristics generate for a given UCTP instance a partial timetable. The essence of sequential heuristics is that, starting with an empty timetable, events are inserted one by one such that those events are scheduled first, which are likely to be difficult to insert in an already populated timetable. For highly constrained instances, such as the ones proposed by Lewis and Paechter in [5], sequential heuristics are very unlikely to produce feasible timetables. It turns out however that combining the new Kempe insertion heuristic with a sequential heuristic generally yields better timetables within limited computation time than the Kempe insertion heuristic alone because sequential event insertion is, in comparison, very fast.

Algorithm 1 shows the sequential heuristic used in conjunction with the Kempe insertion heuristic for obtaining the experimental results in Table 1. SEQUENTIAL_HEURISTIC takes as input a UCTP instance I, and returns a partial timetable. In the initialisation step, the events to be scheduled are sorted lexicographically according to i) the number of suitable time slots and ii) the number of suitable rooms. Hence, events with the most time slot constraints are scheduled first, and among them the ones with the least number of suitable rooms. Then the sorted list μ of events is traversed in forward order, and for each event a random suitable resource, i. e., a suitable $(\mathit{room}, \mathit{time\ slot})$-pair, is picked. If none is available for a particular event, it remains unassigned. As soon as all events in μ have been processed, the resulting partial timetable is returned.

Algorithm 1: SEQUENTIAL_HEURISTIC

```
input  : I: UCTP instance
output: τ: partial/feasible timetable
τ ← empty timetable
ρ ← list of all resources of I arranged in random order
μ ← list of all events to be scheduled
sort items in μ by i) the number of suitable time slots and ii) the number of suitable rooms
foreach event e in μ do
    if ρ contains a suitable resource for e then
        r ← first suitable resource for e in ρ
        remove r from ρ
        assign e to r
    end
end
return τ
```

The crucial part of the sequential heuristic is the determination of the order in which events are inserted in the timetable. We have tried several approaches, including those proposed by Burke *et al.* in [2], as well as the combination of "least saturation degree first"(LSD) and "largest degree first" (LD) used by Tuga *et al.* in [13]. We found that the sorting criteria as given in Algorithm 1 yield the best results in conjunction with our Kempe insertion heuristic. However, switching to the LSD/LD sequential heuristic used by Tuga *et al.* changes the results only marginally.

3.2 The Kempe Insertion Heuristic

The proposed Kempe insertion heuristic is built around a novel neighbourhood structure for partial timetable transformations, we dub *Kempe insertion neighbourhood*. A *neighbourhood* of a partial timetable τ is a collection of sequences of (admissible) Kempe moves on τ. The Kempe insertion neighbourhood has been specifically designed such that each move in the neighbourhood of a partial timetable decreases its distance to feasibility by one. Current state-of-the-art solvers typically use a combination of different neighbourhoods, each one with its individual strengths and weaknesses [3, 7, 10, 13]. For example the hybrid simulated annealing approach for feasible timetable generation by Tuga *et al.* [13] uses a combination of the *simple*, *swap* and *Kempe chain* neighbourhoods. Our objective however is to show that the Kempe insertion neighbourhood is a very good general purpose neighbourhood for feasible timetable generation, so our feasible solution generation approach relies exclusively on the Kempe insertion neighbourhood.

Let $E^- \subseteq E$ be the set of events which are yet to be scheduled in a partial timetable τ. The key observation behind the Kempe insertion neighbourhood is that an event c can be inserted in the time slot s of a timetable τ if the following two conditions are met: First, all events in s conflicting with c can be moved to a different time slot using admissible Kempe such that no additional events conflicting with c are moved to s. And second, suitable rooms can be assigned to all remaining events in s and c. So for an event c and a time slot s, we can define the set

$$\mathscr{N}_c^\tau(s) = \{K \mid K \text{ is a sequence of admissible Kempe moves involving time slot } s \\ \text{s. t. } c \text{ can be inserted in } s \text{ after performing the moves in } K\}.$$

For an event $c \in E^-$, the objective is to find a time slot s such that $\mathscr{N}_c^\tau(s)$ is non-empty. If such a time slot can be found, then c can be inserted in the timetable. Hence, the Kempe insertion neighbourhood $\mathscr{N}_c^\tau$ of a timetable τ with respect to an event $c \in E^-$ is

$$\mathscr{N}_c^\tau = \bigcup_s \mathscr{N}_c^\tau(s) \ .$$

If $\mathscr{N}_c^\tau$ is non-empty, the event c can be inserted in the timetable and as a consequence, the distance to feasibility of τ decreases by one. Hence the goal of the Kempe insertion heuristic is to make a partial timetable feasible by finding a non-empty $\mathscr{N}_c^\tau$ for each $c \in E^-$.

Figure 2 shows, by example, how a single Kempe move k with $(k) \in \mathscr{N}_c^\tau(s)$ is performed to fit an event c in a time slot s. In the example, the only event in conflict with c in s is c_2, which is moved to a time slot t by k. None of the remaining events in s is in conflict with c, so room assignment can be performed for the events $\{c\} \cup \{c_1, c_3, c_4, c_5\}$ using a maximum cardinality bipartite matching algorithm as described in the previous section.

The full Kempe insertion heuristic is outlined in Algorithm 2. INSERTION_HEURISTIC takes as input a partial timetable τ of some UCTP instance I, as well as the exploration parameter d and a time limit. It returns a feasible timetable as soon as one has been found or the best partial timetable found before the time limit is hit. In each iteration, an event e is picked at random from the list μ of unscheduled events. If some element in $\mathscr{N}_e^\tau$ can be found using NEIGHBOURHOOD_SEARCH (see Algorithm 3), e is added to the timetable, otherwise e remains unscheduled. If less than two events were successfully added to the timetable within $\max\{k, |\mu|\}$ iterations, the search is considered stuck and we insert a randomly chosen event $e \in \mu$ in the timetable "by force." This means a target time slot s is picked for e and all events $C_s(e)$ in conflict with e in s are removed from the timetable so that e can be inserted in s. This perturbation move increases the distance to feasibility by $|C_s(e)| - 1$, and therefore, we

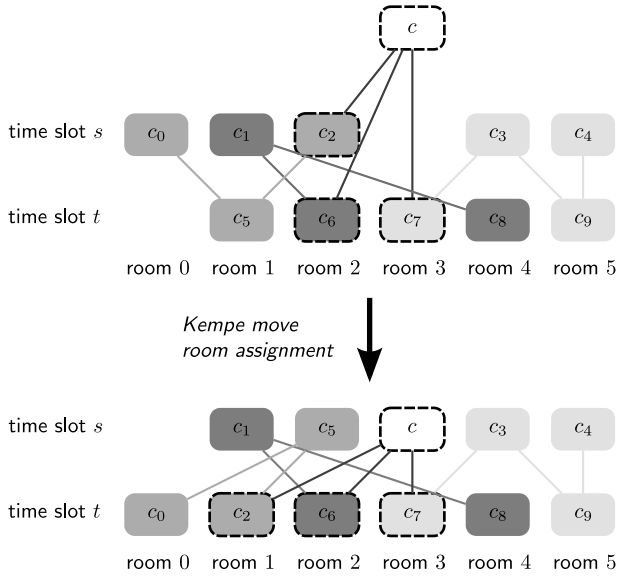


Fig. 2 Inserting an event c in the time slot s: A single Kempe move k with $(k) \in \mathcal{N}_c^\tau$ removes from s all events in conflict with c. When s has been cleared of all such events, c can be inserted in s.

do not want to use this perturbation operation too often. On the other hand, when we have tried to insert a number of events with only little success we have probably wasted CPU time because we are stuck in a local optimum. We can influence how often the perturbation operation is performed by setting the exploration parameter d, which is an upper bound for the number of iterations of NEIGHBOURHOOD_SEARCH before considering the perturbation operation depending on how many events were successfully inserted in the timetable.

The key element of INSERTION_HEURISTIC is the neighbourhood exploration shown in Algorithm 3. In NEIGHBOURHOOD_SEARCH, we try to find for an event c an element of $\mathcal{N}_c^\tau$ in a greedy fashion, so we can add c to the timetable. For each suitable time slot for c, NEIGHBOURHOOD_SEARCH tries to remove events conflicting with c by using Kempe moves. If it succeeds to clean a time slot from all such events, it checks if additional rooms need to be freed using another Kempe move. Now, if rooms can be assigned successfully to all events in $E_s \cup \{c\}$, NEIGHBOURHOOD_SEARCH has found an element of $\mathcal{N}_c^\tau(s) \subseteq \mathcal{N}_c^\tau$ and returns s. It returns "invalid time slot" to indicate that no element of $\mathcal{N}_c^\tau$ has been found. It is possible that NEIGHBOURHOOD_SEARCH cannot find a sequence of moves such that an event e can be inserted in the timetable, although $\mathcal{N}_c^\tau$ is, in principle, non-empty. However, trying to find any sequence of moves such that e can be scheduled is not computationally feasible and, as the experimental results in the next section show, NEIGHBOURHOOD_SEARCH is quite successful in finding such sequences.

In the worst case, NEIGHBOURHOOD_SEARCH tries to fit an event e in every time slot without success. Then for each time slot s and for each event c in the time slot in conflict with e, the algorithm has tried to find a time slot $t \neq s$ such that c can be moved to t without introducing new conflicts. This means, in the worst case, performing NEIGHBOURHOOD_SEARCH results in $O(|T|^2 \cdot |C|)$ attempts to remove events from a time slot using Kempe moves without finding an element of $\mathcal{N}_e^\tau$, where $|T|$ is the number of time slots and $|C|$ the number of conflicts . To increase the likeliness of INSERTION_HEURISTIC to find an

Algorithm 2: INSERTION_HEURISTIC

input : I: UCTP instance
input : k: max. number of iterations until perturbation
in/out : τ: partial timetable

$\tau_{best} \leftarrow \tau$
$\mu \leftarrow$ list of events to be scheduled
while *time limit not hit and* τ *infeasible* **do**
  success $\leftarrow 0$
  for $\max\{k, |\mu|\}$ *iterations* **do**
    e $\leftarrow$ random element from μ
    s $\leftarrow$ NEIGHBOURHOOD_SEARCH(τ, e)
    if s *is a valid time slot* **then**
      insert e in s
      success $\leftarrow$ success $+ 1$
    end
  end
  if $dist(\tau) < dist(\tau_{best})$ **then** $\tau_{best} \leftarrow \tau$
  if success ≤ 1 **then**
    pick e at random from μ
    force insertion of e in τ
    update μ
  end
end
return τ_{best}

Algorithm 3: NEIGHBOURHOOD_SEARCH

input : e: event to be scheduled
in/out : τ: partial timetable

foreach *suitable time slot* s *for* e **do**
  cleanslot $\leftarrow$ s
  `/* try to reschedule all events in s conflicting with e */`
  foreach *event* c *in* s *conflicting with* e **do**
    find a time slot t $\neq$ s s.t. there is an admissible Kempe Move which moves e to t without introducing events conflicting with e in s
    if *suitable* t *was found* **then** KempeMove(s, t, c)
    else cleanslot $\leftarrow$ invalid time slot; break
  end
  `/* if rescheduling failed, try to gather conflicting events in s */`
  if cleanslot $=$ *invalid time slot* **then**
    T $\leftarrow$ suitable time slots not yet processed
    find a time slot t in T and a trigger event c s.t. c triggers an admissible Kempe Move, which increases the number of events conflicting with e in s
    if *suitable* t *and* c *were found* **then**
      KempeMove(s, t, c); break
  end
end
if cleanslot $\neq$ *invalid time slot* **then**
  if *all rooms are booked in* cleanslot **then**
    find a time slot t and a trigger event c s.t. c triggers an admissible Kempe Move, which decreases the number of events in s without introducing events conflicting with e
    if *suitable* t *and* c *found* **then** KempeMove(s, t, c)
  end
  if *suitable rooms can be assigned to events* ($E_{\text{cleanslot}} \cup \{e\}$) **then**
    return cleanslot
end
return invalid time slot

element of $\mathcal{N}_e^\tau$ as early as possible, we do the following: If NEIGHBOURHOOD_SEARCH fails to free a time slot s from all events conflicting with e, we gather events conflicting with e from the time slots which are yet to be processed. More precisely, in the remaining time slots we look for a time slot t such that we can perform an admissible Kempe move which *increases* the number of events conflicting with e in s. Experiments indicated that gathering conflicts in this fashion improves the overall running time of INSERTION_HEURISTIC considerably when feasible solutions are found and also seems to have a beneficial impact on the overall quality of the solutions obtained.

4 Experimental Results

Our experimental results were obtained for the 60 problem instances in [5]. These instances were specifically designed to be hard to solve by sequential heuristics as described in Section 3.1. For each instance however, it is guaranteed that there exists at least one feasible solution. The 60 instances are divided in three categories: Small instances with 200 to 225 events and 5 to 6 rooms, medium instances with 390 to 425 events and 10 to 11 rooms, and large instances with 1000 to 1075 events and 25 to 28 rooms. For all instances, the number of time slots is 45, and all events can be scheduled in any of the 45 time slots if there are no conflicting events in a time slot already.

Our solutions were obtained by running SEQUENTIAL_HEURISTIC 150 times and then using the best partial solution found so far as input for INSERTION_HEURISTIC. Feasible solutions were found by the sequential heuristic for the instances *small* 2, 6, 11, 12 and 20. In comparison, in [13], Tuga *et al.* performed their sequential heuristic 500 times for each instance as a preprocessing step and found feasible solutions for 14 of the 60 instances just using the sequential heuristic. It turned out however, that increasing the number of iterations or modifying the sequential heuristic did not improve the overall solution quality in our experiments. The exploration parameter for INSERTION_HEURISTIC was set to 16 and timeout values were set to 100 s for the small instances, to 200 s for the medium instances and to 500 s for the large instances.

The results shown in Table 1 were obtained by performing 20 consecutive runs for each instance on a *single* core of personal computer equipped with a Intel QuadCore CPU clocked at 3 GHz. Running the sequential heuristic 150 times took between 0.2 s and 1.7 s, depending on the size of the instance. For each instance, the lowest and average distance to feasibility and the average CPU time were recorded. Table 1 shows our results along with the results obtained by Tuga *et al.* in [13] (HSA) on a Pentium IV 3.2 GHz, and Lewis and Paechter in [4] (Lewis I and II) for comparison. Note that different time limits were imposed in the experiments run in [13] and [4]. Tuga *et al.* set the timeouts to 200, 400 and 1000 seconds for the small, medium and large instances, respectively [13]. Lewis and Paechter imposed timelimits of 30, 200 and 800 seconds for small, medium, and large instances to obtain their results [4]. To the knowledge of the authors, no average running times were given in [4].

As shown in Table 1, to combine SEQUENTIAL_HEURISTIC and INSERTION_HEURISTIC consistently outperforms the other algorithms for the small and medium benchmark instances with respect to the distance to feasibility of the obtained solutions and CPU time used. Our Kempe insertion heuristic found feasible timetables for all 40 small and medium instances. Concerning the large instances, our algorithm performs better than both algorithms proposed in [4]. Also, our algorithm is at least as good as the HSA approach by Tuga *et al.* for 13 out of the 20 large instances despite the shorter timeout and uses much less CPU time on average for many instances such as *big* 2, 3, 12, 13, 14 and 16.

	ins. heuristic		HSA		Lewis I	Lewis II
Instance	best(avg)	avg time	best(avg)	time	best(avg)	best(avg)
small 1	0(0)	0.0	0(0)	0	0(0)	0(0)
small 2	0(0)	0.0	0(0)	0	0(0)	0(0)
small 3	0(0)	0.1	0(0)	9	0(0)	0(0)
small 4	0(0)	0.0	0(0)	0	0(0)	0(0)
small 5	0(0)	0.2	0(0)	5	0(1.05)	0(0)
small 6	0(0)	0.0	0(0)	0	0(0)	0(0)
small 7	0(0)	0.1	0(0)	0	0(0)	0(0)
small 8	0(0)	15	0(1.9)	79	4(6.45)	0(1)
small 9	0(0)	1.7	0(3.85)	84	0(2.5)	0(0.15)
small 10	0(0)	0.4	0(0)	15	0(0.1)	0(0)
small 11	0(0)	0.0	0(0)	0	0(0)	0(0)
small 12	0(0)	0.0	0(0)	0	0(0)	0(0)
small 13	0(0)	1.0	0(1)	15	0(1.25)	0(0.35)
small 14	0(0)	33	3(5.95)	136	3(10.5)	0(2.75)
small 15	0(0)	0.0	0(0)	0	0(0)	0(0)
small 16	0(0)	0.0	0(0)	13	0(0)	0(0)
small 17	0(0)	0.1	0(0)	13	0(0.25)	0(0)
small 18	0(0)	0.0	0(0.45)	36	0(0.7)	0(0.2)
small 19	0(0)	0.5	0(1.2)	25	0(0.15)	0(0)
small 20	0(0)	0.0	0(0)	0	0(0)	0(0)
med 1	0(0)	0.21	0(0)	0	0(0)	0(0)
med 2	0(0)	0.15	0(0)	0	0(0)	0(0)
med 3	0(0)	0.73	0(0)	8	0(0)	0(0)
med 4	0(0)	0.30	0(0)	3	0(0)	0(0)
med 5	0(0)	5.2	0(0)	85	0(3.95)	0(0)
med 6	0(0)	4.0	0(0)	20	0(6.2)	0(0)
med 7	0(0)	80	1(4.15)	440	34(51.65)	14(18.5)
med 8	0(0)	4.2	0(0)	12	9(15.95)	0(0)
med 9	0(0.1)	142	0(4.9)	269	17(24.55)	2(9.7)
med 10	0(0)	0.0	0(0)	0	0(0)	0(0)
med 11	0(0)	1.3	0(0)	25	3(13.35)	0(0)
med 12	0(0)	0.2	0(0)	54	0(0.25)	0(0)
med 13	0(0)	1.6	0(0.5)	172	30(43.15)	0(0.5)
med 14	0(0)	1.0	0(0)	59	0(0.25)	0(0)
med 15	0(0)	1.6	0(0.05)	72	0(4.85)	0(0)
med 16	0(0)	7.3	1(5.15)	733	30(43.15)	1(6.4)
med 17	0(0)	1.4	0(0)	39	0(3.55)	0(0)
med 18	0(0)	5.6	0(6.05)	429	0(8.2)	0(3.1)
med 19	0(0)	11	0(5.45)	511	0(9.25)	0(3.15)
med 20	0(0)	15	2(10.6)	457	0(2.1)	3(11.45)
big 1	0(0)	0	0(0)	0	0(0)	0(0)
big 2	0(0.05)	56	0(0)	283	0(0.7)	0(0)
big 3	0(0)	26	0(0)	447	0(0)	0(0)
big 4	0(1.4)	465	0(0)	406	30(32.2)	8(20.5)
big 5	5(8.4)	500	0(1.1)	743	24(29.15)	30(38.15)
big 6	29(40.3)	500	5(8.45)	893	71(88.9)	77(92.3)
big 7	100(109.2)	500	47(58.3)	966	145(157.3)	150(168.5)
big 8	0(0.25)	329	0(0)	210	30(37.8)	5(20.75)
big 9	0(0.7)	434	0(0.05)	419	18(25)	3(17.5)
big 10	9(12.5)	500	0(1.25)	660	32(38)	24(39.95)
big 11	8(10.2)	500	0(0.35)	444	37(42.35)	22(26.05)
big 12	0(0)	37	0(0)	240	0(0.85)	0(0)
big 13	0(0)	70	0(0)	274	10(19.9)	0(2.55)
big 14	0(0)	54	0(0)	271	0(7.25)	0(0)
big 15	0(11.4)	496	0(0)	255	98(113.95)	0(10)
big 16	0(0)	80	0(2)	755	100(116.3)	19(42)
big 17	13(57.5)	500	76(89)	998	243(266.55)	163(174.9)
big 18	0(0)	259	53(62)	764	173(194.75)	164(179.25)
big 19	161(171.5)	500	109(127)	998	253(266.65)	232(247.35)
big 20	6(13.0)	500	40(46.7)	827	165(183.15)	149(164.15)

Table 1 Experimental results for the 60 benchmark instances from [5]. Average time is given in seconds.

5 Conclusions

In this paper, the new Kempe insertion heuristic has been proposed for generating feasible timetables for university course timetabling problems. Our approach is based on exploring a sophisticated neighbourhood structure for partial timetables, the Kempe insertion neighbourhood. Each move in the neighbourhood structure brings the partial timetable computed so far closer to feasibility. In addition, a perturbation strategy has been proposed for preventing the Kempe insertion heuristic from getting stuck in local optima.

The Kempe insertion heuristic has been tested on the 60 benchmark instances from [5]. Our results show that our algorithm consistently outperforms other state-of-the-art algorithms for feasible solution generation [4, 13] for the small and medium benchmark instances with respect to the distance to feasibility of the timetables and CPU time used. For 13 of the 20 large instances, the Kempe insertion heuristic performs at least as good as the Hybrid Simulated Annealing algorithm from [13] despite the shorter timeout and uses much less CPU time on average for many instances. The Kempe insertion heuristic generally outperforms the Grouping Genetic Algorithm and the Heuristic Search Algorithm from [4].

References

1. Edmund K. Burke, Adam J. Eckersley, Barry McCollum, Sanja Petrovic, and Rong Qu. Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research*, 206(1):46–53, 2010.
2. Edmund K. Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.
3. Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms*, 5(1):65–89, 2006.
4. Rhydian Lewis and Ben Paechter. Finding feasible timetables using group-based operators. *IEEE Transactions on Evolutionary Computation*, 11:397–413, 2007.
5. Rhydian Lewis and Ben Paechter. http://www.emergentcomputing.org/timetabling/harderinstances.htm, accessed 2010.
6. Rhydian Lewis, Ben Paechter, and Barry McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Accounting and Finance Working Papers A2007/3, Cardiff University, Cardiff Business School, Accounting and Finance Section, July 2007.
7. Zhipeng Lü and Jin-Kao Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.
8. Liam T. G. Merlot, Natashia Boland, Barry D. Hughes, and Peter J. Stuckey. A hybrid algorithm for the examination timetabling problem. In *Proc. 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 207–231. Springer, 2003.
9. Craig Morgenstern and Harry Shapiro. Coloration neighborhood structures for general graph coloring. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 226–235, 1990.
10. Tomáš Müller. ITC2007 solver description: A hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.
11. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization; Algorithms and Complexity*. Dover Publications, 1998.
12. Jonathan M. Thompson and Kathryn A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25(7-8):637 – 648, 1998.
13. Mauritsius Tuga, Regina Berretta, and Alexandre Mendes. A hybrid simulated annealing with Kempe chain neighborhood for the university timetabling problem. In *Proc. 6th ACIS Int. Conf. on Computer and Information Science (ACIS-ICIS)*, pages 400–405, 2007.

Choquet Integral for Combining Heuristic Values for Exam Timetabling Problem

Tiago Cardal Pais · Edmund Burke

Abstract In this paper we present a constructive heuristic approach based on Choquet integral. We use this method to combine the information given by different basic heuristics. We use a fuzzy measure to model the importance of each heuristic as well as the interaction between them. We test our approach on 2 different testbeds and compare its performance against the individual heuristics. Moreover, we also compare the results against the best results reported in the literature.

Keywords Exam Timetabling, Fuzzy Measure, Choquet Integral, Construction Heuristics

1 Introduction

Problems related to timetabling are present in daily life. Solving timetabling problems is a crucial task and affects many institutions and services like hospitals, transportation enterprizes, educational establishments, among many others. These problems have been the object of increasing interest by the research community. Many interesting proposals have been presented, particulary in the field of Operations Research and Artificial Intelligence, to solve timetabling problems in sports (Easton et al. 2004; Trick 2001), transportations (bus,railways,planes) (Isaai and Singh 2001; Caprara et al. 2001; Qi et al. 2004), schools (Abramson et al. 1999; Colorni et al. 1998; Ribeiro Filho and Lorena 2001; Hansen and Vidal 1995; Schaerf 1999) and universities (Awad and Chinneck 1998; Burke et al. 2006; Burke and Newall 2003; Caramia et al. 2001; Casey and Thompson 2003; Carter et al. 1994, 1996; Corr et al. 2006; Dowsland and Thompson 2004; Erben 2001; Di Gaspero 2002; Di Gaspero and Schaerf 2001; Kendall and Mohd Hussin 2004; Merlot et al. 2003; Paquete and Fonseca 2001; Petrovic and Bykov 2002; Schimmelpfeng and Helber 2007; Thompson and Dowsland 1996, 1998; White and Xie 2001; Yang and Petrovic 2004).

A general definition of timetabling was given by Burke, Kingston, and de Werra (2004):

Tiago Cardal Pais · Edmund Burke
ASAP group, University Of Nottingham, Nottingham, NG8 1BB, UK
E-mail: {tpp,ekb}@cs.nott.ac.uk

"A timetabling problem is a problem with four parameters, T a finite set of times, R a finite set of resources, M, a finite set of meetings: and C, a finite set of constraints. The problem is to assign times and recourses to the meetings so as to satisfy the constraints as far as possible."
Hence, if we consider exams as meetings then we are facing exactly the problem that we want to tackle in this paper, that is, the exam timetabling problem.
In Burke and Newall (2004) an iterated construction algorithm is described. They make use of a construction ordering heuristic as the basic method for scheduling the exams. However, the authors introduce an iterated adaptive method which consists of changing the "degree" of each exam in each iteration. They proposed an incremental and exponential adaptation scheme. In the first case the "degree" is modified by one unit at each iteration. On the other hand, the exponential scheme increments the "degree" by 2^n, where n is the number of iterations by which a particular "degree" was modified. Moreover, they compare the performance of the algorithm when using different basic ordering heuristics. They use the largest degree first (LD), a flat ordering (which initializes every "degree" on 0), smallest degree first (SmD), saturation degree (SD) and random ordering. They tested the algorithm using the Toronto's data set (Carter et al. 1996). It can be observed that the adaptation mechanism helps to improve the initial timetable given by the original order of the exams.
In Asmuni et al. (2009) a fuzzy multiple heuristic ordering approach is presented. In this work a simple heuristic ordering was implemented, based on the Carter and Laporte (1996) algorithm. The following three different criteria were used to order exams: (1) largest degree (LD); (2) largest enrolment (LE); (3) and least saturation (SD) degree criterion. They use a fuzzy inference system to combine the different criteria previously mentioned. All possible combinations were tested ($LD + LE$, $SD + LE$ and $LD + SD$). The Mandani type fuzzy system that they used has a 9 rule structure, meaning that two linguistic variables were used to evaluate the exam "quality" and for each variable three linguistic terms were defined: "small", "medium" or "high". The output linguistic variable "examweight" is also defined by the same three linguistic terms. A pre-normalisation of data was also performed before computing the fuzzy inference system. In this process a linear transformation was used. Furthermore, a tuning process was also implemented. It consists, basically, of changing simultaneously, by small steps, the upper bound, centre and lower bound of the three membership functions. All results obtained for each instance used the best "tuned system". All approaches were tested using the Toronto's data set (Carter et al. 1996). They conclude that the approach using a tuned fuzzy system with the SD and LE as input variables gave, overall, the best results.
Qu et al. (2009a) presented an adaptive hybridisation of basic graph heuristics within a graph hyper-heuristic framework. They first started studying some statistical properties of a random constructive graph hyper-heuristic. They observed that sequences of SD heuristic hybridised with Largest Weighted Degree (LWD) gave better results than if hybridised with LD or LE heuristic. Following that, they proposed an adaptive approach which uses sequences of SD heuristic hybridized with LWD. This new method consists of two steps. Firstly, they iteratively hybridise the LWD heuristic into the first half of a sequence based on SD heuristic. At the end of each iteration, if the solution obtained was feasible, the hybridisation amount was increased by 0.03. On the other

hand, if that solution was feasible but had a higher cost than the best solution previously found, the hybridisation amount was decreased by 0.01. Secondly, they hybridised the entire sequence with LWD. However, in this step only the best sequences obtained from the previous step were used.
The paper is organized as follows. Section 2 presents the mathematical formulation of the exam timetabling problem that we adopted in this work. It is followed, in section 3, by a brief description of the key concepts for a better understanding of the proposed method. Section 4 contains all the details about the construction algorithm and how to combine all the basic heuristic values using Choquet integrals. Afterwards, a description of the experimental design is given in Section 5, as well as the experimental results and discussion. Finally, the conclusions are drawn in Section 6.

2 Exam Timetabling Problem Definition

The exam timetabling problem can be formulated as a combinatorial optimisation problem. In order to compare our approach with other methods proposed in the literature, we adopt the following formulation followed by many authors. Let,

$$E = \text{total number of exams}, \tag{1}$$

$$P = \text{total number of periods}, \tag{2}$$

$$S = \text{total number of students}, \tag{3}$$

$$c_{ij} = \text{number of students enroled in exam } i \text{ and } j \text{ for } i, j = 1, \ldots, E, \tag{4}$$

$$a_{ij} = \begin{cases} 1 & \text{if } c_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i, j = 1, \ldots, E, \tag{5}$$

and consider the following variables:

$$x_i = \text{ period in which exam } i \text{ is scheduled }, i = 1, \ldots, E \tag{6}$$

The formulation is:

$$\min f = \frac{\sum_{i=1}^{E} \sum_{j=1}^{E} proximity\_cost(x_i, x_j) * c_{ij}}{2S} \tag{7}$$

$$\text{subject to:} \quad |x_i - x_j| \geq a_{ij} \text{ for } i, j = 1, \ldots, E \text{ and } i \neq j \tag{8}$$

$$1 \leq x_i \leq P \text{ and integer} \quad \text{for } i, j = 1, \ldots, E \tag{9}$$

where proximity cost is defined as:

$$proximity\_cost(x_i, x_j) = \begin{cases} 2^{5-|x_i - x_j|} & \text{if } 0 < |x_i - x_j| < 6 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

for $i, j = 1, \ldots, E$ and $i \neq j$

The objective function (7) penalizes the proximity of exams with students in common, using as weights the number of students involved in both examinations and a factor that depends on the proximity of the periods, ranging from 16 to 0 (Carter et al. 1996). The constraint 8 ensures that any two exams indexed by i and j with students in common are not assigned to the same period.

3 Essential Concepts

Let us briefly present some useful concepts for our work before we describe the construction algorithm in more detail.

3.1 Fuzzy Sets

Definition 1 (Zimmermann (1996)). *If $\mathcal{X}$ is a collection of objects designated by x then a fuzzy set $\widetilde{A}$ in X is defined by a set of pairs:*

$$\widetilde{A} = \{(x, \mu(x)) | x \in X\}$$

where $\mu_{\widetilde{A}}(x)$ is the membership function of x em $\widetilde{A}$.

From now on we are going to refer to a membership function as $f_{\widetilde{A}}$ instead of the traditional way, as presented in the previous definition, $\mu_{\widetilde{A}}$ to avoid any confusion with the representation of a fuzzy measure μ (see Section 3.3).
For example we can consider the age of a person. Let X be the age domain and x the age of a certain person. Then the fuzzy set YOUNG may be defined by:

$$\widetilde{A} = \{(x, f(x)) | x \in X\}$$

where

$$f_{\widetilde{A}}(x) = \begin{cases} 0 \text{ , if } & x \geq 65 \\ \frac{65-x}{30} \text{ , if } & 35 \leq x \leq 65 \\ 1 \text{ , if } & x \leq 35 \end{cases}$$

Fuzzy sets are often represented by triangular, trapezoidal (triangular as a particular case) or gaussian membership functions. The membership function for YOUNG presented above is an example of a trapezoidal function. Generalising, a trapezoidal function is given by the following membership function:

$$f_{\widetilde{A}} : D \subset X \rightarrow [0, 1]$$

where

$$f_{\widetilde{A}}(x) = \begin{cases} 0 & \text{, if } x \leq a \vee x > d \\ \frac{x-a}{b-a} & \text{, if } a < x \leq b \\ 1 & \text{, if } b < x \leq c \\ \frac{d-x}{d-c} & \text{, if } c < x \leq d \end{cases} \tag{11}$$

where $x \in D \subset \mathcal{X}$ and $\widetilde{A}$ is a fuzzy set in $\mathcal{X}$.
The triangular function is a trapezoidal function where $b = c$.

3.2 Linguistic Variable

The linguistic values or terms of a linguistic variables are concepts defined by words or expressions of a natural language.

Definition 2 (Zadeh (1975)). *A linguistic variable is characterized by the quintuple* $(\mathcal{H}, T(\mathcal{H}), U, G, M)$ *, where* $\mathcal{H}$ *is the name of the variable,* $T(\mathcal{H})$ *is the set of terms or linguistic values of* $\mathcal{H}$*,* U *is the universe of the variable,* G *the semantic rule that generates the terms in* $T(\mathcal{H})$ *and* M *is the semantic rule associating to each term or linguist value its meaning trough the fuzzy set* $M(X)$ *(*$M(X)$ *is a fuzzy set on* U*).*

Let us consider the linguistic variable TEMPERATURE as in (Klir and Yuan 1995). We can have a pure numerical interpretation for this concept as depicted in case (b) in Figure 1, but we can represent it as a linguistic variable (case (a)), characterised by the linguist values { Very Low, Low, Average, High, Very High}.

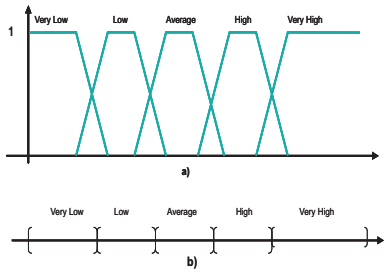


Fig. 1: a) Temperature as Linguistic Variable - b) Numerical representation of Temperature

3.3 Fuzzy Measures

Since here we are working with finite spaces, we are going to present a simplified definition of a fuzzy measure. More details about this topic can be found in Wang and Klir (1992).

Definition 3 (Grabisch (1995)). *A fuzzy measure* μ *defined on the measurable space* $(X, \mathfrak{X})$ *is a set function* $\mu : \mathfrak{X} \to [0, 1]$ *satisfying the following axioms:*

i $\mu(\emptyset) = 0$*,* $\mu(X) = 1$*. This is the usual convention, although in general* $\mu(X)$ *can be any positive finite (or infinity) quantity.*

ii $A \subseteq B \Rightarrow \mu(A) \leq \mu(B)$ *(monotonicity).*

where $X \triangleq \{x_1, \ldots, x_n\}$*, and generally* $\mathfrak{X}$ *is a* σ*-algebra on a space* X*.*
$(X, \mathfrak{X}, \mu)$ *is said to be a fuzzy measure space.*

In this work, we assume that the σ-algebra $\mathfrak{X}$ is the power set of X. Hence, in this case we have $X = \{SD, CD, LD, LWD, LE\}$ and $\mathfrak{X} = \mathcal{P}(X)$.

3.4 Choquet Integral

Again, as in Section 3.3, we present a definition of Choquet integral for the particular case of discrete spaces.

Definition 4. *(Grabisch (1995)): Let $(X, \mathfrak{X}, \mu)$ be a fuzzy measure space. The Choquet integral of a function $f : X \to [0, 1]$ with respect to μ is defined by*

$$\mathfrak{C}_\mu(f(x_1), ..., f(x_n)) \triangleq \sum_{i=1}^{n} (f(x_{(i)}) - f(x_{(i-1)}))\mu(A_{(i)})$$

where $\cdot_{(i)}$ indicates a permutation such that $0 \leq f(x_{(1)}) \leq \ldots \leq f(x_{(n)}) \leq 1$, $A_{(i)} \triangleq \{x_{(i)}, \ldots, x_{(n)}\}$, and $f(x_{(0)}) = 0$.

To better illustrate how Choquet integral works we present the same example described in Murofushi and Sugeno (1989) paper. Consider that there is a rare book collection which consists of two volumes (let us label y_1 as volume 1, and y_2 as volume 2). There is a bookseller who is interested in buying this rare collection. Therefore, he offers: $\mu(\{y_1\})$ monetary units (m.u.) per volume 1; $\mu(\{y_2\})$ m.u. per volume 2; and $\mu(\{y_1, y_2\})$ m.u. for each entire collection. Obviously, he sets a higher values for the complete set, i.e. $\mu(\{y_1, y_2\}) \geq \mu(\{y_1\}) + \mu(\{y_2\})$. Suppose now, that there is a person that sells $x_1 = h(y_1)$ units of volume 1 and $x_2 = h(y_2)$ units of volume 2, where $x_1 \leq x_2$. We can say that he offers x_1 complete collections and $(x_2 - x_1)$ volumes 2. Therefore, he would get $x_1 \times \mu(\{y_1, y_2\}) + (x_2 - x_1) \times \mu(\{y_2\})$ m.u.

4 Construction Heuristic

We implemented a simple construction algorithm. The construction heuristic block is composed by two secondary order heuristics. These are: exam ordering heuristic; and period ordering heuristic. The former one concerns the order in which exams are scheduled. As the latter chooses in which period a particular exam should be scheduled. Furthermore, we also implemented a backtracking procedure which is based on the one described in Carter et al. (1996). A pseudocode of the algorithm is depicted in Algorithm 1.

Algorithm 1 Construction algorithm pseudocode

```
unscheduled_exams ← all exams
while unscheduled_exams is not empty do
  Select an exam from the list unscheduled_exams using a exam_ordering_heuristic
  Select a feasible period to schedule the previous selected exam using a
  period_ordering_heuristic
  if it is impossible to select a feasible period then
    Use the backtracking procedure described in Carter et al. (1996)
    if it is impossible to backtrack then
      break
    end if
  end if
  Update unscheduled_exams list
end while
```

The four period ordering heuristics implemented in this work are the following:

1. random: a random feasible period is chosen using an uniform distribution;
2. first period: the first feasible period is chosen;
3. biased: a roulette-wheel scheme is used to choose the period. The weights are computed using Eq. 7;
4. deterministic and random: the periods are sorted according to the weights obtained by using Eq. 7. Ties are broken randomly.

We implemented three different types of exam ordering. The logic behind them is similar to the one used for period ordering. The description of the heuristics is given below.

1. random: a random exam is chosen from the unscheduled ones using a uniform distribution;
2. biased: a roulette-wheel scheme is used to choose one of the unscheduled exams using an empirical distribution based on one of six heuristics described below;
3. deterministic and random: the exams are sorted according to one of six heuristics described below. Ties are broken randomly.

The heuristics implemented represent how "hard" it is to schedule a particular exam. Each one of the five basic heuristics are described below.

1. Saturation Degree (SD): increasingly order exams by the number of feasible periods in which an exam can be scheduled;
2. Colour Degree (CD): decreasingly order exams by the number of total conflicts that an exam has with the already scheduled exams;
3. Largest Degree (LD): decreasingly order exams by the number of total conflicts;
4. Largest Weighted Degree (LWD): decreasingly order exams by the number of total conflicts weighted by the number of students involved in each one;
5. Largest Enrolment (LE): decreasingly order exams by the number of enrolments;

4.1 Construction Heuristic using Choquet Integral

The more traditional aggregation methods (e.g. Weighted Sum, OWA) are easy to interpret but are too restrictive since they are not able to represent the interaction between the criteria. It is assumed that the criteria is independent, when in most practical cases that does not happen. The motivation to use this method is so that we can represent the information given by the individual heuristics (given above), as well as the interaction between those heuristics by using a fuzzy measure (see Section 3.3) in an understandable way. This way we can model criteria that are not independent.

Choquet integral can be regarded as an extension of Lebesgue integral. That is, if the measure at hand is additive, or in other words, it is a classical measure, then Choquet integral coincides with Lebesgue integral (Murofushi and Sugeno 1989). Therefore, it is easier to interpret its output than other fuzzy integrals. Moreover, Choquet integral has some algebraic properties that other fuzzy integrals do not have (Grabisch 1995). Hence, it makes it more suitable for multicriteria decision making problems (Murofushi and Sugeno 1989; Grabisch 1995, 1996).

Since the information given by the heuristics had different units, we used fuzzy sets (see Section 3.1) to normalise its values into the unit interval. The goal of using this information is to decide which exam is "harder" to schedule. Bearing that in mind, we modeled each basic heuristic as a linguistic variable (see Section 3.2). Hence, we used a triangular membership function (see Eq. 11) to represent the linguistic terms, such as "low SD" value, "high CD" value, "high LD" value, "high LWD" value and "high LE" value. The membership functions are relative to each iteration, e.g., if the highest SD value is 5 (in one particular iteration) all exams with that value are going to have a membership of 0. The membership functions for each linguistic term are defined below.

$$f_{\widetilde{lowSD}}(x_{SD}) = \frac{maxSD - x_{SD}}{maxSD} \tag{12}$$

$$f_{\widetilde{highCD}}(x_{CD}) = \frac{x_{CD}}{maxCD} \tag{13}$$

$$f_{\widetilde{highLD}}(x_{LD}) = \frac{x_{LD}}{maxLD} \tag{14}$$

$$f_{\widetilde{highLWD}}(x_{LWD}) = \frac{x_{LWD}}{maxLWD} \tag{15}$$

$$f_{\widetilde{highLE}}(x_{LE}) = \frac{x_{LE}}{maxLE} \tag{16}$$

Where $maxSD$, $maxCD$, $maxLD$, $maxLWD$ and $maxLE$ is the maximum SD, CD, LD, LWD, LE value in the current iteration, respectively.
Moreover, if some of the $maxCD$, $maxLD$, $maxLWD$, $maxLE$ values are equal to zero, in some iteration, the respective function returns 0 by default. On the other hand, if $maxSD$ value is equal to zero the function returns 1 by default. The weights presented in Table 1 and 2 were chosen using a "rule of thumb". We set the individual weights according to how each individual heuristic performed . The interaction's weights were defined by analysing the information given by the heuristics, i.e., if the heuristics present some kind of complementary information, the weight given to that interaction should be at least higher than the sum of the individual weights. For example, the SD and CD heuristic values capture different information of the timetable being constructed. One could expect that the interaction between these two criteria to be synergetic, i.e. it provides a better understanding of the problem than both heuristics separately. Hence, the weight given to the interaction between these two is higher than the sum of both together. On the other hand, if heuristics present similar information a lower weight should be given. For instance, the LD and LWD heuristic values share, to some degree, the same information; hence the weight given to the interaction between these two is less than the sum of both together. For both individual and interaction weights, we used a trial-and-error approach based on how well the heuristic performed on the hec-s-92 data set.

Table 1: Individual and two-way interaction weights for μ fuzzy measure

Weight	Individual Criterion	Weight	Criteria
0	empty	0.51	SD,LD
0.5	SD	0.515	SD,LWD
0.01	LD	0.52	SD,LE
0.015	LWD	0.8	SD,CD
0.02	LE	0.02	LD,LWD
0.2	CD	0.04	LD,LE
		0.21	LD,CD
		0.045	LWD,LE
		0.3	LWD,CD
		0.32	LE,CD

Table 2: Interaction's weights for μ fuzzy measure

Weight	Criteria	Weight	Criteria
0.6	SD,LD,LE	0.7	SD,LD,LWD,LE
0.62	SD,LWD,LE	0.9	SD,LD,LWD,CD
0.85	SD,LD,CD	0.98	SD,LD,LE,CD
0.88	SD,LWD,CD	0.95	SD,LWD,LE,CD
0.9	SD,LE,CD	0.6	LD,LWD,LE,CD
0.35	LD,LWD,CD	1	SD,LD,LWD,LE,CD
0.06	LD,LWD,LE		
0.4	LD,LE,CD		
0.43	LWD,LE,CD		
0.55	SD,LD,LWD		

With all the values fuzzyfied and a fuzzy measure set we can use the Choquet Integral (see Section 3.4) to combined all the information. As we did with the other basic heuristics, the exams are ordered decreasingly according to the values obtained by using this method. That is, if one exam is attributed value 1 it means that it is very hard to schedule, according to the information given by the five basic heuristics. Hence it should be scheduled before all other exams.
To better illustrate how the process works, an example is here presented. Consider two exams,e_1 and e_2. After the basic heuristic values were computed and normalized, we obtained the following values: $x_{e_1} = (0.4, 0.5, 0.5, 0.6, 0.8)$ and $x_{e_2} = (0.8, 0.4, 0.7, 0.7, 0.4)$, corresponding to the SD, CD, LD, LWD, LE heuristic, respectively. The Choquet integral value for exam e_1 is computed as:

$$
\begin{aligned}
\mathfrak{C}_\mu(x_{e_1}) =& \mu_{\{SD,CD,LD,LWD,LE\}} x_{e_1 SD} + \\
&+ \mu_{\{CD,LD,LWD,LE\}}(x_{e_1 CD} - x_{e_1 SD}) + \\
&+ \mu_{\{LD,LWD,LE\}}(x_{e_1 LD} - x_{e_1 CD}) + \\
&+ \mu_{\{LWD,LE\}}(x_{e_1 LWD} - x_{e_1 LD}) + \\
&+ \mu_{\{LE\}}(x_{e_1 LE} - x_{e_1 LWD}) = \\
=& 1 \times 0.4 + 0.6 \times (0.5 - 0.4) + 0.35 \times (0.5 - 0.5) + \\
&+ 0.045 \times (0.6 - 0.5) + 0.02 \times (0.8 - 0.6) = \\
=& 0.4685
\end{aligned}
$$

An analogous process is also applied to exam e_2, giving the value $\mathfrak{C}_\mu(x_{e_2}) = 0.6150$. Hence, the next exam to be scheduled would be the exam e_2.

5 Experimental Design

To test the performance of the algorithm we used two data sets. The first one was a collection of real problems and is available in an online repository created by Michael Carter[1]. The second data set was put online for the International Timetabling Competition[2] (ITC 2007) (McCollum et al. 2007). In this work, we used the average performance to test whether the algorithm performs significatively better when distinct heuristics are used for ordering exams. For this purpose, we performed 330 runs for each heuristic and for each data file. In total, we ran the algorithm 41580 times ($330 runs \times 6 heuristics \times 21 data files$). For each data file and each heuristic, we computed an average of 33 runs. This way, we obtained 10 samples of the average performance of the algorithm for each data file and each heuristic. We assumed that the 10 samples were normally distributed since we were dealing with averages of 33 random independent and identically distributed variables. Hence, to compare the performance of different heuristics we used the statistical t-test with a significance level of 0.95. The following null hypothesis was used:

$$H0 : \mu_{dfi1_{hj1}} \geq \mu_{dfi2_{hj2}} \tag{17}$$

where μ represents the mean value and $dfi1, dfi2, hj1, hj2$ are the data files $i1, i2 \in$\{Carter's data files, ITC's data files\} and the heuristics $j1, j2 \in$\{SD, CD, LD, LWD, LE, CI\}, respectively.
In the next section we only present the results regarding the determinist order of exams and periods. The results of the other ordering strategies were considerably worse when compared to the aforementioned ones.

5.1 Computational Results

Tables 3 and 4 depict the minimum (min), maximum (max) and average (avg) values over the 330 runs for all data files (lines) and for each heuristic (columns). The best results are presented in boldface.
We also computed how many time each heuristic was statistically better than other k heuristics (with $k = 0, 1, 2, 3, 4, 5$) across all 21 data files. The results are depicted in Figure 2. For instance, we can observe that SD heuristic (see Figure 2a) performed better than 1 heuristic in 2 data files and was also better than all other heuristics in the other 2 data files.

5.2 Discussion

From analysing Figure 2 it can be seen that amongst the basic heuristics, the one that performs better is the SD heuristic followed by the CD heuristic.

[1] ftp://ftp.mie.utoronto.ca/pub/carter/testprob
[2] http://www.cs.qub.ac.uk/itc2007/

Table 3: Computational results of the basic and Choquet heuristics for the Carter's data set

Data Set		SD	CD	LD	LWD	LE	CI
car-f-92	min	4.56	4.74	5.15	5.17	5.03	**4.44**
	max	**6.17**	7.51	7.61	7.44	7.54	6.99
	avg	**5.07**	5.77	6.07	6.06	6.12	5.28
car-s-91	min	5.25	5.39	5.54	5.69	5.83	**5.18**
	max	7.15	7.76	8.77	8.33	8.34	**6.95**
	avg	5.73	6.25	6.70	6.65	6.82	**5.66**
ear-f-83	min	40.74	40.22	41.02	43.65	43.67	**39.55**
	max	58.78	61.46	58.73	60.06	61.34	**57.53**
	avg	46.56	49.00	49.13	51.03	51.04	**45.43**
hec-s-92	min	12.59	12.68	14.16	14.14	12.73	**12.20**
	max	**22.80**	26.61	27.14	23.15	23.40	25.27
	avg	**16.07**	17.22	18.41	17.53	18.11	16.48
kfu-s-93	min	15.92	16.12	16.41	16.05	16.70	**15.46**
	max	26.00	29.44	26.12	26.81	26.98	**22.98**
	avg	18.55	19.23	20.57	20.00	20.44	**17.54**
lse-f-91	min	11.96	12.03	12.95	12.28	12.48	**11.83**
	max	19.25	19.43	19.25	20.01	19.08	**15.55**
	avg	14.25	14.81	14.24	14.96	15.30	**12.89**
pur-s-93	min	4.95	4.96	5.04	5.05	5.15	**4.93**
	max	5.85	6.01	6.05	7.42	7.82	**5.81**
	avg	5.33	5.38	5.37	5.58	5.64	**5.19**
rye-s-93	min	10.48	10.65	12.35	10.33	10.92	**10.04**
	max	17.84	19.15	19.02	18.49	19.17	**16.07**
	avg	12.78	13.35	15.07	13.76	14.14	**11.85**
sta-f-83	min	**159.35**	159.71	162.11	163.09	161.71	160.50
	max	185.63	**182.97**	199.70	192.51	202.30	184.89
	avg	169.94	169.95	180.09	173.50	174.71	**169.46**
tre-s-92	min	8.90	9.04	10.18	9.25	9.47	**8.71**
	max	12.40	13.12	13.34	13.73	12.67	**11.07**
	avg	10.10	10.76	11.66	11.20	11.27	**9.27**
uta-s-92	min	3.64	3.67	4.04	3.76	3.86	**3.49**
	max	**4.69**	5.49	6.09	6.80	6.70	5.17
	avg	3.95	4.22	4.83	4.74	4.84	**3.80**
ute-s-92	min	28.93	**28.65**	31.34	29.55	29.45	29.44
	max	43.78	45.25	45.21	46.60	49.48	**39.15**
	avg	34.91	34.09	36.82	37.12	36.55	**33.44**
yor-f-83	min	43.29	43.07	45.27	46.38	45.74	**42.19**
	max	56.59	59.20	58.13	56.27	58.62	**54.77**
	avg	49.10	50.64	51.65	51.07	51.46	**47.94**

This can be somewhat explained since these two heuristics have a dynamic behaviour, while the other three are static, as mentioned in Section 4.

As it can be observed in Table 3 and 4, the CI heuristic almost always obtained the best results across all data sets. It obtained the best minimum, maximum and average results in 15, 17 and 18 out of 21 data files, respectively. Figure 2f shows that the CI heuristic was significatively better than all of other heuristics in 81% of the instances. Moreover, this heuristic was better than at least 4 heuristic in 90% of the cases.

Only two of the constructive methods (Burke and Newall 2004; Qu et al. 2009a) seem to perform better than the rest. However, as was described in Section 1, all methods (with the exception of Carter and Laporte (1996)) have incorporated a

Table 4: Computational results of the basic and Choquet heuristics for the International Timetabling Competition (ITC) data set

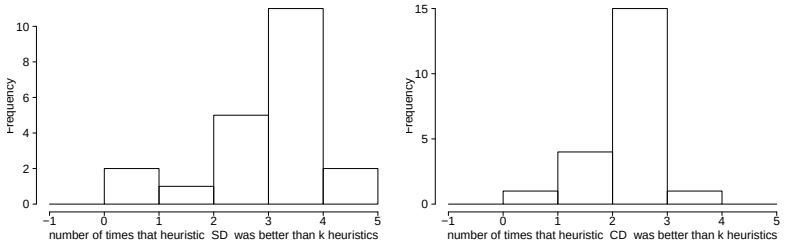
Data Set		SD	CD	LD	LWD	LE	CI
ITC1	min	1.20	1.20	1.30	1.22	1.22	**1.12**
	max	1.51	1.49	1.55	1.42	1.44	**1.30**
	avg	1.36	1.36	1.42	1.31	1.33	**1.21**
ITC2	min	0.28	0.28	0.30	0.28	0.28	**0.26**
	max	0.40	0.38	0.40	0.39	0.39	**0.35**
	avg	0.34	0.33	0.35	0.33	0.34	**0.30**
ITC3	min	1.91	1.90	1.99	1.88	1.93	**1.81**
	max	2.29	2.31	2.29	2.25	2.23	**2.06**
	avg	2.10	2.09	2.10	2.07	2.07	**1.93**
ITC4	min	13.99	13.66	14.49	15.77	15.91	**13.54**
	max	**22.82**	24.63	25.28	28.10	29.59	28.49
	avg	16.91	17.48	18.46	20.95	20.90	**16.67**
ITC5	min	0.49	0.50	0.59	0.53	0.55	**0.44**
	max	0.72	0.73	0.82	0.69	0.74	**0.61**
	avg	0.60	0.60	0.71	0.60	0.63	**0.52**
ITC6	min	4.63	4.78	4.87	**4.43**	4.67	4.50
	max	6.09	6.29	6.88	7.65	7.41	**5.49**
	avg	5.27	5.34	5.63	5.47	5.77	**4.90**
ITC7	min	**0.10**	0.11	0.13	0.14	0.14	0.11
	max	0.17	0.17	0.19	0.19	0.19	**0.16**
	avg	0.14	0.14	0.16	0.17	0.16	**0.13**
ITC8	min	**0.18**	0.18	0.23	0.22	0.24	0.18
	max	0.32	0.33	0.33	0.33	0.35	**0.30**
	avg	**0.25**	0.25	0.28	0.28	0.30	0.25

more sophisticated method to improve the construction process. Therefore, the heuristic described in this work held a much greater potential since the fuzzy measure used in this work was not a subject of any kind of sophisticated tuning procedure. Nevertheless, if we compare the results obtained by the CI heuristic with some other constructive methods described in the literature (which are depicted in table 5) we can observe that it presents very competitive results in most of the instances of the Carter's data set. Moreover, the CI heuristic is faster (see Table 6) than most of the construction heuristics depicted in Table 5. This makes it suitable to be used as a generator for population based algorithms since it builds good quality timetables.

6 Conclusions

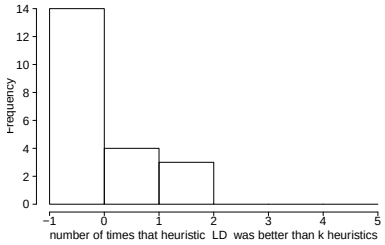
In this work we presented a construction algorithm which uses a fuzzy measure and Choquet integral to combine the information given by 5 basic heuristics (see Section 4). The exams are then decreasingly ordered according to the value obtained by the Choquet integral and scheduled into a time period. This is chosen in virtue of minimising the total cost of the timetable (which is given by Equation 7).

The new method proposed in this work performs better than all basic heuristics in most of the test instances. However, in some of them the SD heuristic obtained better results. Nevertheless, we expect to enhance the performance of

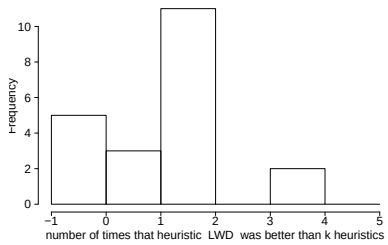


(a) SD heuristic

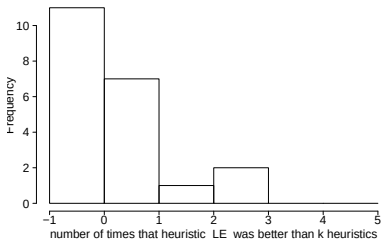
(b) CD heuristic



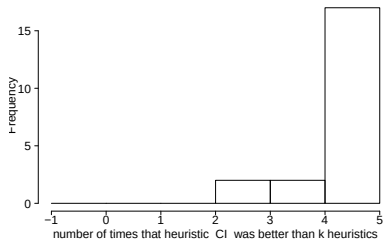
(c) LD heuristic



(d) LWD heuristic



(e) LE heuristic



(f) CI heuristic

Fig. 2: Histogram of the number of times that each heuristic performed better than k others

CI heuristic by using other techniques (e.g. differential evolution (Price et al. 2005)) to tune the weights of the fuzzy measure. Moreover, instead of tuning the weights for each instance we can use a training data set. Through this, we expect the heuristic to perform well across a different range of instances with different characteristics.

Acknowledgements This research has been supported by Fundação para a Ciência e Tecnologia (FCT) Portugal grant number SFRH/BD/43486/2008.

Table 5: Best computational results of some constructive methods and the Choquet heuristic for the Carter's data set

Data Set	(Carter and Laporte 1996)	(Burke and Newall 2004)	(Asmuni et al. 2009)	(Qu et al. 2009a)	Best reported (Qu et al. 2009b)	CI
car-f-92	6.2	4.32	4.54	4.32	**3.93**	4.44
car-s-91	7.1	4.97	5.29	5.11	**4.5**	5.18
ear-f-83	36.4	36.16	37.02	35.56	**29.3**	39.55
hec-s-92	10.8	11.61	11.78	11.62	**9.2**	12.20
kfu-s-93	14.0	15.05	15.80	15.18	**13.0**	15.46
lse-f-91	10.5	10.96	12.09	11.32	**9.6**	11.83
pur-s-93	**3.9**	-	-	-	-	4.93
rye-s-93	7.3	-	10.38	-	**6.8**	10.04
sta-f-83	161.5	161.91	160.42	158.88	**134.9**	160.50
tre-s-92	9.6	8.38	8.67	8.52	**7.9**	8.71
uta-s-92	3.5	3.36	3.57	3.21	**3.14**	3.49
ute-s-92	25.8	27.41	28.07	28.00	**24.4**	29.44
yor-f-83	41.7	40.77	39.80	40.71	**36.2**	42.19

Table 6: Computational times (in seconds) for the CI heuristic. The values are an average of 300 runs.

Data Set	car-f-92	car-s-91	ear-f-83	hec-s-92	kfu-s-93	lse-f-91	pur-s-93
Times	5.03	9.13	0.34	0.15	2.89	1.66	382.81
Data Set	rye-s-93	sta-f-83	tre-s-92	uta-s-92	ute-s-92	yor-f-83	-
Times	3.42	0.11	0.60	6.97	0.24	0.43	-

References

D. Abramson, M. Krishnamoorthy, and H. Dang. Simulated annealing cooling schedules for the school timetabling problem. *Asia Pacific Journal of Operational Research*, 16:1–22, 1999.

H. Asmuni, E.K. Burke, J.M. Garibaldi, B. McCollum, and A.J. Parkes. An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers and Operations Research*, 36(4):981–1001, 2009.

R. Awad and J. Chinneck. Proctor Assignment at Carleton University. *Interfaces*, 28 (2):58–71, 1998.

E. K Burke and J. P Newall. Enhancing Timetable Solutions with Local Search Methods. *Lectue Notes in Computer Science*, pages 195–206, 2003.

E. K. Burke and J. P. Newall. Solving examination timetabling problems through adaption of heuristic orderings. *Annals of operations Research*, 129(1):107–134, 2004.

EK Burke, J. Kingston, and D. de Werra. Applications to timetabling. *Handbook of Graph Theory*, pages 445–474, 2004.

E.K. Burke, A.Eckersley, B.McCollum, S.Petrovic, and R.Qu. Hybrid variable neighbourhood approaches to university exam timetabling. Technical Report NOTTCS-TR-2006-2, School of Computer Science and IT, University of Nottingham, 2006.

A. Caprara, M. Fischetti, PL Guida, M. Monaci, G. Sacco, and P. Toth. Solution of real-world train timetabling problems. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, page 10, 2001.

M. Caramia, P. DellÓlmo, and G.F. Italiano. New algorithms for examination timetabling. In *Algorithm Engineering*, volume 1982 of *Lecture Notes in Computer Science*, pages 230–242. Springer, 2001.

M. W. Carter, G. Laporte, and J.W. Chinneck. A general examination scheduling system. *Interfaces*, 24(3):109–120, 1994.

Michael W. Carter and Gilbert Laporte. Recent developments in practical examination timetabling. In *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling*, pages 3–21, London, UK, 1996. Springer-Verlag. ISBN 3-540-61794-9.

M.W. Carter, G. Laporte, and S.Y. Lee. Examination Timetabling: Algorithmic Strategies and Applications. *Journal of the Operational Research Society*, 47(3): 373–383, 1996.

S. Casey and J. Thompson. Grasping the examination scheduling problem. In *Practice and Theory of AutomatedTimetabling IV*, volume 2740/2003 of *Lecture Notes in Computer Science*, pages 232–244. Springer Berlin / Heidelberg, 2003.

A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for High School Timetabling. *Computational Optimization and Applications*, 9(3):275–298, 1998.

PH Corr, B. McCollum, MAJ McGreevy, and P. McMullan. A New Neural Network Based Construction Heuristic for the Examination Timetabling Problem. *Lecture Notes in Computer Science*, 4193:392, 2006.

L. Di Gaspero. Recolour, shake and kick: A recipe for the examination timetabling problem. In *Proceedings of the fourth international conference on the practice and theory of automated timetabling*, pages 404–407, Gent, Belgium, 2002.

Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In *PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, pages 104–117, London, UK, 2001. Springer-Verlag. ISBN 3-540-42421-0.

KA Dowsland and JM Thompson. Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, 56(4):426–438, 2004.

K. Easton, G. Nemhauser, and M. Trick. Sports scheduling. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2004.

W. Erben. A Grouping Genetic Algorithm for Graph Colouring and Exam Timetabling. *Lecture Notes in Computer Science*, pages 132–158, 2001.

M. Grabisch. Fuzzy integral in multicriteria decision making. *Fuzzy Sets and Systems*, 69(3):279–298, 1995.

M. Grabisch. The application of fuzzy integrals in multicriteria decision making. *European journal of operational research*, 89(3):445–456, 1996.

M.P. Hansen and R.V.V. Vidal. Planning of high school examinations in Denmark. *European Journal of Operational Research*, 87(3):519–534, 1995.

MT Isaai and MG Singh. Hybrid applications of constraint satisfaction and metaheuristicsto railway timetabling: a comparative study. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 31(1):87–95, 2001.

Graham Kendall and Naimah Mohd Hussin. An investigation of a tabu search based hyper-heuristic for examination timetabling. In Graham Kendall, Edmund K. Burke, and Sanja Petrovic, editors, *Selected papers from MISTA 2003*. Kluwer Publication, 2004. URL `http://www.asap.cs.nott.ac.uk/publications/pdf/nbh_mista03_extend.pdf`.

G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. Prentice Hall, 1995.

Barry McCollum, Paul McMullam, Edumund K. Burke, Andrew J. Parkes, and Rong Qu. The second international timetabling competition: Examination timetabling track. Technical Report QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, Queen's University Belfast, September 20 2007.

L.T.G. Merlot, N. Boland, B.D. Hughes, and P.J. Stuckey. A hybrid algorithm for the examination timetabling problem. In *Practice and Theory of Automated-Timetabling IV*, volume 2740/2003 of *Lecture Notes in Computer Science*, pages 207–231. Springer Berlin / Heidelberg, 2003.

T. Murofushi and M. Sugeno. An interpretation of fuzzy measures and the Choquet integral as an integral with respect to a fuzzy measure. *Fuzzy sets and Systems*, 29(2):201–227, 1989.

Luis F. Paquete and Carlos M. Fonseca. A sutdy of examination timetabling with multiobjective evolutionary algorithms. In *Proccedings of 4th Metaheuristics International Conference*, 2001.

Sanja Petrovic and Yuri Bykov. A multiobjective optimisation technique for exam timetabling based on trajectories. In *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT 2002)*, volume 2740 of *Lecture Notes in Computer Science*, pages 179–192, Gent, Belgium, Aug 21-23 2002. Springer. URL `http://www.asap.cs.nott.ac.uk/publications/pdf/yuri_Patat02sv.pdf`.

Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., 2005. ISBN 3540209506.

X. Qi, J. Yang, and G. Yu. Scheduling problems in the airline industry. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2004.

R. Qu, E.K. Burke, and B. McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404, 2009a.

R. Qu, EK Burke, B. Mccollum, LT Merlot, and SY Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):89, 2009b.

G. Ribeiro Filho and L.A.N. Lorena. A Constructive Evolutionary Approach to School Timetabling. *Applications of Evolutionary Computing - Lecture Notes in Computer Science*, 2037:130–139, 2001.

A. Schaerf. Local search techniques for large high school timetabling problems. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 29(4):368–377, 1999.

K. Schimmelpfeng and S. Helber. Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum*, 29(4):783–803, 2007.

J.M. Thompson and K.A. Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations research*, 63(1):105–128, 1996.

Jonathan M. Thompson and Kathryn A. Dowsland. A robust simulated annealing based examination timetabling system. *Comput. Oper. Res.*, 25(7-8):637–648, 1998. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/S0305-0548(97)00101-9.

M.A. Trick. A Schedule-Then-Break Approach to Sports Timetabling. *Lecture Notes in Computer Science*, pages 242–253, 2001.

Zhenyuan Wang and George J. Klir. *Fuzzy measure theory*. Plenum Publishing Corporation, 1992.

George M. White and Bill S. Xie. Examination timetables and tabu search with longer-term memory. In *PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, pages 85–103, London, UK, 2001. Springer-Verlag. ISBN 3-540-42421-0.

Yong Yang and Sanja Petrovic. A novel similarity measure for heuristic selection in examination timetabling. In *Proccedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004)*, Pittsburg, August 18-20 2004. URL `http://www.asap.cs.nott.ac.uk/publications/pdf/yxy_Patat04.pdf`.

L.A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, Part I. *Information Sciences*, 8(3):199–249, 1975.

H. J. Zimmermann. *Fuzzy Set Theory and its applications*. Kluwer Academic Publishers, 1996.

An Overview of School Timetabling Research

Nelishia Pillay

School of Computer Science, University of KwaZulu-Natal

+27 2605644

+27 2605648

pillayn32@ukzn.ac.za

Abstract. Although there has been a fair amount of research in the area of school timetabling, this domain has not grown as well as other fields of educational timetabling such as university course and examination timetable. This can possibly be attributed to the fact that the studies in this domain have generally been conducted in isolation of each other and have addressed different school timetabling problems. Furthermore, there have been no comparative studies on the success of different methodologies on a variety of school timetabling problems. As a way forward this paper provides an overview of the research conducted in this domain, details of problems sets which are publically available and proposes areas for further research in school timetabling.

Keywords: school timetabling, educational timetabling

1. Introduction

Educational timetabling encompasses university course timetabling, examination timetabling and school timetabling. A lot of progress has been made in university course and examination timetabling research. This can be attributed to the variety of problems publicly available which has enabled a comparative study of different methodologies for these domains. Research in school timetabling has not advanced as rapidly as in the other two areas of educational timetabling (Kingston 2006). This is possibly due to studies being done in isolation of each other for specific schools (Santos et al. 2008) and the lack of a variety of problems which can be accessed publicly (Schaerf 1991; Smith et al. 2003; Jacobsen et al. 2006; Post et al. 2008). According to Nurmi et al. (2008) and Post et al. (2008) the school timetabling problem has not been studied as extensively as the university course and examination timetabling problems. Furthermore, methods implemented have not been widely tested on a variety of school timetabling problems (Jacobsen et al. 2006; Santos et al. 2005) which is essential in order to ascertain how well the methodology can generalize. Due to the unavailability of a general set of benchmarks, there have not been much comparative analyses of different methodologies in solving the school timetabling problem. In cases where such studies have been performed, different methods are compared for a single school and not on a variety of problems.

This paper serves as a starting point for further development in the field. The definition of the school timetabling problem and the hard and soft constraints associated with each problem instance differs from one study to the next. The following section attempts to provide a standardized definition of the problem. Section 3 presents an overview of research in the field.

Problems sets that are publicly available for research are described in section 4. Section 5 proposes a way forward by proposing future directions of research in school timetabling.

2. The School Timetabling Problem (STP)

The school timetabling problem and the terminology used in defining the problem differs drastically from one study to the next. For consistency the following terms are defined.
A *class* refers to a group of students that will be taught a particular subject, e.g. Mathematics. A *lesson* refers to a particular subject being taught to a class by a teacher. The teaching period refers to the duration of the timetable and is usually a week.

Carter et al. (Carter et al. 1997) define the school timetabling problem as a subtype of course timetabling. Solving the school timetabling problem essentially involves allocating class, teacher and room tuples to timetable slots so as to satisfy the hard and soft constraints of the problem (Abramson et al. 1991; Beligiannis et al. 2008; Post et al. 2008). Students are usually grouped into classes prior to the timetable construction process (Abramson 1991). The school timetabling problem differs for each country due to the characteristics and regulations specific to the particular education system (Alvarez-Valdes et al. 1996; Post et al. 2008). In some cases the room does not form part of the tuple and each class-teacher pair has to be allocated to a period and a venue (Wilke et al. 2008; Post et al. 2008). The number and duration of lessons can also differ for grades or class levels in a school (Post et al. 2008). Furthermore, some schools extend over more than one site (Schaerf 1991).

The hard and soft constraints also differ drastically from one problem to the next. Hard constraints are constraints that must be met by a timetable in order for it to be operable. A timetable meeting all the hard constraints of a problem is called a feasible timetable. Soft constraints define the quality of a timetable. Soft constraints are usually contradictory, and as such it is impossible to satisfy all the soft constraints. Thus, we attempt to minimize the soft constraint cost.

The STP can be defined in terms of the requirements of the problem, the hard constraints and soft constraints. Section 2.1 presents a model for specifying the requirements of the STP. Sections 2.2 and 2.3 provide a comprehensive set of hard and soft constraints, respectively, for the school timetabling problem that should be catered for in a standardized definition of the problem. Some constraints are treated as hard constraints in one STP and soft constraints in another. These are described in section 2.4. Objective functions generally used to evaluate school timetables are discussed in section 2.5.

2.1 Requirements of the Problem

The requirements of the STP differ from problem to problem. This section attempts to describe the requirements as generally as possible so as to accommodate most variations of the STP. The requirements of the STP can be defined in terms of the following:

- General statistics – Number of courses, number of teachers, number of periods or number days and number of periods per day.
- Available periods – The number and duration of available periods. An indication of which session the period falls in, e.g. morning or afternoon or the time of each period.
- Lesson requirements – These requirements specify the number of times a teacher has to meet a class over the teaching period. In most problem instances the requirements are specified in terms of class, teacher, room tuples which must be assigned to periods.
- Room requirements - A variation of this problem does not include rooms in the tuple, and room allocation forms part of the timetable construction process.
  In this case class sizes must be provided. In addition to this a list of rooms and their corresponding capacities must be specified. Certain lessons may require specialized rooms such as laboratories or a gymnasium. These need to be indicated. In some versions of the problem classes have fixed rooms and teachers move from one class to another. In this case classes will only move to specialized rooms or in the case of splits or mergers for lessons.
- Teacher requirements - Each teacher's workload is defined in terms of the maximum and minimum number of periods or hours the teacher teaches for the week. Teachers may also be given a set number of free periods over the teaching period. These must be specified. It a teacher is required to commute between two different school sites in a day this must be indicated.
- Teacher unavailabilities – Teachers are usually unavailable for certain periods. This may be due to other administrative duties. Alternatively, teachers may be allocated certain free periods or days off and thus may not be available. Some schools employ teachers on a part-time basis. In this case teachers will teach at different schools on different days of the week and thus may not be available.
- Teacher preferences – Teachers may have preferences to teach in certain periods and not in others.
- Class requirements - In some cases classes are split into subgroups and each subgroup is taught a different subject simultaneously in different venues. Alternatively, classes are merged together for certain subjects and taught in one large venue. The merger can occur over more than one grade. Another scenario involves splitting one or more classes in a form and rejoining the classes into different subgroups. Each subgroup is taught a different subject in the same period in different venues. These requirements must be specified.

  It may also be necessary for classes to have double or triple (i.e. two or three consecutive periods) lessons for certain subjects. Double and triple period requirements must be specified.

  If a lunch break is not built into the timetable, and the corresponding period is not included in the available periods, a specification indicating the need for a lunch break and the duration and range of periods during which it should be scheduled must be provided.

- Class preferences – There may be certain preferences as to when lessons for particular classes should be held, e.g. Mathematics for lower grades in morning sessions. This must be specified.

2.2 Hard Constraints

The hard constraints for the STP can be described in terms of the hard constraints for classes, teachers, and rooms.

Hard Constraints for Classes

- Every class must be allocated.
- Classes must be scheduled for the required number of meetings for each subject over the teaching period (usually a week).
- Classes must not be scheduled more than once during a period, i.e. there must not be any class clashes.
- Splitting and merging of classes (Beligiannis et al. 2008; Jacobsen et al. 2006; Kingston 2004; Kwok et al. 1997; Marte 2006; Post et al. 2008; Wilke et al. 2008) – Classes may be merged together for a lesson. In some cases classes may have to split into subgroups with each subgroup being taught a different subject simultaneously. The split subgroups may also need to be merged differently from the original configuration. The splitting and merging may take place for the same grade or across grades.
- Sequence of lessons (Melicio et al. 2006) – Certain subjects may have to be taught before or after other subjects.
- One period, in a specific range, should be allocated as a lunch break for pupils (Colorni et al. 1998; Wilke et al. 2008).

Hard Constraints for Teachers

- Teachers must not be scheduled more than once during a period.
- Teachers must be scheduled for the required number of meetings with each class over the teaching period.
- Teachers must only be scheduled when available (Beligiannis et al. 2008; Birbas et al. 1997; Post et al. 2008; Santos et al. 2008; Valouxis et al. 2003; Wilke et al. 2008) – Teachers may be unavailable during certain periods due to administration tasks, teaching in another school, allocated free periods or days off. Teachers must not be scheduled to teach during these periods.
- Teacher workload must be adhered to – The teacher workload is defined in terms of a minimum and maximum number of teaching lessons or hours per week (Birbas et al. 2009; de Haan et al. 2007; Nurmi et al. 2008; Santos et al. 2005; Wilke et al. 2008).
- Time permitted for commutation between schools (Schaerf 1991) – Some schools are located over more than one site. Thus, it may be necessary for teachers to commute

between schools. Thus, time for commutation must be allowed, for example a teacher cannot be scheduled to teach two consecutive periods, each on a different site.

Hard Constraints for Rooms

In some versions of the STP rooms have to be allocated as part of the timetable construction process and are not pre-assigned. The following constraints generally have to be met with respect to venues:

- Room capacities must not be exceeded (Wilke et al. 2008).
- All rooms must be used (Groebner et al. 2003).
- Certain lessons require specialized rooms, e.g. science labs, computer lab, the gymnasium. These requirements must be met. In some problems the specialized rooms are highly utilized, making the STP more difficult (Wilke et al. 2008; Wright 1996; Wood et al. 1998).
- Some schools are located over more than one site. In this case a room at the same location must be allocated for all class-teacher meetings of a subject (Post et al. 2008).

2.3 Soft Constraints

The soft constraints for the STP can be described in terms of the soft constraints for classes and teachers.

Soft Constraints for Classes

- Lesson session preferences – This refers to period preferences for lessons (Melicio et al. 2006; Wright 1996). There may be preferences for some subjects to be taught in morning sessions, e.g. mathematics or afternoon sessions (Colorni et al. 1998), a particular subject should not be taught in the first period of a day (Nurmi et al. 2008).

Soft Constraints for Teachers

- Teacher preferences - A teacher may prefer to teach in certain periods and not in others (Schaerf 1991; Nurmi et al. 2008; Valouxis et al. 2003).

2.4 Hard or Soft Constraints

The following constraints have been treated as hard constraints in some versions of the STP and soft constraints in others:

Constraints for Classes

- Idle or free periods – This constraint differs from one STP to the other. In some cases free periods are not allowed at all (Alvarez-Valdes et al. 1996; Jacobsen et al. 2006; Melicio et al. 2006; Wilke et al. 2008). In other problems only some grades or levels, usually higher grades, can have free periods (Filho et al. 2001). Some problems also

stipulate when the free periods are permitted, e.g. last two periods of the day (Schaerf et al. 1991; Valouxis et al. 2003).

- Lesson spread – Different STPs have different spread requirements for lessons. For example, there may be a restriction of at most one lesson for a subject per day (Alvarez-Valdez et al. 1996; Filho et al. 2001; Melicio et al. 2006; Wright 1996). Alternatively, lessons must not be taught on consecutive days for *n* days (Alvarez-Valdez et al. 1996). Lessons for each subject must be distributed uniformly throughout the week (Alvarez-Valdez et al. 1996; Birbas et al. 2009; Colorni et al. 1998; Wright 1996).
  There must not be more than two daily lessons with the same teacher (Birbas et al. 1997; Santos et al. 2005). The same subject must not be taught in the last period of one day and the first period of the following day (Wright 1996).
- Double or triple lessons (DeHaan et al. 2007; Filho et al. 2001; Jacobsen et al. 2006; Kingston 2004; Melicio et al. 2006; Santos et al. 2008; Schaerf 1991) – It may be necessary to schedule a double (two consecutive lessons) or triple lesson (three consecutive lessons) with a class for a particular subject.

Constraints for Teachers

- Lesson spread – The lessons taught by a teacher should be well-spaced throughout the week (Valouxis et al. 2003). Alternatively, the lessons taught by a teacher should be concentrated over a limited number of days (Birbas et al. 1997; Filho et al. 2001; Santos et al. 2008).
- Idle/Free periods - Teachers are generally allowed some free periods (Wilke et al. 2008; Wright 1996), however the number of free periods for each teacher should be minimized when constructing the timetable (Birbas et al. 2009; de Haan et al. 2007; Post et al. 2008; Wilke et al. 2008).

2.5 Objective Function

One of two objective functions has been used to calculate the cost of the timetable. The first function is basically the sum of the hard and soft constraint violations (Abramson et al. 1991; Valouxis et al. 2003; Wood et al. 1998). The second function used is the weighted sum of the hard and soft constraint violations which allows for some constraints to have higher priority than others (Schaerf 1991; Wright 1996).

3. Solving the School Timetabling Problem

This section examines some of the methodologies that have been used to solve the STP. This survey of techniques is a work-in-progress and is by no means exhaustive. Methods used to solve the school timetabling problem include simulated annealing, evolutionary algorithms, tabu search, integer programming, constraint programming, GRASP (Greedy Randomized Search Procedure), and tiling algorithms. In some cases hybrid approaches, combining the use of two or more methodologies are implemented. Comparative studies, comparing the performance of two or more

techniques in solving a particular STP have also been conducted. This section provides an overview of the different methods and studies.

3.1 Simulated Annealing

Abramson (1991) applies simulated annealing to the school timetabling problem. The atoms correspond to elements of the timetable and the energy to the cost of the timetable.
In order to allow for scheduling to be more flexible, assignments are made to room groups instead of individual rooms. If a group of classes must always take place at the same time, the classes should be scheduled as a group instead of individually. The system was tested on randomly generated problems and data from an Australian school.

Melicio et al. (2006) developed the THOR school timetabling tool to solve the STP for Portuguese schools. THOR firstly creates an initial solution using a heuristic constructive algorithm. This solution is then improved using fast simulated annealing.

3.2 Evolutionary Algorithms

Abramson et al. (1991) use a genetic algorithm to solve the school timetabling problem. A parallel algorithm is applied to speed up the process. Each chromosome consists of *n* periods and each period contains *m* tuples. The mutation operator changes the period of a tuple. Crossover is also applied to two chromosomes by choosing a crossover point in each chromosome and swapping the fragments. One child is returned which contains the first fragment of the first parent and the second fragment of the second parent. Crossover may result in the "label replacement problem", i.e. the child may contain some duplicated and/or missing genes. A label replacement algorithm is used to rectify this problem. The GA was used to solve nine highly constrained school timetabling problems.

Beligiannis et al. (2008) use an adaptive evolutionary algorithm to solve the school timetabling problem. Each element of the population is a matrix with the rows corresponding to the classes and the columns to the periods. Each cell in the matrix stores the teacher that will teach the class in the particular period. Initial studies indicated that crossover was not effective and time consuming and hence it was not used. The period mutation operator swaps the teachers between two time periods for a class. The periods chosen for swapping are randomly selected. The bad period mutation operator does not randomly choose periods, instead the two "most costly" periods in the corresponding teacher timetable are selected. Linear ranking selection is used to choose parents. The best chromosome of each generation is copied into the next generation. This algorithm successfully generated solutions to the Greek high school timetabling problem.

Caldiera et al. (1997) evaluate the use of genetic algorithms (GAs) to solve the STP by applying a GA to a small randomly generated school timetabling problem. An initialization procedure is used to create an initial population of feasible timetables. A GA is used to improve the quality of the initial population. Roulette-wheel selection and an ultra-elitism method are used for selection. Reproduction, mutation and crossover are applied to parents to create the offspring of the next generation. A repair algorithm is applied to offspring to ensure that they are feasible.

Filho et al. (2001) use a constructive genetic algorithm to solve the school timetabling problem for two Brazilian high schools.

Nurmi et al. (2008) convert the curriculum-based university course timetabling problem for the 2nd International Timetabling (ITC '07) into one for school timetabling and use a genetic algorithm to solve this problem. The GA uses a greedy hill-climbing mutation operator to solve problem.

Raghavjee et al. (2008) apply a genetic algorithm to five highly constrained school timetabling problems (Beasley 2010). The algorithm firstly creates an initial population of timetables using a sequential construction method employing the largest degree heuristic. The mutation operator is used to iteratively refine initial population. A variation of tournament selection is used to choose the parents of each generation. The algorithm found solutions for all five problems and produced better results than other methodologies applied to the same set of problems.

Wilke et al. (2002) use a genetic algorithm to solve the German school timetabling problem. The initial population is comprised of potential timetable solutions, i.e. timetables are directly represented. Each chromosome contains the class timetables for the school. Roulette-wheel selection is used to choose parents. An elitist strategy copying the best two individuals into the next generation is also employed. In addition to crossover and mutation operators, a number of hybrid operators are applied. If there is no improvement in the fitness of offspring, a reconfiguration step is performed during which the parameters of the GA are reset.

3.3 Tabu Search

Bello et al. (2008) treat the school timetabling problem as a graph coloring problem. An adjunct graph is created and colored using an adaptation of the Tabu search algorithm for graph coloring (Tabucol), namely, Modified Tabucol (MT). The system was applied to five instances from Brazilian high schools.

In the approach taken by Jacobsen et al. (2006), an initial solution is firstly created using a construction heuristic with a graph coloring algorithm. The initial solution is then improved using Tabu search. The system was tested on data from German high schools.

Santos et al. (2005) firstly apply a constructive algorithm to create an initial solution. A tabu search using an informed diversification strategy is applied to the initial solution to improve the quality of the timetable. The diversification strategy was tested with transition based long term memory and residence based long term memory. The study showed that the use of a diversification strategy improved the quality of the timetable produced by the tabu search. The algorithm was used to solve the STP for Brazilian high schools.

3.4 Integer Programming

Earlier work by Birbas et al. (1997) use integer programming to solve the school timetabling problem for Greek high schools. This work is extended further in Birbas et al. (2009) which takes a hybrid approach to solving the problem. The first phase solves the shift assignment problem in

which teachers are allocated to shifts. The second phase solves the school timetabling problem. Integer programming is used in both phases. The approach was successfully applied to a secondary Hellenic school.

Santos et al. (2008) use mixed integer programming to solve the STP for Brazilian high schools. A cut and column generation algorithm is implemented. The algorithm uses Fenchel cuts.

3.5 Constraint Programming

Valouxis et al. (2003) use constraint programming (CP) in combination with local search to solve the school timetabling problem for Greek high schools. CP is used to find a feasible timetable. The quality of the timetable is then improved using local search until further improvement is not possible. The stopping criterion is a runtime of one hour.

3.6 GRASP

Moura et al. (2010) use GRASP with path-relinking to solve the STP for three Brazilian high schools. GRASP takes a three stage approach to the problem. The first phase ranks lessons. During the second phase the ranking is improved using local search. In the third phase a path-relinking strategy is used to identify optimal solutions. These three phases are repeated a number of times.

3.7 Tiling Algorithms

Kingston (Kingston 2004; Kingston 2006) uses a tiling algorithm in combination with hill-climbing to allocate meetings (teacher and class tuples) and an alternating path algorithm for assigning recourses to meetings after times are fixed. Meetings are firstly placed onto tiles and then the tiles are timetabled. Resources are then allocated to meetings. Later research conducted by Kingston (2008) investigates the use of a bipartite matching model, namely, global tixel matching, to assign resources such as teachers and rooms to meetings. These algorithms have been applied to Australian high schools.

3.8 Hybrid Approaches

Alvarez-Valdes et al.(1996) takes a three phase approach to the school timetabling problem. In the first phase a parallel heuristic algorithm with priority rules is used to create an initial timetable which is not usually feasible. Phase 2 applies a variation of the standard tabu search to the initial timetable created in phase 1 to produce a feasible timetable. Phase 3 improves the quality of the feasible timetable developed in phase 2. A graph theory approach, using the Floyd-Warshall algorithm, is taken in this phase. This approach was tested on randomly generated problems and data sets from 14 Spanish schools.

De Haan et al. (2007) take a four-phase approach to solving the STP. A preprocessing phase is conducted to cluster events into clusters schemes using a branch-and-bound algorithm. The second and third phases focus on constructing feasible timetables. The second phase assigns

lessons to day-parts using a dynamic priority rule. The cluster with the lowest availability is scheduled first. If this leads to unscheduled lessons the heuristic value is recalculated. During the third phase day-parts are allocated to timeslots. A graph coloring first-fit heuristic is used for this. The fourth phase uses a Tabu search to improve the feasible timetable. The system was successfully applied to a data set from a Netherlands high school.

The method employed by Schaerf (1991) firstly constructs an initial timetable by randomly assigning teacher-class pairs according to the requirements matrix. The RNA (Randomized Non-Ascendant) search is then applied to improve the initial timetable until no further improvement is possible. At this point the tabu search is applied until there is no more improvement. During the RNA phase the hard constraint cost is weighed higher than the soft constraint cost. During the tabu search this weight is "adjusted dynamically". This is referred to as adaptive relaxation. Adaptive relaxation was found to be essential for finding feasible solutions. The RNA and tabu phases are repeated sequentially until there is no further improvement in the quality of the timetable. This hybrid system was used to solve the STP for a randomly generated data set and data sets obtained from two Italian schools.

3.9 Comparative Studies

Colorni et al. (1998) compare the performance of simulated annealing, tabu search with local search and genetic algorithms in solving the school timetabling problem for two Italian high schools. The GA uses reproduction and crossover and applies mutation iteratively. Mutation swaps a set of contiguous genes in the same row. Day mutation swaps two days in the same row. A filtering algorithm is used to convert an infeasible offspring to a feasible one. Instead of creating timetables from scratch the previous year's handmade solution was used as a starting point. Tabu search produced the best results followed by genetic algorithms and simulated annealing.

Smith et al. (2003) use a Hopfield neural network to solve the school timetabling problem. The neural network is used to solve the problem for nine highly constrained school timetabling problems made available by Abramson (Beasley 2010). The performance of the Hopfield neural network on this data set is compared to that of greedy search, simulated annealing and tabu search. The neural network performed better than the other methods, followed by simulated annealing.

Wilke et al. (2008) compare the performance of tabu search, simulated annealing, genetic algorithms and branch and bound in solving the school timetabling problem for a German high school. The comparison is performed with respect to computation time and solution quality. Tabu search had the best runtimes but was unable to find feasible solutions. Simulated annealing found feasible solutions. The GA was not able to find feasible solutions. Branch and bound had the highest runtimes and also did not produce valid solutions.

4. Benchmark Data Sets for the STP

For further advancement in school timetabling research, it is essential that there is a variety of school timetabling data sets publicly available to test and compare the performance of different

methodologies in solving the STP. This section describes school timetabling data sets that are publicly available.

Five of the data sets used in the study conducted by Smith et al. (2003) are available from the OR-Library maintained by Beasley(2010) at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/tableinfo.html.

These problems have been described as "hard" timetabling problems and are more highly constrained than real-world school timetabling problems. The cost of the timetable is the number of class, teacher and venue clashes.

The seven data sets from Greek high schools tested by Beligiannis et al. (2008) are available at http://prlab.ceid.upatras.gr/timetabling/ . Each data set contains a requirements matrix specifying how many times each teacher must meet each class. In addition to this each data set also includes co-teaching and sub-classing (i.e. splitting and merging of classes) requirements.

Post et al. (2008) have initiated a project to facilitate the easy exchange of benchmark school timetabling data sets and so promote research in this domain. Post et al. propose an XML format to describe school timetabling problems and have setup a website for the submission of problems, namely, http://wwwhome.math.utwente.nl/~postgf/BenchmarkSchoolTimetabling/. There are 19 data sets from 7 different countries, namely, Australia (3), Brazil (6), England (1), Finland (4), Italy (1), Greece (1) and the Netherlands (3), available from the website.

5. Future Research Directions for School Timetabling

It is evident from the previous section that a variety of school timetabling data sets are now available. This will facilitate a comparison of different methodologies in solving the school timetabling problem and promote further development of the school timetabling domain.

A majority of the methods described in this paper firstly implement a construction phase during which a heuristic is used to sort class-teacher (or class-teacher-room) tuples in order of difficulty to schedule and allocate each tuple in sequence. An area which has not been investigated as thoroughly as in university examination timetabling (Carter et al. 1996) is different heuristics that can be used to estimate the difficulty of scheduling a tuple. The heuristic commonly used in the domain of school timetabling is a largest degree heuristic which gives priority to the most constrained class-teacher tuples (Raghavjee et al. 2008; Valouxis et al. 2003). The derivation and evaluation of other heuristics for this domain needs to be examined.

The aim of hyper-heuristics is to generalize well over the problems in a particular domain, rather than producing the best result for one or more problem sets (Burke et al. 2003). Hyper-heuristics search a heuristic space rather than a solution space. The heuristic space is usually comprised of combinations of low-level heuristics which can be constructive or perturbative (Pillay et al. 2009). While there has been research into the effectiveness of hyper-heuristics for university course and examination timetabling (Burke et al. 2007), this has not been studied for school timetabling.

A way of stimulating research in a particular field is to arrange competitions for the particular domain. This is evident from the 2nd International Timetabling Competition (McCollum et al. 2009) which has promoted research in examination, post enrollment and curriculum based university course timetabling. Arranging a track for school timetabling in future competitions will help develop the field more rapidly.

Building school timetabling systems that can be deployed in schools and are not just research tools is important to the development of the field. Such a system must allow for timetable reconstruction without much effort. The user must be able to easily make minor changes to the constraints, change the weighting of constraints, make manual changes and request a new timetable taking these into consideration. When methodologies are evaluated in this domain, the evaluation must also take into consideration reconstruction ability.

The application of some search techniques, e.g. evolutionary algorithms, can be time consuming. There have been earlier studies investigating the use of parallel processing to decrease the runtime of timetabling systems (Abramson 1991; Abramson et al. 1991). Given the emergence of multi-core processors the effectiveness of parallel processing in improving runtimes of school timetabling systems needs to be examined.

6. Conclusion

Research in the domain of school timetabling has not advanced as rapidly as other spheres of educational timetabling. This has been attributed to most studies in this domain being done in isolation of each other and the lack of a variety of benchmark problems to perform comparative studies on. The definition of the school timetabling problem varies drastically from one study to the next. This paper has attempted to provide a standardized definition of the problem in terms of problem requirements, hard constraints and soft constraints. The paper provides an overview of methodologies employed to solve the school timetabling problem. In addition to this the paper provides details of publicly available school timetabling data sets. Finally, the paper describes future directions of research in this field, namely, the derivation of new heuristics, an evaluation of hyper-heuristics in this domain, arranging competitions for school timetabling, developing usable systems that promote timetable reconstruction and the use of parallel processing to improve the runtimes of school timetabling systems.

7. References

1. Abramson D (1991) Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms. Management Science, Vol. 37, No. 1, 98-113.
2. Abramson D, Abela J (1991) A Parallel Genetic Algorithm for the Solving the School Timetabling Problem. In proceedings of the Fifteenth Australian Conference: Division of Information Technology, C.S.I.R.O, 1-11.
3. Alvarez-Valdes R, Martin G, Tamarit JM (1996) Constructing Good Solutions for the Spanish School Timetabling Problem, in the Journal of the Operational Research Society, Vol. 47, No. 10, 1203-1215.

4. Beasley JF (2010) OR Library, http://people.brunel.ac.uk/mastjjb/jeb/orlib/tableinfo.html, last accessed 1 February 2010.
5. Beligiannis GN, Moschopoulos CN, Kaperonis GP, Likothanassis SD (2008) Applying Evolutionary Computation to the School Timetabling Problem: The Greek Case. Computers and Operations Research, Vol. 35, 1265-1280, Elsevier.
6. Bello GS, Rangel MC, Boeres MCS (2008) An Approach for the Class /Teacher Timetabling Problem. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008), http://w1.cirrelt.ca/~patat2008/ PATAT_7__PROCEEDINGS/Papers/Boeres-WA2b.pdf. Last accessed 05/02/10.
7. Birbas T, Daskalaki S, Housos E (1997) Timetabling for Greek High Schools. Journal of the Operational Research Society, Vol. 48, No. 2, 1191-1200, December 1997.
8. Birbas T, Daskalaki S, Housos E (2009) School Timetabling for Quality Student and Teacher Schedules. Journal of Scheduling, Vol. 12, Issue 2, 177-197, April 2009, Kluwer Academic Publishers.
9. Burke E, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003) Hyper-Heuristics: An Emerging Direction in Modern Research. Handbook of Metaheuristics, chapter 16. Kluwer Academic Publishers, 457– 474.
10. Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A Graph-Based Hyper-Heuristic for Educational Timetabling Problems. European Journal of Operational Research (EJOR), 176, 177 – 192.
11. Calderia JP, Ross AC (1997) School Timetabling Using Genetic Search. In the proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT '97), 115-122.
12. Carter MW, Laporte G, Lee SY (1996) Examination Timetabling: Algorithmic Strategies and Applications. The Journal of the Operational Research Society, 47(3), 373-383.
13. Carter MW, Laporte G (1997) Recent Developments in the Practical Course Timetabling. In Burke E, Carter M (eds.) Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science, Vol. 1408, 3-19, Springer.
14. Colorni A, Dorigo M, Maniezzo V (1998) Metaheuristics for High School Timetabling. Computational Optimization and Applications, Vol. 9, 275-298, Kluwer Academic Publishers.
15. de Haan P, Landman R, Post G, Ruizenaar H (2007) A Four-Phase Approach to a Timetabling Problem for Secondary Schools. In Burke EK, Rudova H (eds.) The Practice and Theory of Automated Timetabling VI, Lecture Notes in Computer Science, Vol. 3867, 267-279, Springer-Verlag.
16. Filho GR, Lorena LAN (2001) A Constructive Evolutionary Approach to School Timetabling. In Proceedings of the EvoWorkshops on Applications of Evolutionary Computing, Lecture Notes in Computer Science, Vol. 2037, 130-139, Spring-Verlag.

17. Grobner M, Wilke P, Buttcher S (2003) A Standard Framework for Timetabling Problems. In Burke EK, De Causmaecker P (eds.): Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), LNCS 2740, 24-38, Springer-Verlag Berlin Heidelberg.
18. Jacobsen F, Bortfeldt A, Gehring H (2006) Timetabling at German Secondary Schools: Tabu Search versus Constraint Programming. In Burke EK, Rudova H (Eds.): proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), 439-442, ISBN 80-210-3726-1.
19. Kingston JH (2004) A Tiling Algorithm for High School Timetabling. In Practice and Theory of Automated Timetabling V, Lecture Notes in Computer Science, Vol. 3616/2005, 208-225, Springer Berlin/Heidelberg.
20. Kingston JH (2006) The KTS High School Timetabling Systems. In Burke EK, Rudova H (Eds.): proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), 181-195, ISBN 80-210-3726-1.
21. Kingston JH (2008) Resource Assignment in High School Timetabling. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008), http://w1.cirrelt.ca/~patat2008/PATAT7 PROCEEDINGS/Papers/Kingston-WA2b.pdf. Last accessed 05/02/10.
22. Kwok L, Kong S, Kam Y (1997) Timetabling in Hong Kong Secondary Schools. Computer Education, Vol. 28, No. 3, 173-183, Elsevier Science Ltd.
23. Marte M (2006) Towards Constraint-Based Grammar School Timetabling. http://www3.deis.unibo.it/Events/Deis/Workshops/PapersCPAIOR99/21final.ps, Last accessed 12 February 2010.
24. McCollum B, McMullan P, Paechter B, Lewis R, Schaerf A, Di Gaspero L, Parkes AJ, Qu R, Burke EK (2009a) Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS Journal on Computing, doi: 101287/ijoc.1090.0320.
25. Melicio F, Calderia JP, Rosa A (2006) THOR: A Tool for School Timetabling. In Burke EK, Rudova H (eds.): Proceedings of the 6th International Conference on the Practice and Teaching of Automated Timetabling (PATAT 2006), 532-535, ISBN 80-210-3726-1.
26. Moura AV, Scaraficci RA (2010) A GRASP Strategy for a More Constrained School Timetabling Problem. International Journal of Operational Research, Vol. 7, No. 2,152-170, 2010.
27. Nurmi K, Kyngas J (2008) A Conversion Scheme for Turning a Curriculum-Based Timetabling Problem into a School Timetabling Problem. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, http://w1.cirrelt.ca/~patat2008/PATAT_7_PROCEEDINGS/Papers/Nurmi-TC1.pdf, last accessed 12 February 2010.
28. Pillay N, Banzhaf W (2009) A Study of Heuristic Combinations for Hyper-Heuristic Systems for the Uncapacitated Examination Timetabling Problem. European Journal of Operational Research(EJOR), 197, 482-491.

29. Post G, Ahmadi S, Daskalaki S, Kingston JH, Kyngas J, Nurmi C, Ranson D, Ruizenaar H (2008) An XML Format for Benchmarks in High School Timetabling. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, http://w1.cirrelt.ca/~patat2008/ PATAT_7_ PROCEEDINGS/Papers/Post-WD2a.pdf, last accessed 12 February 2010.
30. Raghavjee R, Pillay N (2008) An Application of Genetic Algorithms to the School Timetabling Problem. In Cilliers C, Barnard L, Botha RA (Eds.) Proceedings of SAICSIT 2008, 193-199, ACM Press.
31. Santos HG, Ochi LS, Souza MJF (2005) A Tabu Search Heuristic with Efficient Diversification Strategies for the Class/Teacher Timetabling Problem. Journal of Experimental Algorithms, Vol. 10, 2.9, ACM.
32. Santos HG, Uchoa E, Ochi LS, Maculan N (2008) Strong Bounds with Cut and Column Generation for Class-Teacher Timetabling. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, http://w1.cirrelt.ca/~patat2008/ PATAT_7_ PROCEEDINGS/Papers/Santos-WD2b.pdf, last accessed 12 February 2010.
33. Schaerf A (1991) Tabu Search Techniques for Large High-School Timetabling Problems. Technical Report CS-R9611 1996, Computer Science/Department of Interactive Systems, Centrum Voor Wiskunder en Informatica (CWI), ISSN 0169-118X.
34. Smith KA, Abramson D, Duke D (2003) Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparative Results, in Computers and Industrial Engineering, Vol. 44, 283-305, Pergamon.
35. Valouxis C, Housos E (2003) Constraint Programming Approach for School Timetabling. In Computers and Operations Research, Vol. 30, 1555-1572, Pergamon.
36. Wilke P, Grobner M, Oster N (2002) A Hybrid Genetic Algorithm for School Timetabling. AI 2002: Advances in Artificial Intelligence, Lecture Notes in Computer Science, Vol. 2557/2002, 455-464, Springer Berlin.
37. Wilke P, Ostler J (2008) Solving the School Timetabling Problem Using Tabu Search, Simulated Annealing, Genetic and Branch & Bound Algorithms. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, http://w1.cirrelt.ca/~patat2008/PATAT_7_PROCEEDINGS/ Papers/Wilke-WD2c.pdf, last accessed 12 February 2010.
38. Wright M (1996) School Timetabling Using Heuristic Search. Journal of the Operational Research Society, Vol. 47, No. 3, 347-357.
39. Wood J, Whitaker D (1998) Student Central School Timetabling. Journal of the Operational Research Society, Vol. 49, No. 11, 1146-1152.

Evolving Hyper-Heuristics for a Highly Constrained Examination Timetabling Problem

Nelishia Pillay

School of Computer Science, University of KwaZulu-Natal, South Africa
+27 2605644
+27 2605648
pillayn32@ukzn.ac.za

Abstract: A lot of research has been conducted on hyper-heuristics for examination timetabling. However, most of this work has been focused on an uncapacitated version of the problem. This study reports on evolving hyper-heuristics for a highly constrained version of the problem, namely, the set of problems from the second International Timetabling Competition (ITC '07). Previous work has shown that using an evolutionary algorithm (EA) based hyper-heuristic with more than one chromosome representation is more effective than the standard EA using a single representation. This study evaluates an EA hyper-heuristic, using three different chromosome representations, in solving the capacitated examination timetabling problem. The results produced by the hyper-heuristic were found to be comparable to other methodologies applied to the same problem set.

Keywords: hyper-heuristics, examination timetabling, evolutionary algorithms

1. Introduction

The main aim behind hyper-heuristics is to generalize well in a particular domain rather than producing the best result for one or more problems in that domain (Burke et al. 2003 ; Ross et al. 2005). Hyper-heuristics select or combine either perturbative or constructive low-level heuristics. The study presented in this paper focuses on the combination of constructive heuristics. There have been numerous studies investigating the use of constructive hyper-heuristics in the examination timetabling domain. An overview of the most relevant studies follows.

Qu et al. (2005) apply variable neighborhood search to a space of combinations of two or more constructive low-level heuristics. Burke et al. (2005; 2007) employ a tabu search to explore the space of heuristics combinations. Qu et al. (2009b) analyze the heuristic combinations, found by a tabu search hyper-heuristic, that produce feasible timetables in order to identify patterns of low-level heuristics that lead to good quality solutions. Qu et al. (2009a) compare the performance of different local search strategies in exploring the heuristic space. Iterated local search produced the best results. The study also revealed that searching the solution space whilst constructing the timetable using the heuristic combination output by exploring the heuristic space, produces better

quality timetables. A Greedy Adaptive Search Procedure (GRASP) is used by Burke et al. (2009) to search a space of heuristic combinations of two constructive low-level heuristics. The quality of the feasible timetable constructed using the heuristic combination returned by GRASP is further improved using steepest descent. Asmuni et al. (2005; 2007; 2009) combine two or three constructive low-level heuristics using a fuzzy logic function. This function estimates the difficulty of scheduling an examination. Examinations are sorted according to their difficulty and scheduled in sequence. Pillay et al. (2007) implement a genetic programming system to search a space of constructive heuristic combinations. The length of the combinations in the initial population is randomly chosen to be between two and a preset maximum. Tournament selection is used to choose parents, to which the crossover and mutation operators are applied to create the next generation. The studies described thus far have combined heuristics linearly and applied them sequentially. Pillay et al. (2009) achieve good results with combining constructive low-level heuristics hierarchically using logical operators and applying them simultaneously. Four heuristic combinations are created and tested. This work is extended further by Pillay (2009) by employing genetic programming to search a space of such heuristic combinations.

All these studies have used the Carter benchmark set of timetabling problems (Carter et al. 1996) to test the hyper-heuristics. This set of benchmarks is comprised of 13 real-world problems. The hard constraint for this set of problems is that no students must be scheduled to sit two examinations at the same time and the soft constraint aims to spread the examinations for each student. A more recent set of examination timetabling problems has been made available by the organizers of the second International Timetabling Competition (ITC ’07). This set of eight problems is highly constrained and is representative of the current real-world examination timetabling problem. At the time of writing this paper, studies into applying hyper-heuristics to such a highly-constrained, multi-objective examination timetabling problem as that represented by the ITC ’07 problem set had not as yet been conducted or published.

The main contribution of the study presented in this paper is the evaluation of the performance of an evolutionary algorithm hyper-heuristic on the set of highly-constrained capacitated examination timetabling problems. The study presented in this paper employs an evolutionary algorithm (EA) to search the heuristic space of linear combinations of constructive low-level heuristics. In previous work three different representations, namely, fixed length, variable length, and *n*-times representation were evaluated for the Carter benchmark problems. A separate EA run using each of the representations as well as an EA combining all three representations were implemented. The study revealed that the EA combining all three representations performed better than the EAs using each of the representations separately. Thus, the EA in this study combines the three chromosome representations. Note that the aim of the study is not to compare the three representations but test the effect of an EA combining the three representations on a more highly constrained, capacitated version of the examination timetabling problem.

The following section provides an overview of the examination timetable problem as defined for the second International Timetabling Competition. Section 3 presents the EA-based hyper-heuristic. The experimental setup for testing the EA-HH is described in section 4.

The performance of the EA-HH on the eight problems is discussed in section 5. The outcome of this study and future extensions of this work are summarized in section 6.

2. The Examination Timetabling Problem for "ITC'07"

The examination timetabling problem requires the allocation of examinations to timeslots so that the hard constraints of the problem are satisfied and the soft constraint cost is minimized. A timetable is said to be feasible if it meets all the hard constraints of the problem. The hard and soft constraints differ drastically from one examination timetabling problem to the next. The ITC '07 problem set has the following hard constraints:

- All examinations must be scheduled.
- There are no clashes, i.e. a student is not scheduled to sit two examinations during the same period.
- The duration of the period that each examination is assigned to is not less than the duration required for the examination.
- The number of students writing an examination does not exceed the capacity of the room the examination is assigned to.
- Period related hard constraints must be met. There are three such constraints: some examinations must occur after other examinations; certain examinations must be written during the same period while others must not be scheduled in the same period.
- Room related hard constraints must be satisfied. In some cases an examination must be assigned exclusively to a room.

The soft constraints for the ITC '07 problem set are summarized below:

- Two in a row – The number of examinations taken back to back by students is minimized.
- Two in a day – The number of examinations written in the same day by students is minimized.
- Period spread – The number of examinations written within a specified period, e.g. 5 days, is minimized.
- Mixed durations – Examinations are of different durations. The number of examinations with different durations in the same room for a period is minimized.
- Larger examinations scheduled earlier in the examination timetable – The number of examinations with a "larger" number of students scheduled in the latter part of the timetable is minimized.
- Room penalties – Certain rooms have a penalty associated with using them. The number of times rooms with penalties are utilized is minimized.
- Period penalties – Certain periods also have a penalty associated with their use. The number of times these periods are used is minimized.

A more detailed description of these soft constraints can be found in (McCollum, 2007). The winner of the competition has taken a multi-phased approach to the problem (Muller, 2008). An iterative forward search, using conflict-based statistics to prevent cycling, is firstly applied to find a feasible timetable. The second phase employs hill-climbing to further improve the quality of the feasible timetable. If hill-climbing can no longer improve the solution, a variation of the Great Deluge algorithm is applied for further improvement.

Gogos et al. (2008), who were placed second, use a combination of the Greedy Randomized Adaptive Search Procedure (GRASP), simulated annealing and mathematical programming to solve the examination timetabling problem.

A variation of GRASP incorporating tabu search is firstly used to find a feasible solution. This solution is then improved using simulated annealing. In the last phase integer programming with branch and bound is used to further improve the quality of the timetable.

Atsuta et al. (McCollum et al. 2009b) were placed third in the competition and implement a constraint satisfaction problem solver which uses tabu search and local iterated search in solving the examination timetabling problem.

De Smet (2007) combines the use of the drools-solver and tabu search to solve the problem. This approach was placed fourth in the competition.

Pillay (2007) takes a developmental approach (DA) to the examination timetabling problem. The DA mimics the processes of cell biology. Each organism developed represents a timetable with each cell representative of an examination period. The creation of an organism begins with a single cell which is developed into a fully grown organism by means of cell division, cell interaction and cell migration. The fully grown organism then goes through a process of maturation in which cell migration is used to further improve the quality of the timetable. The DA was placed fifth in the competition.

The organizers of the competition take a two-phased approach to the problem (McCollum et al. 2009a; McCollum et al. 2009b). The first phase is a construction phase which uses an adaptive ordering heuristic to create a feasible solution. The feasible solution is improved using an extension of the Great Deluge algorithm.

The results obtained by these methods are presented in section 5.

3. The Evolutionary Algorithm

The EA employs the generational control model (Koza 1992) and the population size remains fixed from one generation to the next. An initial population is created and iteratively improved via the processes of evaluation, selection and recreation. These processes are described in the sections below.

3.1 Initial Population Creation

Each element of the population is a string containing two or more characters representing the following constructive low-level heuristics:

- Largest degree (*l*) – Examinations involved in the largest number of clashes are scheduled first.
- Largest enrolment (*e*) – Examinations with the largest number of students are given priority.
- Largest weighted degree (*w*) – Examinations with the largest number of students involved in clashes are allocated first.
- Saturation degree (*s*) – Examinations with the least number of feasible options available on the timetable developed thus far are given priority.
- Spread heuristic (*h*) – Is an estimate of the spread of examinations over a range of periods for each student. The estimate is defined in terms of the proximity of the examinations for a student to each other, and weighted by the number of students involved. Thus, examinations with a higher value are given priority. Like the saturation degree, this heuristic is not static and its value depends on the current state of the timetable. Thus, it needs to be recalculated whenever an allocation is made to the timetable.

These low-level heuristics are combined using one of the following representations:

- Fixed length heuristic combination (FHC) – The length of the combination is equal to the number of examinations, e.g. *well* if the number of examinations is four. One heuristic is used to schedule each examination.
- Variable length heuristic combination (VHC) – Studies conducted by Cowling et al. (2002) and Han et al. (2003) applying a hyper-heuristic genetic algorithm to the trainer scheduling problem have revealed that a chromosome representation with variable length produces better results than a fixed length representation as the GA is able to evolve a chromosome of the optimal length. A similar representation is used in this study. The length of each combination is randomly chosen to be between two and a specified maximum, e.g. *lessh*. Each heuristic is used to schedule an examination.

  If the length of the combination is less than the number of examinations the combination is wrapped around beginning at the start of the string again. If the combination is longer than the number of examinations, only a substring of the combination is applied. Thus, two combinations of length larger than the number of examinations would essentially be clones of each other. Due to this together with the fact that mutation and crossover may produce clones, the reproduction operator is not used.
- N-times heuristic combination (NHC) – Each combination is composed of integers and characters representing low-level heuristics, e.g. *3h2l3s1w1s*. The integer preceding the heuristic specifies the number of examinations the heuristic will be used to schedule. In the example the first three examinations will be allocated according to the spread heuristic, the next two with the largest degree heuristic and so on. The sum of the integer values in the combination is equal to the number of examinations to be scheduled. The reason for including this representation is that it may result in the algorithm converging quicker to certain areas of the heuristic space. For example, it may take longer to evolve

the combination *lesllllllllhh* than it would take to evolve *1l1e1s8l2h*. In this way more of the heuristic space may be explored in a shorter time.

The size of the initial population is a genetic parameter and differs for each problem domain. The population consists of an equal number of combinations of each type of representation. Previous work has shown that in the domain of EA-based hyper-heuristics for examination timetabling different representations are suitable for different problems. Thus, an EA providing more than one chromosome representation in the initial population is more effective. The EA converges to the most suitable representation.

3.2 Evaluation and Selection

Each heuristic combination is assigned a fitness measure. The fitness measure of a combination is a function of the hard and soft constraint cost of the timetable constructed using the combination. During the timetable construction process each examination is allocated to the feasible minimum cost timeslot. If there is more than one option the period is randomly chosen from the possible options. If a feasible period is not available, a period is randomly selected. If there is more than one room available, the room with the best fit is chosen. If there is more than room with the same best fit value, the lowest penalty is used to decide which room to use. The fitness measure is the soft constraint cost multiplied by the hard constraint cost incremented by one. Based on trial runs performed, this fitness function proved to be representative of the fitness of an individual without any processing overheads. The fitness measure is used by the selection method to choose parents of the next generation. The tournament selection method is used in this study.

A tournament of *t* individuals is randomly chosen. The fittest individual in the tournament is returned as a winner and is used as a parent for the next generation. The value of *t* is a genetic parameter and is problem dependant.

3.3 Recreation

Two genetic operators, namely, mutation and crossover are implemented to create the next generation. The mutation operator randomly changes a low-level heuristic in a copy of the selected parent. The tournament selection method is evoked to choose a parent. For example, if *welsh* is a chosen parent, the offspring could be *weesh*. In this case *l* was chosen to be replaced. The heuristic *e* was randomly chosen to replace *l*. There is no limit set on the size of the offspring.

The crossover operator randomly chooses two crossover points in two parents selected using tournament selection, and swaps the fragments at the crossover points to produce two offspring. For example suppose w***e******h***l*l* and *l***e***h* are the chosen parents and three is the crossover point in the first parent and two in the second. The resulting offspring are w*eeh* and *lhll*. Trial runs have indicated that returning the fitter offspring is more effective than returning both offspring. Crossover occurs between parents of the same representation, i.e. both parents must be NHC or VHC. Crossover is also permitted between VHC and FHC parents. Crossover is structure-preserving in the case of NHC ensuring that an integer value precedes each heuristic.

4. Experimental Setup

The EA-HH was tested on the eight problems from the examination timetabling track of ITC '07. The four hidden data sets are not publicly available. The characteristics of the eight data sets as presented by McCollum et al. (2009a) are listed in Table 1.

Table 1: Characteristics of the Problem Set

Problem	Conflict Density (%)	No. of Exams	No. of Students	No. of Periods	No. of Rooms
Exam_1	5.05	607	7891	54	7
Exam_2	1.17	870	12743	40	49
Exam_3	2.62	934	16439	36	48
Exam_4	15.0	270	5045	21	1
Exam_5	0.87	1018	9253	42	3
Exam_6	6.16	242	7909	16	8
Exam_7	1.93	1096	14676	80	15
Exam_8	4.55	598	7718	80	8

The genetic parameters used by the EA are tabulated in Table 2. These values were obtained by performing test runs. Due to the stochastic nature of evolutionary algorithms, ten runs were performed for each problem set, each with a different random number generator seed. The EA was implemented in Java and simulations were run on a system with a 1995 Mhz Intel Core 2 Duo processor and 2 gigabytes of memory.

Table 2: Genetic Parameters

Parameter	Value
Number of generations	100
Population size	500
Maximum initial length	5
Tournament size	10
Crossover rate	0.3
Mutation rate	0.7

5. Results and Discussion

The EA-HH produced feasible timetables for all eight problem sets. The best soft constraint cost obtained over ten runs for each problem is listed in Table 3. Although the main aim of a hyper-heuristic is to generalize well rather than producing the best result, for completeness Table 3 compares the performance of the EA-HH to other methodologies applied to the same set of problems. These methodologies are described in section 2. Note that while these methodologies perform one or more improvement phases to reduce the soft constraint cost of the timetable, the EA-HH does not perform additional optimization once a feasible solution is found. Furthermore, in this study the time limitation imposed by the competition was not adhered to and the time taken by the EA-HH was not monitored. It is assumed that the EA-HH will have longer runtimes due to the overhead of solving the problem using each heuristic combination in each population.

Table 3: A Comparison of Results

Problem	EA-HH	Muller	Gogos	Atusta et al.	De Smet	Pillay	McCollum et al.
Exam_1	8559	4370	5905	8006	6670	12035	4633
Exam_2	830	400	1008	3470	623	2886	405
Exam_3	11576	10049	13771	17669	-	15917	9064
Exam_4	21901	18141	18674	22559	-	23582	15663
Exam_5	3969	2988	4138	4714	3848	6860	3042
Exam_6	28340	26585	27640	29155	27815	33005	25880
Exam_7	8167	4213	6572	10506	5436	17666	4037
Exam_8	12658	7742	10521	14317	-	15592	7461

Although the EA-HH does not further optimize feasible solutions, its performance is comparable to the other methodologies applied to this problem set. For all problem sets the EA-HH has produced better results than at least one to three other methodologies. Table 4 lists the representation, i.e. FHC, VHC or NHC, of the best heuristic combination evolved for each problem over the ten runs. Note that the representation converged to for each run maybe different and that the population at the end of the run will have a majority of the individuals with the same structure, because the EA has converged to a particular area of the heuristic space, but not all the individuals will necessarily have the same representation.

Table 4. Representations Converged to

Problem	Representation
Exam_1	FHC
Exam_2	VHC
Exam_3	VHC
Exam_4	VHC
Exam_5	FHC
Exam_6	NHC
Exam_7	FHC
Exam_8	NHC

The algorithm converged to a combination with the NHC representation for two of the problem sets, with the FHC representation for three of the problem sets and with the VHC representation for three of the problem sets. An analysis into a possible correlation between the representation converged to and the characteristics of each problem will be conducted as part of future work.

6. Conclusion and Future Work

This paper presents an EA-based hyper-heuristic for a highly constrained examination timetabling problem, namely, that used for the examination timetabling track of the second International Timetabling Competition. The EA combines three different chromosome representations. The EA-HH produced feasible timetables for all eight competition timetabling problems. Furthermore, the quality of the timetables produced by the EA-HH was comparable to and in some cases better than the best timetable produced by other methodologies, even through the EA-HH did not perform additional optimization after a feasible solution was obtained. However, the time needed to find an optimal heuristic combination was not monitored and thus the

approach may have the advantage of longer runtimes. It is interesting to note that the representation of the heuristic combination producing the best quality timetable differed for each problem. Future work will investigate whether there is a correlation between the representation of the best heuristic combination and the characteristics of the problem.

7. References

1. Asmuni H, Burke EK, Garibaldi JM (2005) Fuzzy Multiple Ordering Criteria for Examination Timetabling. In Burke EK, Trick M (eds.), selected Papers from the 5th International Conference on the Theory and Practice of Automated Timetabling (PATAT 2004)- The Theory and Practice of Automated Timetabling V, Lecture Notes in Computer Science, 3616, 147–160.
2. Asmuni H, Burke EK, Garibaldi JM, McCollum B (2007) Determining Rules in Fuzzy Multiple Heuristic Orderings for Constructing Examination Timetables. In proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications Conference (MISTA 2007), 59-66.
3. Asmuni H, Burke EK, Garibaldi JM, McCollum B, Parkes AJ (2009) An Investigation of Fuzzy Multiple Heuristic Orderings in the Construction of University Examination Timetables. Computers and Operations Research, Elsevier, 36(4), 981-1001.
4. Burke E, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003) Hyper-Heuristics: An Emerging Direction in Modern Research. Handbook of Metaheuristics, chapter 16. Kluwer Academic Publishers, 457– 474.
5. Burke EK , Dror M , Petrovic S, Qu R (2005) Hybrid Graph Heuristics with a Hyper-Heuristic Approach to Exam Timetabling Problems. In Golden BL, Raghavan S, Wasil E.A. (eds.), The Next Wave in Computing, Optimization, and Decision Technologies – Conference Volume of the 9th Informs Computing Society Conference, 79 -91.
6. Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A Graph-Based Hyper-Heuristic for Educational Timetabling Problems. European Journal of Operational Research (EJOR), 176, 177 – 192.
7. Burke EK, Qu R, Soghier A (2009) Adaptive Selection of Heuristics within a GRASP for Examination Timetabling Problems. In proceedings of Multidisciplinary International Conference on Scheduling 2009 (MISTA 2009), 409-422.
8. Carter MW, Laporte G, Lee SY (1996) Examination Timetabling: Algorithmic Strategies and Applications. The Journal of the Operational Research Society, 47(3), 373-383.
9. Cowling P, Kendall G, Han L (2002) An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem. In proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL '02), 267-271.
10. DeSmet G (2007) ITC-2007- Examination Track Drools-solver. http://www.cd.qub.ac.uk/winner/bestexamsolutions/Geoffrey_De_smet_examination_description.pdf, last accessed 12 February 2010.

11. Gogos C, Alefragis P, Housoss E (2008) A Multi-Stage Algorithmic Process for the Solution of the Examination Timetabling Problem. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008),Montreal,http://w1.cirrelt.ca/~patat2008/PATAT_7_PROCEEDINGS/Papers/Gogos-HC2b.pdf, last accessed 12 February 2010.
12. Han L, Kendall G (2003) Guided Operators for a Hyper-Heuristic for Genetic Algorithms. In proceedings of AI-2003 Advances in Artificial Intelligence-The 16th Australian Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence, 2903, 807-820.
13. Koza JR (1992) Genetic Programming I: On the Programming of Computers by Means of Natural Selection, MIT Press.
14. McCollum B (2007) International Timetabling Competition: Evaluation Function. http://www.cs.qub.ac.uk/itc2007/examtrack/, last accessed 12 February 2010.
15. McCollum B, McMullan P, Paechter B, Lewis R, Schaerf A, Di Gaspero L, Parkes AJ, Qu R, Burke EK (2009a) Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS Journal on Computing, doi: 101287/ijoc.1090.0320.
16. McCollum B, McMullan PJ, Parkes AJ, Burke EK, Abdullah S (2009b) An Extended Great Deluge Approach to the Examination Timetabling Problem. In proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Application (MISTA 2009), 424-434.
17. Muller T (2008) ITC2007 Solver Description: A Hybrid Approach. In the proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT'08), http://w1.cirrelt.ca/~patat2008/PATAT_7_PROCEEDINGS/Papers/Muller-HC1c.pdf, last accessed 12 February 2010.
18. Pillay N, Banzhaf W (2007) A Genetic Programming Approach to the Generation of Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem. In Neves et al. (eds.), Progress in Artificial Intelligence, Lecture Notes in Artificial Intelligence, 4874, 223-234.
19. Pillay N (2008) A Developmental Approach to the Examination Timetabling Problem. http://www.cd.qub.ac.uk/winner/bestexamsolutions/pillay.pdf, last accessed 12 February 2010.
20. Pillay N (2009) Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem. In proceedings of Multidisciplinary International Conference on Scheduling 2009 (MISTA 2009), 409-422.
21. Pillay N, Banzhaf W (2009) A Study of Heuristic Combinations for Hyper-Heuristic Systems for the Uncapacitated Examination Timetabling Problem. European Journal of Operational Research(EJOR), 197, 482-491.
22. Qu R, Burke EK (2005) Hybrid Neighbourhood HyperHeuristics for Exam Timetabling Problems. In Proceedings of the MIC2005: The Sixth Metaheuristics International Conference, Vienna, Austria.

23. Qu, R, Burke EK (2009a) Hybridizations within a Graph Based Hyper-Heuristic Framework for University Timetabling Problems. Journal of Operational Research Society (JORS), 60, 1273-1285.
24. Qu R, Burke EK, McCollum B (2009b) Adaptive Automated Construction of Hybrid Heuristics for Exam Timetabling and Graph Colouring Problems. European Journal of Operational Research (EJOR), 198(2), 392-404.
25. Ross P (2005) Hyper-heuristics. In Burke E.K., Kendall G. (eds): Search Methodologies: Introductory Tutorials in Optimization and Decision Support Methodologies, chapter 17. Kluwer, 529 -556.

An XML Format for Benchmarks in High School Timetabling II

Gerhard Post · Jeffrey H. Kingston · Samad Ahmadi · Sophia Daskalaki · Christos Gogos · Jari Kyngas · Cimmo Nurmi · Haroldo Santos · Ben Rorije · Andrea Schaerf

Abstract We present the progress on the benchmarking project for high school timetabling that was introduced at PATAT 2008. In particular, we announce the High School Timetabling Archive HSTT2010 with 15 instances from 7 countries and an evaluator capable of checking the syntax of instances and evaluating the solutions.
Keywords. Timetabling, high school, benchmark, XML, scheduling.

1 Introduction

"It is surprising that no standard format for exchanging datasets in the field of high school timetabling has emerged until now." This sentence was the motivation for a

Gerhard Post
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands and
ORTEC, Groningenweg 6k, 2803 PV Gouda, The Netherlands

Jeffrey H. Kingston
School of Information Technologies, The University of Sydney, Australia

Samad Ahmadi
Dept. of Informatics, De Montfort University, The Gateway, LE1 9BH, UK Leicester

Sophia Daskalaki
Engineering Sciences Department, University of Patras, 26500 Rio Patras, Greece

Christos Gogos
Engineering Sciences Department, University of Patras, 26500 Rio Patras, Greece

Jari Kyngas
Satakunta University of Applied Sciences, Tiedepuisto 3, 28600 Pori, Finland

Cimmo Nurmi
Satakunta University of Applied Sciences, Tiedepuisto 3, 28600 Pori, Finland

Haroldo Santos
Computing Department, Universidade Federal de Ouro Preto, Brazil

Ben Rorije
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Andrea Schaerf
DIEGM, University of Udine, via delle Scienze 206, 33100 Udine, Italy

group of researchers to define a format capable of expressing high school timetabling instances from all over the world, see (Post 2010).

The high school organization is different around the world, consequently the problems in high school timetabling that arise from real cases in various countries differ as well. As a specific example, one of the main differences that has emerged from our research is related to allowing idle times for students during school hours versus the cases where this is not allowed. In the first case teachers are usually not preassigned to the lessons, as this may lead to infeasibilities. In the second case, teachers are mostly preassigned, leading to the problem of eliminating idle times for students and minimizing them for teachers.

Another important difference is related to the granularity of the scheduling process: sometimes it is performed at the level of an entire class, whereas in other cases of a single student. In the latter case the problem usually becomes harder, since the schedule of each individual student has to be evaluated during the solving process, thus making the process computationally more expensive.

The purpose of this paper is to report on the progress of this collaborative research and reflect on the current situation; we will give a short overview and motivation of the XML format in Section 2, discuss the current archive in Section 3, the evaluator in Section 4, and give an outlook to the future in Section 5.

2 The format

Differences in the organization of high schools in different countries imply that the definition of a unified format for high school timetabling is not a trivial task. The current format has emerged after many iterations; indeed the format discussed in (Post 2010) differs considerably from the original version presented at PATAT 2008.

The format of our benchmark is mapped out by an XML schema which defines the compulsory and optional elements that need to be present in the XML files holding the instances and solutions. The basic elements of an instance are the *times*, *resources*, and *events*, complemented by the *constraints*, which are imposed on them. We believe that the structure of the format as it is now will essentially remain the same over time. The reason lies in the principal choice to embed the "business logic" in the constraints, and not in the basic elements. At present, the format contains (only) 15 constraint types, including "obvious" ones, like *AssignTimeConstraint* (assign a start time to selected events) and *AvoidClashesConstraint* (a resource may be involved with at most one event at a time). The modular nature of the schema assures that new constraints can be added without having to change its structure. Indeed, we believe that the set of constraints will probably be extended further to incorporate new specialized constraints to deal with unforeseen problems in other countries.

A fragment of an instance file is shown in Figure 1. All objects have the attribute *Id* for referencing, and the child *Name* for displaying. *Times*, *Resources* and *Events* can be grouped in *TimeGroups*, *ResourceGroups* and *EventGroups*, respectively. For demonstration purposes the *Times* section is expanded with some more detail.

One of the main discussions during the design of this format was about the "domain specific" structure and the "solver needed" structure. The "domain specific" structure reflects how a timetabler at a school has structured the data; for example, a timetabler will distinguish days of the week, will think in terms of students, school classes or teachers (certainly not of general resources), and will consider courses and subjects

```
<Instance Id="Example">
  <Times>
    <TimeGroups>
      <Day Id="Day1"/> <Name>Monday</Name> </Day>
      ...
      <Day Id="Day5"/> <Name>Friday</Name> </Day>
      <TimeGroup Id="AllTimes"/> <Name>AllTimes</Name> </TimeGroup>
      ...
    </TimeGroups>
    <Time Id="Day1_1">  <Name>Monday 1</Name>
      <TimeGroups>
        <Day Reference="Day1"/>
        <TimeGroup Reference="AllTimes"/>
      </TimeGroups>
    </Time>
    ...
  </Times>
  <Resources>
     ...
  </Resources>
  <Events>
     ...
  </Events>
  <Constraints>
     ...
  </Constraints>
</Instance>
```

Fig. 1 A problem instance in the XML format

as principal objects for scheduling. The solver instead requires a structure organized in terms of variables which represent units of lessons or resources, while conceptual entities are not important.

In the format, although some of the domain specific structural elements (for example "days" and "courses") are supported and their use is recommended, it is not obligatory to use them. We mention the two most important ones. A *Day* is introduced as a time group, that almost certainly will be needed, because in high school timetabling there are many constraints at a daily level. For example, constraints about "working hours", "idle times", and "number of days present". In addition, by introducing days we are able to display daily schedules for the resources. Another element is the *Course*, which is introduced as an *EventGroup* for a subject and a student group combination. This is important in order to control the spreading of the individual lessons (events) of a course over the week days. *Courses* also allow control on events with similar properties; if certain events are identical, they can be clustered to one event and allow the lessons to be sub-events. Moreover, constraints like the *SplitEventsConstraint* can prescribe how such an event should be split into sub-events.

Our view on inclusion of new constraints has changed during the past two years. Originally we tried to include all the constraints that we encountered in the literature, or that we could imagine to be useful. On the contrary, the current set of constraints in the format reflect only the constraints needed by the contributors, and no more. The reason for this is that complicated constraints usually need to be clarified by an expert.

3 The archive HSTT2010

At the website (Post 2008) the archive HSTT2010 with 15 instances from 7 different countries is available. These instances have appeared previously in the literature, but were not available for download. Apart from the instances, solutions are also available. We intend to keep record of the best found solutions for researchers to be able to validate their solvers. In Table 1 we present the instances that have been contributed. In the columns are given

- the country (Country);
- name of the instance (Name);
- total duration of all events (EvD);
- the number of teachers (T);
- the number of school classes (SC);
- the number of students (St);
- the number of rooms (Ro).

Country	Name	EvD	T	SC	St	Ro
Australia	BGHS98	1564	56	30		45
	SAHS6	1876	43	20		36
	TES99	806	37	13		26
Brazil	Instance 1	75	8	3		
Brazil	Instance 4	300	23	12		
England	St Paul	1227	68	67		67
Finland	Artificial	200	22	13		12
	College	854	46	31		33
	High school	319	18	102		13
	Secondary school	306	25	14		25
Greece	High School 1	372	29	66		
Italy	Instance 1	133	13	3		
Netherlands	GEPRO	2675	132	44	846	80
	Kottenpark 2003	1203	75	18	453	41
	Kottenpark 2005	1272	78	26	498	42

Table 1 The instances in the archive HSTT2010.

Note that the instances vary significantly in size. Most instances are described at the level of school classes, which might split further to form sub groups. The Dutch instances,however, carry information at the level of individual students as well. For the lower grades the groups of students (school classes) are fixed and all students of a group attend most lessons together. Conversely, for the higher grades the timetable of each student is mostly personal, since the compulsory lessons constitute only $\frac{1}{3}$ of the lessons. For the Australian instances, the teachers have to be assigned as well in the timetabling process and in such case split assignments should be avoided.

4 The evaluator

Apart from the instances, the format also models solutions. A solution is presented by describing the duration of all events (as mentioned previously it is possible to define

a "course" event of duration 3, which can be split into, for example, three sub-events of duration 1), the time slots assigned to each event, and (in some cases) the resources assigned to events. Once a solution is provided we can then evaluate it. The evaluation leads to two integers: the infeasibility value (i.e. the total cost of the hard constraints violations) and the objective value (the total cost of the soft constraints). The total cost is generated from two different constraint types: if the constraint is "hard", then the cost is added to the infeasibility value, otherwise to the objective value. Depending on the type of the constraint, the cost is attributed to: an event (for example: "is there a time assigned to the event?"), to an event group (for example: "is the course well-spread?") or to a resource (for example: "are the idle times within the given limits?"). The cost of the schedule is the sum of all separate costs. The cost value V is then calculated from the deviation D of the constraint C with weight λ by the formula:

$$V = \lambda \cdot f_C(D)$$

Here f_C is a cost function and its type is specified in the constraint. The cost functions supported currently are: step function, linear function, and quadratic function.

The format supports multiple instances and multiple groups of solutions to these instances:

```
<HighSchoolTimetableArchive>
   <Instances>
      <Instance Id="Instance1">
      ...
      </Instance>
   </Instances>
   <SolutionGroups>
      <SolutionGroup>
         <Solution Reference="Instance1">
            ...
         </Solution>
      </SolutionGroup>
   </SolutionGroups>
</HighSchoolTimetableArchive>
```

In benchmarking an indisputable interpretation of the data and constraints is essential. The documentation for the current constraints is given on the website devoted to this project (Post 2008). Although we believe that this effort so far was very important, we still thought it was not enough. For this reason an evaluator was additionally developed and can be accessed from the Internet, see (Kingston 2009). The task of the evaluator is three-fold: first it checks if the provided instances and solutions satisfy the syntax rules. This includes checking consistency of the used Ids and whether a solution respects the *preassignments*. The second task is to provide the infeasibility value and the objective value of the solution and, if indicated, a full report on the deviations for all constraints. Finally, if several solutions of the same instance are included, a comparison table is presented. The first two parts are very useful for the implementation as they provide the user with checks on the generated format and implementation of the constraints. In this sense the evaluator is the ultimate documentation: either the result of the evaluator is accepted, or the behaviour of the evaluator is marked as "bug".

Some cases led to discussions of the interpretation. One example is the constraint *LimitBusyTimesConstraint*, which limits the busy times of a resource on a day between a minimum and a maximum. Initially this constraint generated the cost for the days without work too. In the revised implementation these days are skipped. This is

reasonable, since by using another constraint one can describe the number of days a resource should be busy.

5 The future

After the past and the present let us make some speculations about the future, based on our experiences till now. First of all we have noted that there is a great interest in this format; interest from researchers in high school timetabling, but also from other areas of timetabling. In our opinion this shows that many researchers feel the urge for exchangeable datasets. Our vision in building this data format is our belief that efforts to define very general formats right from the start have a great chance to fail. We believe that the current format keeps a good balance between tangibility and abstraction.

Though several researchers have expressed their interest, converting their formats to the proposed format requires time. In view of this we are proud that we can present an archive with 15 datasets from 7 contributors. By active acquisition and support we hope to extend this in the near future. As an example we can refer to the development of a automated repository for High School timetabling. This repository will have facilities to convert data sets to the standard data format, uploading new data sets, download of existing data and use of the evaluator. A work in progress version of this site is available (Ahmadi 2010).

If new contributors appear, new constraints or variants of the current constraints may appear. One type of constraint needed is a sequencing constraint. Looking at a resource the sequence of events can be important. An example is when the events take place at different locations: in such cases one would like to minimize the number of location changes, or have breaks or idle times in between. Another example could be a sequence of the subjects like Mathematics, Physics, Chemistry, and Biology for a group of students.

Though new constraints will be needed, one should keep in mind that an instance is usually just an approximation of practice. The timetabler at school will have a clear view of the schedules, but to formalize this in constraints is not always easy (or interesting). In practice hard constraints can turn out to be soft, if necessary, while giving weights to the soft constraints can be difficult. The violation of some important soft constraints can turn out to be unacceptable to the timetabler. The timetabler will rather change the data, and try again to find a good solution. On one hand this might be discouraging for the researchers, but on the other hand it is always a challenge to cope with the reality.

References

[(Ahmadi 2010)] Samad Ahmadi and Ben Rorije, 'High School Timetabling Problem Repository', `http://opt-kd.cse.dmu.ac.uk/www/`, (2010).

[(Kingston 2009)] , Jeffrey H. Kingston, 'The HSEval High School Timetable Evaluator', `http://www.it.usyd.edu.au/~jeff/hseval.cgi`, (2009).

[(Post 2008)] Gerhard Post, 'High school timetabling web site', `http://wwwhome.math.utwente.nl/~postgf/BenchmarkSchoolTimetabling/`, (2008).

[(Post 2010)] Gerhard Post et al, 'An XML Format for Benchmarks in High School Timetabling', Annals of Operations Research, (2010) [DOI 10.1007/s10479-010-0699-9].

A Construction Approach for Examination Timetabling based on Adaptive Decomposition and Ordering

Syariza Abdul Rahman[1], Edmund Burke[1], Andrzej Bargiela[1], Barry McCollum[2] and Ender Özcan[1]

[1]*University of Nottingham, School of Computer Science, Jubilee Campus, Nottingham NG8 1BB, UK*
{sax, ekb, abb, exo}@cs.nott.ac.uk

[2] *Queen's University Belfast, School of Electronics, Electrical Engineering and Computer Science, University Road Belfast, BT7 1NN, Northern Ireland, UK*
b.mccollum@qub.ac.uk

In this study, we investigate an adaptive decomposition strategy that automatically divides examinations into *difficult* and *easy* sets for constructing an examination timetable. The examinations in the difficult set are considered to be hard to place and hence are listed before the ones in the easy set. Moreover, the examinations within each set are ordered using different strategies based on graph colouring heuristics. Initially, the examinations are placed into the easy set. During the construction process, the examinations that cannot be scheduled are identified as the ones causing infeasibility and are moved forward in the difficult set to ensure earlier assignment than the others for the subsequent attempts. On the other hand, the examinations that can be scheduled remain in the easy set. Within the easy set, a new subset called the boundary set is introduced to accommodate shuffling strategies to change the given ordering of examinations. The proposed approach which incorporates different ordering and shuffling strategies is explored on the Carter benchmark problems. The empirical results show that its performance is promising and comparable to existing constructive approaches.

Keywords: timetabling, decomposition, graph colouring, heuristic, grouping.

1 Introduction

Timetabling attracts numerous researchers and practitioners due to its challenging nature. Timetabling problems are NP hard real-world problems (Even et al. 1976) that are hard to solve and often require considerable amount of either human or computational time or both. There are many types of timetabling problems e.g. educational timetabling, nurse rostering, etc. The focus of this study is the university examination timetabling problem. Principally, the examination timetabling problem is concerned with the scheduling of a list of examinations into a restricted number of time-slots while satisfying a defined set of constraints. Hard constraints must be satisfied in creating a feasible solution e.g. no student should take two examinations at the same time. Soft constraints on the other hand can be broken but it is desirable to satisfy them as much as possible. The evaluation of the degree these soft constraints are satisfied provides an indication of the overall quality of a given solution. In relation to examination timetabling, evaluating the average cost of student spread in the timetable as an indicator of how 'good' a given solution is was introduced by Carter et al. (1996). More overview information on the examination timetabling problem and associated constraints can be found in (Carter and Laporte 1996; Carter et al. 1996; Petrovic and Burke 2004; Qu et al. 2009).

Considering that the only constraint dealt with is the requirement that no student should sit two examinations at the same time, the formulation of the examination timetabling problem is closely similar to other graph colouring problems. Ülker et al. (2007) discusses a grouping representation for this type of examination timetabling problems. The vertices and edges of a graph denote the examinations and the conflicting examinations that should not be scheduled at the same time, respectively, where the colour of a vertex denotes a time-slot in the timetable. The heuristic ordering methods for graph colouring are considered constructive approaches. These approaches have been used to find an initial solution before getting further to the improvement phase. There are several heuristic ordering methods commonly used in examination timetabling i.e. largest degree, saturation degree, largest weighted degree, largest enrolment and colour degree (Carter 1986; Carter and Laporte 1996, Burke et al. 2004a).

A wide variety of approaches have been applied to examination timetabling. The approaches vary from exact methods, such as, constraint logic programming

and constraint satisfaction to meta-heuristic approaches, such as, tabu search, simulated annealing and population based approaches, such as, evolutionary algorithms. Recent applications of search methodologies, such as hyper-heuristics that perform search over the heuristics space (Burke et al. 2003; Özcan et al. 2008) and case-based reasoning approaches to timetabling aim to work at a higher level of generality than meta-heuristics. An overview of methodologies employed for examination timetabling is provided in Table 1.

Recent studies in timetabling have focused on the constructive approaches for obtaining high quality solutions. Graph colouring heuristics have been customized with the adaptive approaches to order the examinations based on their difficulty of timetabling (Burke and Newall 2004). We have utilised the framework of 'squeaky wheel optimisation' (Joslin and Clement 1999), where the difficulty of scheduling an examination is identified based on its feasibility versus infeasibility in a previous iteration. In this work, the difficulty indicator of scheduling an examination was subsequently increased based on a certain parameter to enable it be scheduled earlier in the next iteration. In 2009, Abdul Rahman et al. extended this study by introducing more strategies for choosing an examination to be scheduled and the time-slots. In another adaptive approach, Casey and Thompson (2003) developed a GRASP algorithm for solving the examination timetabling problems. In their approach, the next examination to be scheduled is chosen from the top items in the list (called candidate list) using roulette wheel selection and then assigned to the first available slot.

Table 1. Some representative methodologies for solving examination timetabling problems

Methodology	**Reference(s)**
Cluster-based/decomposition	Balakrishnan et al. (1992), Burke and Newall (1999), Qu and Burke (2007)
Tabu search	Di Gaspero and Schaerf (2001), White and Xie (2001)
Simulated annealing	Thompson and Downsland (1998), Merlot et al. (2003)
Great deluge algorithm	Burke et al. (2004b)
Variable neighbourhood search	Burke et al. (2010)
Large neighbourhood search	Abdullah et al. (2007)
Iterated local search	Caramia et al. (2001)

GRASP	Casey and Thompson (2003)
Genetic algorithms	Burke et al. (1995), Ülker et al. (2007)
Memetic algorithms	Burke and Newall (1999), Ozcan and Ersoy (2005), Ersoy et al. (2007)
Ant algorithms	Eley (2007)
Exact method	Boizumault et al. (1996), David (1998), Merlot et al. (2003)
Multi-objective	Petrovic and Bykov (2003), Ülker et al. (2007)
Hyper-heuristic	Bilgin et al. (2007), Ersoy et al. (2007), Pillay and Banzhaf (2009)
Case-based reasoning	Burke et al. (2006)
Fuzzy approaches	Asmuni et al. (2009)
Neural network	Corr et al. (2006)
Constructive approaches	Burke and Newall (2004), Abdul Rahman et al. (2009)

The study by Qu and Burke (2007) describes an adaptive decomposition approach for constructing an examination timetable. This paper draws upon the research on similar adaptive approaches that make use of a decomposition strategy. We propose an approach which divides the problem into two sub-problems. We adopt the same naming convention introduced by Qu and Burke (2007) for these sets as *difficult* and *easy*. In this study, the problem is decomposed into difficult and easy sets at each iteration. A timetable is constructed based on the associated heuristic ordering for each set. We also introduce an additional set of examinations which is located in between the difficult and easy sets, which is referred to as the *boundary* set. This study describes several mechanisms associated with the boundary set in order to vary the search space of solutions. In Section 2, we present the details of our approach based on adaptive decomposition and ordering for examination timetabling. Section 3 describes the experimental data and discusses the results. Finally, the conclusion is provided in Section 4.

2 Automated Decomposition and Ordering of Examinations

Most of the timetabling approaches described do not make use of the information obtained from the process of building an infeasible timetable. The examinations causing the infeasibility of a solution provide an indication that those examinations are very difficult to place and should perhaps be treated in different ways. We propose a general constructive framework as presented in Pseudocode 1 for solving the examination timetabling problem based on the automated decomposition of a set of examinations into two sets i.e. difficult and easy.

Pseudocode 1: Improvement and construction of a timetable based on automated decomposition and ordering of examinations.

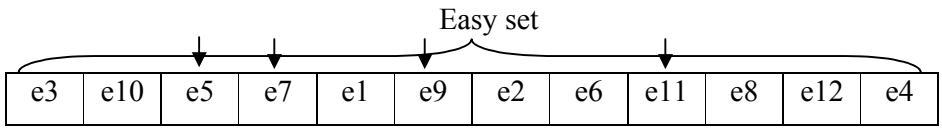
```
1.  E={e1, e2,…., eN}
2.  BoundarySetSize=δ
3.  EasySet=E; DifficultSet=ø; BoundarySet= ø; TempSet=ø
4.  Divide E into subsets
5.  FOR i=0 to MAXIter
6.    OrderExamsWithinSubsets(DifficultSet, EasySet)
7.    BoundarySet=CreateBoundarySet(DifficultSet, EasySet)
8.    WHILE (there are examinations to be scheduled)
9.      Consider changing the ordering of examinations
         using Shuffling-Strategy
10.     Employ Selection-Strategy to choose an
          unscheduled exam, e
11.     IF e can be scheduled THEN
12.        TempSet=TempSet ∪{e}
13.        Schedule e to the time-slot with the least penalty
             In the case of the availability of multiple
             time-slots with the same penalty,
             choose one randomly
14.     ELSE
15.        Move exam e to DifficultSet
16.     END-IF
17.     EasySet=TempSet
18.    END-WHILE
19.    Evaluate solution, store if it is the best found so far
20. END-FOR
```

During each iteration, a new solution is constructed from an ordered list of examinations. The difficult set consists of the examinations that cannot be placed into a time-slot within the timetable due to some conflicts with other examinations from the previous iteration. These examinations need to be associated with a large penalty imposed on the unplaced examinations. On the other hand, the examinations in the easy set cause no violations during the

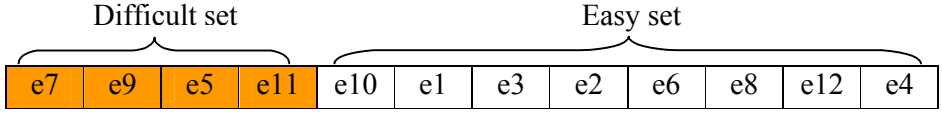
timetabling. In our approach, all the examinations that contribute to the infeasibility in a solution are given priority. They are moved forward in the ordered list of examinations and treated first. Such examinations are detected and included in the difficult set at each iteration and a predefined ordering strategy is employed before their successive assignment to the available timeslots. The remaining examinations that generate no feasibility issues are placed into the easy set and the original ordering of those examinations is maintained. In order to incorporate a stochastic component for the selection of examinations from the generated ordering, some shuffling strategies are utilised. The following subsections discuss these strategies.

2.1 Interaction between Difficult and Easy Sets through a Boundary Set

An adaptive decomposition approach is experimented with using two graph colouring heuristics for generating the initial ordering of examinations. We have tested the largest degree heuristic that orders the examinations decreasingly with respect to the number of conflicts with each examination and the saturation degree heuristic that dynamically orders the unscheduled examinations based on the number of available time-slots for each during the timetable construction. The reason for testing these two graph colouring heuristics is to compare their achievement in terms of solution quality and the contribution of difficult set size, as they represent static and dynamic ordering heuristics. Initially, all the examinations are considered to be a member of the easy set (as illustrated in Figure 1(a)).



(a)



(b)

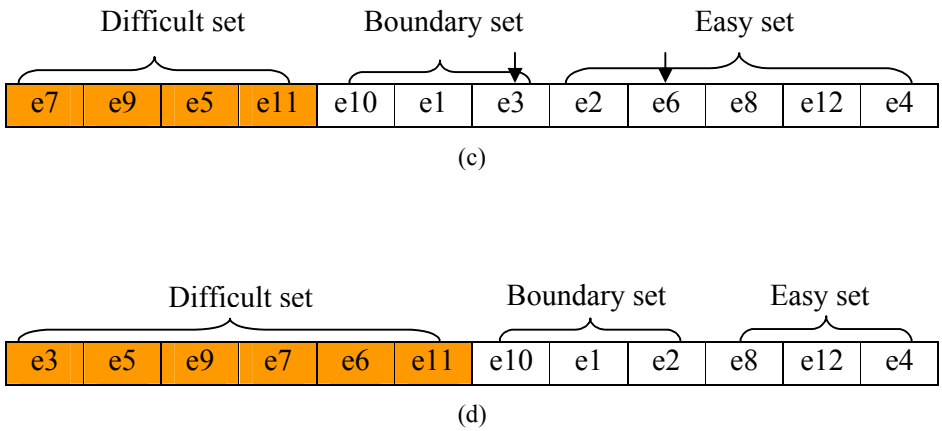


Figure 1. (a) All examinations are in a easy set in the first iteration and examinations that cause infeasibility are marked, (b) difficult and easy sets after an iteration resulting with an infeasible solution, (c) boundary set with a prefixed size is added to the difficult set after an iteration and reordering is performed, (d) the step in (a) is repeated and the infeasible examinations are placed in the difficult size, the size of difficult set increased.

During each iteration, the examinations causing infeasibility are identified. As in Figure 1(a), all such examinations are marked as a member of the difficult set to be moved forward towards the top of the list of examinations (Figure 1(b)), while the examinations that caused no violation during the assignment to a time-slot remain in the easy set. In Figure 1(c), the boundary set is created between the difficult and easy set and is merged with the difficult set before a reordering is performed to the difficult set. In the next iteration, more infeasible examinations are detected and included in the difficult set. Consequently, the size of the difficult set is increased from one iteration to another.

2.2 Swapping the Examinations Between Difficult and Boundary Sets

This strategy shuffles the difficult set and the boundary set by swapping the examinations in between them randomly. Occasionally, the examination causing infeasibility is not necessarily the one that is very difficult to schedule. The infeasibility may happen due to the previous assignment and ordering. This strategy introduces the opportunity for some of the examinations in the difficult set to be chosen later in the timetable. There is also a possibility that the examinations in the boundary set are swapped back to the original set because this process is done randomly. Figure 2 illustrates how the swapping of examinations between two sets might take place.

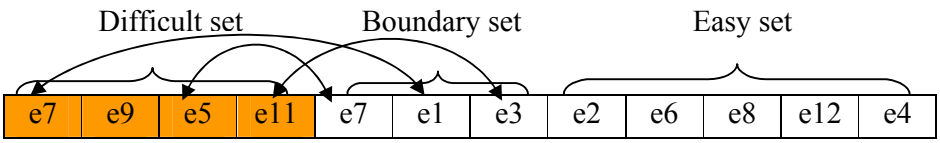


Figure 2: The boundary set is swapped with the difficult set and is reordered before assigning examinations to the time-slots.

2.3 Roulette Wheel Selection for Examinations

We utilised a roulette wheel selection strategy that incorporates a stochastic element in choosing examinations before assigning them to the time-slots. If there is no improvement evident for a certain time, a list of examination of size *n* was chosen from the ordered list in the difficult set from which and an examination is chosen based on a probability. The probabilities of an examination being chosen were calculated based on a score, s_i of each examination in the list of size *n*. The new size of the difficult set will be the set which includes the size of boundary whenever there is improvement to the solution quality. The score value, s_i is a dynamic measure that is obtained from the largest and saturation degree values (as in equation 1), where Num_clash_i is the number of examinations in conflict with the examination *i*, *Max_clash* is the maximum number of conflicts with all examinations, Sat_degree_i is the saturation degree value for the examination *i* and *Num_slots* is the number of time-slots given to the specified problem. *Sat_degree* value in this problem is initialised as 1.

$$s_i = \frac{Num\_clash_i}{Max\_clash} + \frac{Sat\_degree_i}{Num\_slots} \qquad (1)$$

The probability, p_i of an examination being chosen from *n* list of examinations is,

$$p_i = \frac{s_i}{\sum_{i=0}^{n-1} s_i}, \qquad (2)$$

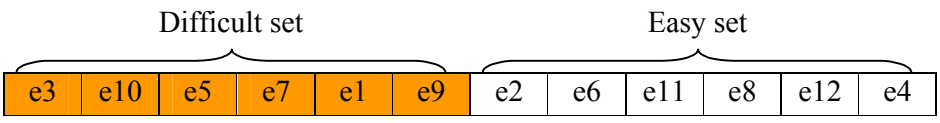
A random number from (0, 1) is obtained in order to choose an examination from a list of examination of size *n*. Those examinations with higher score values will have a greater chance of being chosen.

2.4 Comparison of Our Approach to a Previous Study

Qu and Burke (2007) previously proposed an adaptive decomposition approach to construct examination timetables. Their approach starts with an initial ordering of examinations using a graph colouring heuristic, namely saturation degree. In the approach, a perturbation is made by randomly swapping two examinations in order to obtain a *better* ordering. Examinations are then decomposed into two sets: difficult and easy.

The initial size of the difficult and easy sets are prefixed as half of the number of examinations in a given problem as shown in Figure 3(a). At each iteration, the size of the difficult set is modified according to the feasibility of the solution. If the solution is infeasible after the adjustment of the ordering of examinations then the first examination that causes infeasibility (e.g. e11) is moved forward for a fixed number of places (e.g., five as illustrated in Figure 3(b)). The size of the difficult set is then re-set to the point where the difficult examination is placed. Otherwise, if feasible solution or an improved solution is obtained, the size of the difficult set is increased (Figure 3(c)).

Our approach initialises with the easy set including all the examinations and the difficult set is formed during each construction phase at each iteration. The size of the difficult set depends on the number of unscheduled examinations that cannot be assigned to any time-slot from all previous iterations. The size of the difficult set never decreases and after a certain number of iteration, the number of examinations in the difficult set might be sustained. On the other hand, in the previous approach, the size of the difficult set is prefixed and increased when the feasible solution or improved solution is obtained statically. The set is also allowed to shrink. Additionally, the previously proposed approach uses an initial ordering and reorders all the examinations without using a heuristic, which is not the case in our approach. Although we have used the same approach for reordering the examinations in difficult and easy sets separately, examinations in different sets can be reordered based on a different heuristic at each iteration.



(a)

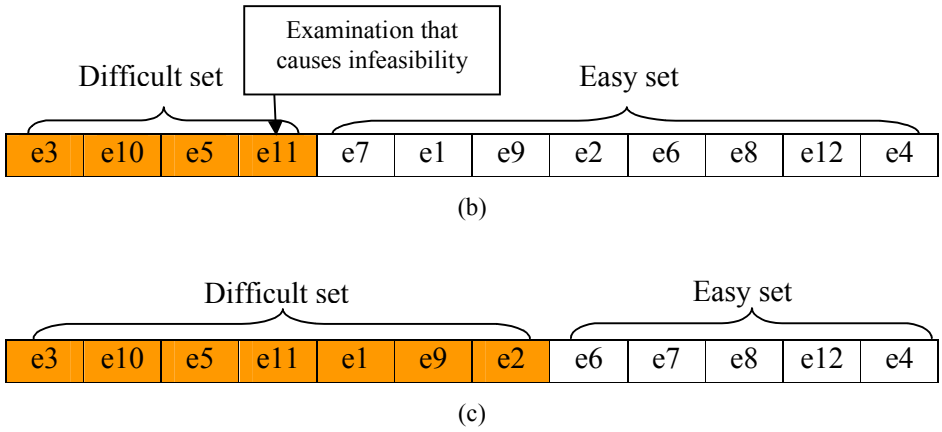


Figure 3. Difficult and easy sets (a) in the first iteration, (b) after an iteration is over (a) resulting with an infeasible solution, (c) after an iteration is over (a) resulting with a feasible solution.

3 Experiments

The experiments were tested on benchmark problems introduced by Carter et al. (1996) and are publicly available at ftp://ftp.mie.utoronto.ca/pub/carter/testprob/. In this study, we used version I of the 12 problems that were adapted from Qu et al. (2009) to differentiate various versions of the problem. During the experiments, five runs are performed and the stopping conditioned has been set as 10 000 iteration as to be equal with the experiment done by Qu and Burke (2007). Two types of heuristic ordering for initialisation are investigated: largest degree (LD) and saturation degree (SD). The difficult set is created using these two initial orders are then reordered with either largest degree or saturation degree. In this study, the same heuristic ordering is used for the examinations in the easy set. The heuristics used in a given approach will be denoted by a triplet as [*heuristic used for the initial ordering – heuristic used for ordering the examinations in the difficult set – heuristic used for ordering the examinations in the easy set*] from this point onwards. The size of the boundary set is fixed as 5.

Table 2 summarises the experimental results obtained applying the proposed approach to the benchmark problem instances. By looking at the best ordering for the difficult sets, we observe that the adding boundary set strategy performed better with largest degree initial ordering where nine out of the twelve problem instances has performed significantly better than saturation degree initial ordering while the swapping strategy has performed better with saturation degree initial ordering with seven out of twelve problem instances are better compare to largest

degree initial ordering. The best combination ordering for adding boundary set strategy is [LD-SD-LD] while the swapping boundary set strategy performed the best with [SD-SD-SD]. From the perspective of the strategies, it is clear that by swapping the boundary set with the difficult set produced better solution quality as compared to just combining the boundary set as a part of the difficult set. The swapping strategy has obtained seven better results while the combining strategy produced the better results for only five problem instances.

Table 2. Comparing solution quality for (a) [LD-LD-LD], (b) [SD-LD-SD], (c) [LD-SD-LD], (d) [SD-SD-SD] by adding boundary set into difficult set and swapping examinations between boundary and difficult sets with δ=5. (LD: largest degree; SD: saturation degree) (Bold font indicates the best for different ordering and strategy and italic is the best of all for each problem instance).

Problem	**Add the boundary set (δ=5) into the difficult set**				**Swap examinations in the boundary (δ=5) and difficult sets**			
	(a)	**(b)**	**(c)**	**(d)**	**(a)**	**(b)**	**(c)**	**(d)**
car91	5.72	5.60	5.77	**5.44**	5.71	5.37	5.75	***5.34***
car92	4.97	***4.76***	4.90	4.85	5.03	**4.87**	4.95	4.91
ear83 I	***41.36***	41.42	42.53	42.51	**41.90**	42.48	43.38	42.78
hec92 I	12.98	12.76	***12.24***	12.45	13.32	13.15	12.72	**12.52**
kfu93	16.68	16.57	**16.35**	16.40	***16.16***	16.61	16.38	16.49
lse91	13.44	12.96	**12.64**	12.85	13.45	***12.52***	12.93	12.95
rye93	11.13	10.79	***10.23***	10.31	11.35	10.66	10.53	**10.27**
sta83 I	163.93	162.12	**159.32**	159.74	161.98	159.34	159.08	***158.99***
tre92	9.77	9.72	**9.54**	9.69	9.81	9.50	9.66	***9.41***
ute92	30.68	30.08	**29.11**	**29.11**	30.21	29.79	29.34	***28.96***
uta92 I	3.92	**3.78**	3.96	3.87	3.96	***3.77***	3.89	3.82
yor83 I	45.85	46.97	***44.16***	44.75	45.84	46.28	**45.30**	45.39

In the next set of experiments, the effect of incorporating the roulette wheel into the examination selection process is tested with $n = 3$. As we can see from the results in Table 3, the adding boundary set strategy with roulette wheel selection has performed better by providing eight better solutions as compared to the swapping strategy with roulette wheel selection. From the results, the adding boundary set and selection strategy performed the best with combination of [LD-SD-LD] while the best combination ordering for swapping with selection strategy is [SD-LD-SD]. Comparing the best results obtained from the strategies without

roulette wheel selection in Table 2 and the strategies with roulette wheel selection in Table 3, it shows that when incorporating the selection strategy improves the performance of the approach.

Table 3. Comparing solution quality for (a) [LD-LD-LD], (b) [SD-LD-SD], (c) [LD-SD-LD], (d) [SD-SD-SD] with shuffling strategies of adding the boundary set into the difficult set and swapping examinations between the boundary and difficult sets with δ=5 and includes roulette wheel selection for examinations with *n*=3. (LD: largest degree; SD: saturation degree) (Bold font indicates the best for different ordering and strategy and italic is the best of all for each problem instance).

Problem	**Add the boundary set (δ=5) into the difficult set + Roulette wheel selection (*n*=3)**				**Swap examinations in the boundary (δ=5) and difficult sets + Roulette wheel selection (*n*=3)**			
	(a)	**(b)**	**(c)**	**(d)**	**(a)**	**(b)**	**(c)**	**(d)**
car91	5.67	***5.28***	5.64	5.43	5.67	5.57	5.77	**5.48**
car92	4.98	4.91	4.81	***4.76***	4.90	4.95	**4.86**	4.89
ear83 I	41.29	***40.60***	41.39	42.74	**41.67**	42.24	42.14	42.51
hec92 I	***12.09***	12.36	12.45	12.70	**12.20**	12.97	12.38	12.48
kfu93	16.25	16.22	***16.04***	16.43	16.43	16.07	16.20	**16.05**
lse91	12.70	***12.03***	12.76	12.67	13.06	12.75	**12.14**	12.82
rye93	10.52	**10.25**	10.40	10.41	10.45	10.45	***10.20***	10.54
sta83 I	160.20	162.26	**158.68**	160.00	158.60	160.43	***158.39***	161.75
tre92	9.31	9.72	***9.08***	9.55	9.43	9.79	**9.21**	9.63
ute92	**27.81**	27.93	28.57	27.90	28.01	27.84	***27.30***	27.55
uta92 I	3.95	***3.73***	3.83	3.81	3.88	3.92	3.89	**3.74**
yor83 I	45.48	45.24	45.76	**44.36**	545.04	***44.33***	44.51	44.81

Table 4 compares our best results obtained from the strategy of roulette wheel selection to the other previous results based on constructive approaches. Given by Qu and Burke (2007) is the closest comparison to our approach as they have also implemented a decomposition strategy. Comparing the solutions across all problem instances, it is observed that our approach does not yield the best results. However, it provides one better result when compared to the approach proposed by Qu and Burke (2007) for car91. Moreover, we have obtained better results than the approach by Asmuni et al. (2009) for four problems (car91, lse91, rye93 and ute92), Carter et al. (1996) for two problems (car91, car92),

respectively. However, Burke and Newall (2004) and Qu and Burke (2007) do not provide the result for rye93.

Table 4. Comparison of different constructive approaches (LD: largest degree; SD: saturation degree; RWS: roulette wheel selection) (The bold entries indicate the best results for constructive approaches only, while the italic ones indicate the best results for the decomposition approach).

Problem	**Burke & Newall (2004)**	**Carter et al. (1996)**	**Asmuni et al. (2009)**	**Abdul Rahman et al. (2009)**	**Qu & Burke (2007)**	**SD-LD-SD(RWS)**
car91	**4.97**	7.10	5.29	5.08	5.45	*5.28*
car92	**4.32**	6.20	4.54	4.38	*4.5*	4.91
ear83 I	36.16	36.40	37.02	38.44	*36.15*	40.60
hec92 I	11.61	**10.80**	11.78	11.61	*11.38*	12.36
kfu93	15.02	**14.00**	15.80	14.67	*14.74*	16.22
lse91	10.96	**10.50**	12.09	11.69	*10.85*	12.03
rye93	-	**7.30**	10.38	9.49	-	10.25
sta83 I	161.90	161.50	160.40	157.72	***157.21***	162.26
tre92	**8.38**	9.60	8.67	8.78	*8.79*	9.72
ute92	27.41	**25.80**	28.07	26.63	*26.68*	27.93
uta92 I	**3.36**	3.50	3.57	3.55	*3.55*	3.73
yor83 I	40.88	41.70	**39.80**	40.45	42.2	45.24

The overall results once again highlight the importance of the methodology used to change the ordering of difficult examinations, particularly the ones causing infeasibility. In our approach, the ordering of the examinations within the difficult set with respect to the others appears to be vital combined with the assignment strategy. As shown in Figure 4, for the experiments adding and swapping boundary set and difficult set without roulette wheel selection, the average number of the examinations in the difficult set varies with different ordering strategies. The approach using the largest degree ordering generates infeasibility more often for a given solution during the time-slot assignments as compared to the one using the saturation degree ordering. On the other hand, saturation degree ordering might easily create a feasible solution for some problem instances (e.g. car91 and uta92 I). However, using the saturation degree alone does not for guarantee a good solution quality.

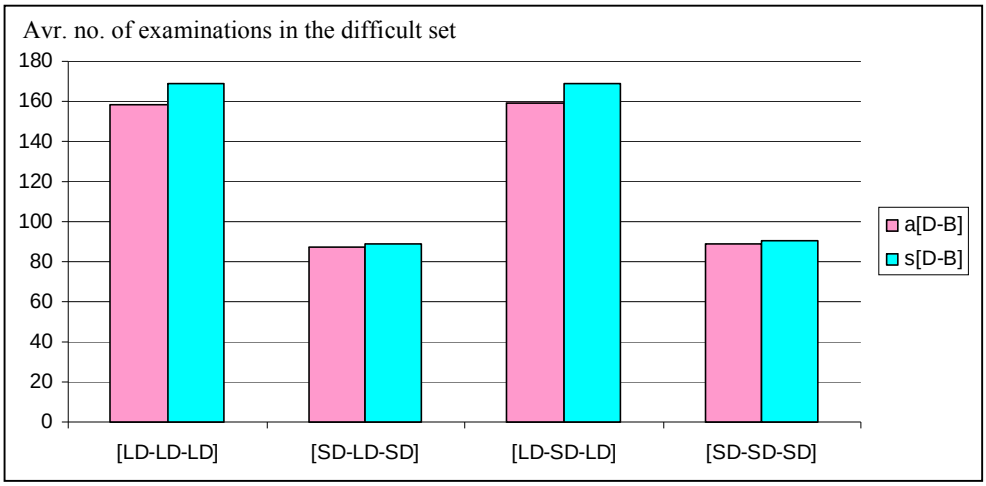
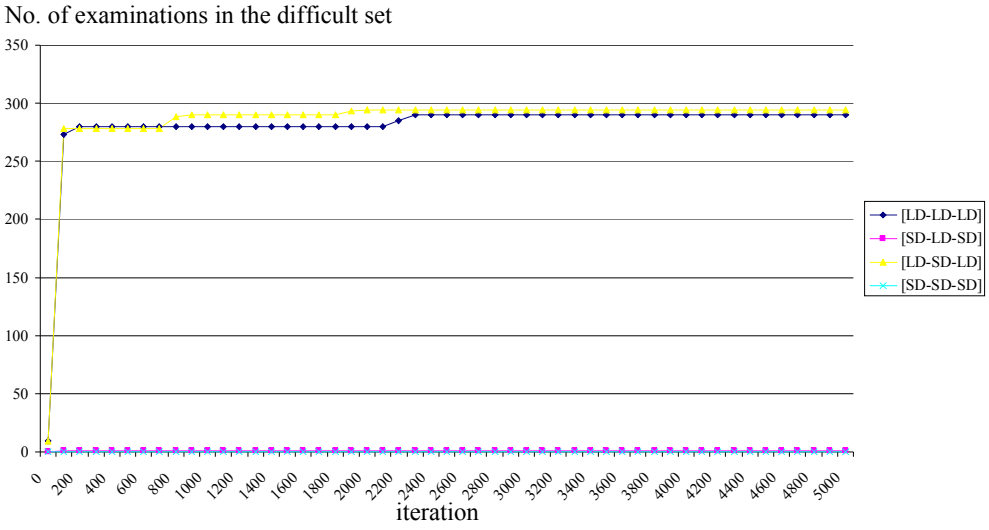


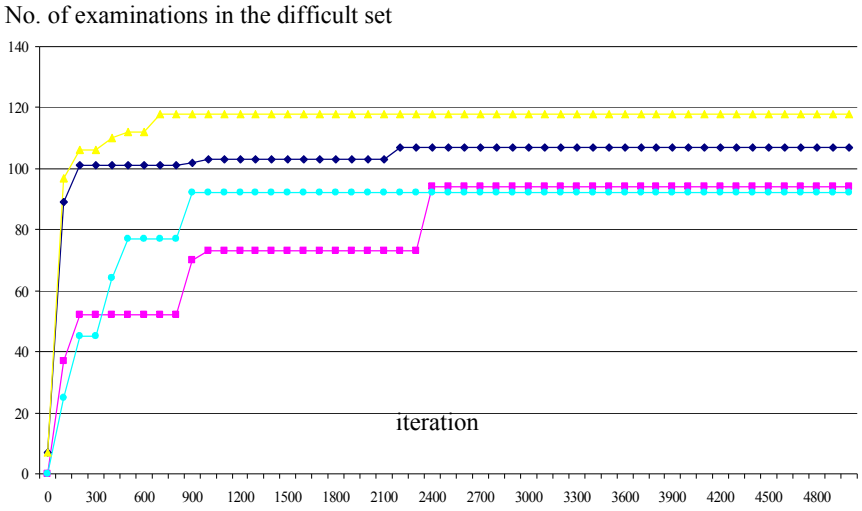
Figure 4. Average number of examinations in the difficult set (its size) over all problems considering all shuffling strategies using different initialisation and reordering heuristics. (LD: largest degree, SD: saturation degree, B: boundary set, D: difficult set, a: add, s: swap).

In some cases, using the saturation degree ordering may easily create a feasible solution when adding or swapping with the boundary set, the infeasible examinations can be obtained in this approach since this approach gives priority of ordering the difficult set. Consequently, adding or swapping the boundary set with the difficult set might have increased the number of examinations in the difficult set.

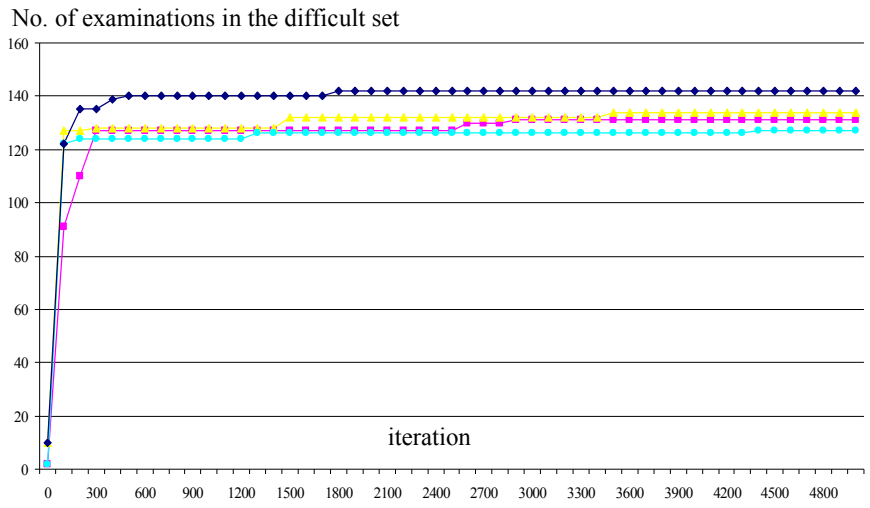
Figure 5((a), (b), (c)) illustrate the number of infeasible at each 100 iteration for different combination of initial ordering and reordering heuristics for the difficult set for car91, kfu93 and yor83 I, respectively. It shows that using largest degree causes increasing number of examinations to generate infeasible solution when compared to the saturation degree. car91 has an obvious difference in the number of infeasible examinations when comparing with the other two types of ordering i.e. [LD-LD-LD] and [LD-SD-LD]. In the other problem, kfu93 and yor83 I the number of infeasible examinations for different ordering is approximately the same but still using [SD-LD-SD] and [SD-SD-SD] are slightly advantageous. In all problems, the number of infeasible examinations is converged to a steady state after some point.



(a) car91



(b) kfu93



(c) yor83 I

Figure 5. The change in the size of the difficult set and the solution quality at every 100 iteration during the sample runs for (a) car91, (b) kfu93, (c) yor83 I. (LD: largest degree, SD: saturation degree).

4 Conclusion

This study discusses a novel approach based on adaptive strategies that decomposes the examinations in a given problem into two sets: a set of difficult to schedule and a set of easy to schedule examinations. This decomposition is performed automatically at each iteration, and is augmented with suitable ordering of examinations within each set. In this study, it is observed that by merging or swapping the boundary set with the difficult set could improve the solution quality. A stochastic component based on roulette wheel selection is embedded into the approach in order to shuffle the order of examinations. This mechanism gives a higher chance to an examination with a higher score to be selected for timetabling. It is observed that using saturation degree could decrease the possibility of creating infeasible solution and that dynamic ordering gives better ordering of examinations in the list. This preliminary study shows that the proposed approach is simple to implement, yet it is competitive to the other previous constructive approaches. In this study, the same ordering heuristics are used for reordering the examinations in the difficult and easy sets. In fact, the proposed framework allows the use of different strategies. As a future work,

different strategies will be investigated for reordering of examinations and choosing the examinations from the difficult set.

5 References

Abdul Rahman, S., Bargiela, A., Burke, E. K., McCollum, B., & Ozcan, E. (2009). Construction of examination timetables based on ordering heuristics. ISCIS 2009 (pp. 727-732) (In proceeding of 24th international symposium of computer and information sciences, Cyprus).

Abdullah, S., Ahmadi, S., Burke, E. K., & Dror, M. (2007). Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling. *OR Spectrum*, 29(2), 351-372.

Asmuni, H., Burke, E. K., Garibaldi, J. M., McCollum, B., & Parkes, A. J. (2009). An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers and Operations Research*, 36, 981-1001.

Balakrishnan, N., Lucena A., & Wong, R. T. (1992). Scheduling examinations to reduce second order conflicts. *Computers and Operations Research*, 19, 353-361.

Bilgin, B., Özcan, E., & Korkmaz, E. E. (2007). An experimental study on hyper-heuristics and exam scheduling, *Practice and theory of automated timetabling* 2006, Springer-Verlag, selected papers, *Lecture notes in computer science*, vol. 3867, 394-412.

Boizumault, P., Delon, Y., & Peridy, L. (1996). Constraint logic programming for examination timetabling. *Journal of Logic Programming*, 26(2), 217-233.

Burke, E. K., Petrovic, S., & Qu, R. (2006). Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9, 115-132.

Burke, E. K., & Newall, J. P. (2004). Solving examination timetabling problems through adaptation of heuristics orderings. *Annals of Operational Research*, 129, 107-134.

Burke, E. K., & Newall, J. P. (1999). A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1), 63-74.

Burke, E. K., Bykov, Y., Newall, J. P., & Petrovic, S. (2004b). A time-predefined local search approach to exam timetabling problem. *IIE Transactions*, 36(6), 509-528.

Burke, E. K., Eckersley, A., McCollum, B., Petrovic, S., & Qu, R. (2010). Hybrid Variable Neighbourhood Approaches to University Exam Timetabling. *European Journal of Operational Research*, to appear 2010.

Burke, E. K., Elliman, D. G., & Weare, R. (1995). A hybrid genetic algorithm for highly constrained timetabling problems. Paper presented at the 6th International Conference on Genetic Algorithms (ICGA'95), San Francisco, CA, USA, Pittsburgh, USA.

Burke, E. K., Kendall, G., Newall, J., Hart, E., Ross, P., & Schulenburg, S. (2003). Hyperheuristics: an emerging direction in modern search technology. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics*. 457-474. Netherlands: Kluwer.

Burke, E. K., Kingston, J., & de Werra, D. (2004a). Applications to timetabling. In J. Gross & J. Yellen (Eds.), *Handbook of graph theory* (pp. 445-474). Chapman Hall/CRC Press.

Caramia, M., DellOlmo, P., & Italiano, G. F. (2001). New algorithms for examination timetabling. In S. Naher & D. Wagner (Eds.), *Lecture notes in computer science. Algorithm engineering 4th international workshop, proceeding WAE 2000* (vol. 1982, pp. 230-241). Berlin: Springer.

Carter, M. W., Laporte, G., & Lee, S. (1996). Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47(3), 373-383.

Carter, M. W., & Laporte, G. (1996). Recent developments in practical examination timetabling. In E. K. Burke & P. Ross (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling I: selected papers from the 1st international conference* (vol. 1153, pp. 3-21). Berlin: Springer.

Carter, M. W. (1986). A survey of practical applications of examination timetabling algorithms. *Operational Research*, 34, 193-202.

Casey, S., & Thompson, J. (2003). GRASPing the examination scheduling problem. In E. K. Burke & P. De Causmaecker (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling IV: selected papers from the 4th international conference* (vol. 2740, pp. 234-244). Berlin: Springer.

Corr, P. H., McCollum, B., McGreevy, M. A., & McMullan, P. (2006). A new neural network based construction heuristic for the examination timetabling problem. In the international conference on parallel problem solving from nature (PPSN 2006) (pp. 392-401), Reykjavik, Iceland, September 2006.

David, P. (1998). A constraint-based approach for examination timetabling using local repair techniques. In E. K. Burke & M. W. Carter (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling II: Selected papers from the 2nd international conference* (vol. 1408, pp. 169-186). Berlin: Springer.

Di Gaspero, L., & Schaerf, A. (2001). Tabu search techniques for examination timetabling. In E. K. Burke & W. Erben (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling III: selected papers from the 3rd international conference* (vol. 2079, pp. 104-117). Berlin: Springer.

Eley, M. (2007). Ant algorithms for the exam timetabling problem. In E. K. Burke & H. Rudova (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling VI: selected papers from the 6th international conference* (vol. 3867, pp. 364–382). Berlin: Springer.

Ersoy, E., Özcan, E., & Uyar, Ş. (2007). Memetic algorithms and hyperhill-climbers. In P. Baptiste, G. Kendall, A. Munier-Kordon, & F. Sourd (Eds.), *Proceedings of the 3rd multidisciplinary international conference on scheduling: theory and applications* (pp. 159-166). Paris, France.

Even, S., Itai, A., & Shamir, A. (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4), 691-703.

Joslin, D. E., & Clements, D. P. (1999). "Squeaky wheel" optimization. *Journal of Artificial Intelligence Research*, 10, 353-373.

Merlot, L. T. G., Boland, N., Hughes, B. D., & Stuckey, P. J. (2003). A hybrid algorithm for the examination timetabling problem. In E. K. Burke & P. De Causmaecker (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling IV: selected papers from the 4th international conference* (vol. 2740, pp. 207-231). Berlin: Springer.

Özcan, E., Bilgin, B., & Korkmaz, E. E. (2008). A comprehensive analysis of hyper-heuristics, *Intelligent Data Analysis*, 12(1), 3-23.

Özcan, E., & Ersoy, E. (2005). Final exam scheduler - FES. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC'05) (vol. 2, pp. 1356-1363).

Petrovic, S., & Burke, E. K. (2004). Chapter 45: University timetabling. In J. Leung (Ed.), *Handbook of scheduling: Algorithms, models, and performance analysis*. CRC Press.

Petrovic, S., & Bykov, Y. (2003). A multiobjective optimisation technique for exam timetabling based on trajectories. In E. K. Burke & P. De Causmaecker (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling IV: selected papers from the 4th international conference* (vol. 2740, pp. 179-192). Berlin: Springer.

Pillay, N., & Banzhaf, W. (2009). A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. *European Journal of Operational Research*, 197(2), 482-491.

Qu, R., & Burke, E. K. (2007). Adaptive decomposition and construction for examination timetabling problems. In Multidisciplinary international scheduling: theory and applications (MISTA'07) (pp. 418-425). Paris, France.

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1), 55-89.

Thompson, J. M., & Dowsland, K. A. (1998). A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25(7/8), 637-648.

Ülker, Ö., Özcan, E., & Korkmaz, E. E. (2007). Linear linkage encoding in grouping problems: applications on graph coloring and timetabling, *Practice and theory of automated timetabling* 2006, Springer-Verlag, selected papers, *Lecture notes in computer science*, vol. 3867, 347-363.

White, G. M., & Xie, B. S. (2001). Examination timetables and tabu search with longer-term memory. In E. K., Burke & W. Erben (Eds.), *Lecture notes in computer science. Practice and theory of automated timetabling III: selected papers from the 3rd international conference* (vol. 2079, pp. 85-103). Berlin: Springer.

The New Timetable Era: The Single 'W' Concept and Timetable Hub

Amir Said · Rafiq Muhammad.

Received: date / Accepted: date

Abstract The main of this paper is to deliver our insight on timetable solution based on our 3 years of experience in within the Malaysian Higher Education sectors. This paper is written purely on the basis of consultant point of view rather than from a researchers expert opinion. Most of the findings are basically from our survey both qualitatively and quantitatively which are then compile and reviewed by our consultants for our development stages. Our close proximity with our pool of clients has helped our Research and Development team of consultants to develop our timetable software called Timetabler Plus initiated from version 1.0 till version 3.0 in the last 3 years. Efforts taken in solving most of the colleges, polytechnics, institutions and universities constraints and demands were built around our interest to know more and solve more. This is what this paper is all about; sharing our experiences as a timetable consultant and also our vision for the new era of timetable solutions called timetable hub.

Keywords Timetabling Solution · Commercial Product · Timetabling Process · 'W' Concept · Timetable Hub

1 Introduction

Timetabler Plus (T-Plus) is a commercial timetabling software designed to solve timetabling problem in all education institutions. With the tag word of "ultimate timetabling solution for schools, colleges and universities", the ultimate mission of T-Plus is to become a universal timetabling application for all kinds of education timetabling problems. From

Amir Said
Timetable Hub
BLK 978D Buangkok Crescent 01-233 Singapore 536978
Tel.: +65 6795 9516
E-mail: amirezy@gmail.com

Rafiq Muhammad
EEET Department, Yanbu Industrial College
P.O. Box 30436 Yanbu Al-Sinayah, Saudi Arabia
Tel.: +966 55 815 2904
E-mail: mrafiq@yic.edu.sa

a humble beginning as an in-house developed timetabling software, it has evolved into a full scale commercial software that is capable of handling diverse timetabling scenario in different education institution ranging from schools, colleges, polytechnics and universities. Since its intervention in Malaysia education market, it has made some kind of impact by securing 45 satisfied clients and currently launching it latest v3.0.

This paper provides our insight on timetable solution, purely from the commercial point of view. It is written based on our 3 years of experience in providing consultation and timetabling solution to diverse education institutions within the Malaysian Higher Education sectors. Our close proximity with our pool of clients has helped us to develop in-depth understanding of the actual problems faced by most of the industry users and knowing their actual demands for a good timetabling solution. These findings have helped us to develop T-Plus initiated from version 1.0 till version 3.0 in the last three years. This is what this paper is all about; sharing our experiences as a timetable consultant with the hope to bridge the gap between theory and practice, education and industry, and also our vision for the new era of timetable solutions called timetable hub.

At this point, we would like to strongly emphasis that this paper is lacking in some of kind of proper referencing at this moment as this paper is purely citing issues and solutions based on the practical experience having solved 45 clients based on personal consultation and coaching. Currently, this paper also does not follow a proper academic research guideline due to the time constraint. Discussion will be focused in areas of timetable issues, our analysis, our strategy, consistent problems faced by our consultants and The Future of Timetable Solution will be outlined throughout this paper.

2 Common Timetable issues in Malaysian Higher Education

From our three years of timetable consultation within Malaysian Higher Education, having closely knitted relationship with our clients has given our RD consultants opportunity to explore timetable issues ranging from minor to major problems.

In each of our 45 institutions, they faced common timetable issues either in macro or micro level. Every institution face both the micro and macro level of timetable issues during the same period or time or at different time and the degree of the problems may also varies in each institution.

Macro level involved the management level across the institutions or even the governing body when related to government institution in Malaysia. Sometimes political changes and top level changes in the management level within the Education Ministry level could result in big impact on the macro level of timetable issues. As for private colleges, there is a different set of macro level issue being commonly identified. Decisions by the management to maximize profits and implement cost efficiencies could result in lack of resources while trying to cater for each individual student demanding for different combination of subject. These may result in major constraints and conflicts in the timetable issues. In summary, Macro level usually may cause the whole timetable that has been either ready or even running to be re-adjusted in the major or minor scale. However from our experience, most macro level issues will have a major impact on the timetable output. The worst case scenario is when there is no warning sign and the changes need to take place immediately without any room of negotiation.

Micro level issues usually involved internal decision making between Head of Department or individual request by certain lecturers and the timetable coordinators level of kindness. In the normal human behavior, most timetable coordinator will only make changes to those that they feel close to them or called favourable buddies. Other common micro level issues are the Repeaters issues and staff deployment issues due to the shortage of staffs or staff turnover issues. However in most micro level cases, they are usually not as serious issues compared to macro level and can be solved in a shorter period of time. In both macro and micro level, there is one common platform which needs to be avoided, which is clashing. Clashing issue is the main reason why institutions decide to engage timetable consultants like us. We realized that improper timetable planning and the occurrence of micro and macro level issues are the major contributions to this Clashing issues. In some occasions, in order to meet the demand of micro and macro level issue there is a possibility that clashing issue could not be avoided. From our experience, the inflexibility of macro issues are more prompt to the greater possible of the incurrence of Compulsory Clashing.

Another common timetable issues within most privately-owned is the examination timetable. Examination can cause a big headache for timetable coordinators who need manage incoming examination timetable with some courses that are still running clashes need to be avoided by recognizing available rooms and available lecturers for invigilators for within the examination schedule. Another issue can arise in the examination problem is when each student subject enrolment is unique from another student commonly found in the privately run colleges which maximize profits and minimize costs.

3 Concept of 'W' Timetable Creation Cycle

From our survey and analysis citing all the common timetable problems, we have realized that all the problems are part of the one common element; lack of understanding of timetable concept. In our last 3 years R and D, we have realized that there is no common platform or even a formula for creating quality timetable. Despite the vast availability of benchmark and measurement techniques in academic timetable research, it is not known if such a benchmark ever exists for commercial timetable product. Therefore it is difficult for commercial users to compare one software with another. It is important to note that there is a huge different between evaluating a commercial timetable solution to an academic one. In an academic exercise, the primary interest is to evaluate the efficiency and effectiveness of the timetable generating algorithm. Efficiency and effectiveness refer to the ability of the timetabling algorithm to create a timetable output that satisfies most of the hard and soft constraints in the shortest timeframe. Commercial users, on the other hand are more interested in the timetabling process. Timetabling process can be summarized as from raw data to an acceptable timetable output. This is the reason why we proposed the W concept, which represents a single cycle in producing workable timetable. Achieving a single W is the ideal of every commercial timetable process.

The single W model concept is shown in Figure 1. This is regarded as the perfect timetable creation cycle for each timetable creation process. Table 1 summarizes the activities in each of these processes in the W cycle.

This W concept can act like a base formula in the creation process of perfect timetable for both user and also software developer. Achieving a single W is the ulti-

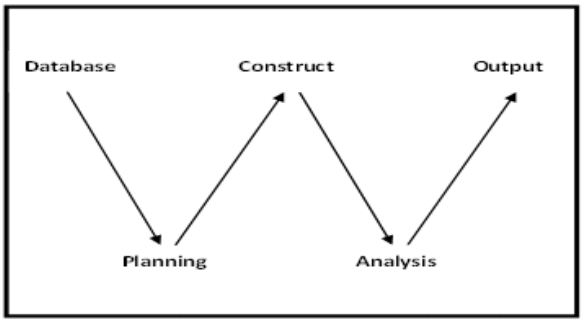


Fig. 1 Single W Model Concept (The perfect model for timetable creation cycle)

Table 1 Activities in 'W' Cycle

Processes	**Activities**
Preparation of database (Database)	- Course List - Lecturer List - Room List
Planning by management (Planning)	- Who teaches What. of lectures/labs/tutorials per subject per week - Rooms allocation per department - Merging classes according to demand - Relief/Part-timer lecturers/teachers - No of group of student for each batch - Start and End time inclusion of break time - List of prioritize subjects. (Eg: subjects having 3 hrs or more duration, compulsory subjects etc) - Plan and Understand all the Constraint
Execution: Constructing timetable (Construct)	- Construct Start and End time - Listing all available room - Analyze and identify constraints for batch of students, courses and lecturers - Construct timetable for (as sequence): 1. prioritize subjects 2. remaining subjects
Analyze the completed timetable (Analysis)	- No clashes for batch of student, courses, rooms and lecturers. - All offered subjects are included into the timetable with the right constraints. - Lecturer timetable daily/weekly workload/schedule - Student timetable - daily distribution of lessons in a week - Rooms utilization
Output and distribution process (Output)	- Output need to be copied again to the right template

mate target and goal for every timetable coordinator as it saves time and resources. In order for a perfect W concept to works, timetable coordinator needs to understand each of the W components; Database, Planning, Construct, Analysis and Output. These five W components are also a good guide for a new timetable coordinator who has no knowledge on timetable creation cycle.

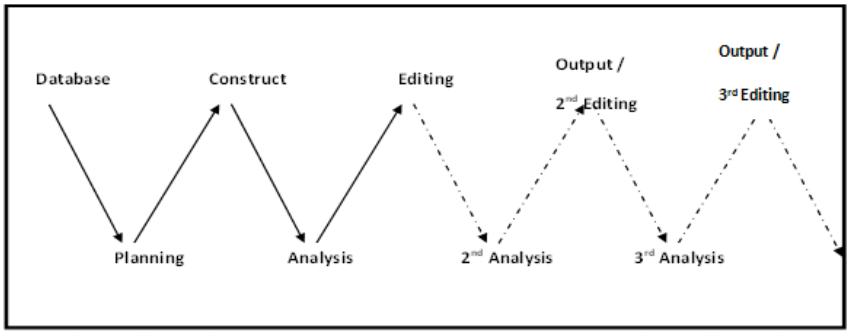


Fig. 2 Multiple W Models (Commonly found in the Conventional way concept)

In practice it is quite impossible to obtain a perfect W. Many modifications would come along the way which will disrupt the perfect 'W'. This results in the multiple 'W' cycle as shown in figure 2. The quality of timetabling process in commercial world is measures by the ability to minimize the 'W' cycle, ultimately to the perfect 'W' (i.e. a single cycle). Large W cycles denotes inefficiency in timetable creation process.

As it is obvious from the 'W' cycle, construction of timetable is just one of the processes. From the commercial point of view, good timetabling software is not only software that is capable to generate the output in the shortest period of time, but is also capable to assist in other processes in the cycle. One particularly important feature which often neglected is the ability of the software to manage micro changes (editing) to the result without affecting those that are already in their desired place.

In this paper, we are proposing the W concept a benchmark to measure commercial timetable applications. The quality of a commercial timetable solution is measure by its ability to minimize the number of W cycle. The development and improvement of any timetable system should also take into account of W principles. In this paper, based on our experience, we will explain about the relationship of our software being developed in the last 3 years based on the W concept.

4 Introduction to Timetabler Plus

When Timetabler Plus (T-Plus) is being introduced in July 2007, its main mission is to be the ultimate timetabling solution for schools, colleges and universities. T-Plus is carefully designed and constantly updated to cater the needs and demands of timetabling process. Table 2 summarizes the features of Timetabler Plus.

With friendly icons and lightweight standalone application, T-Plus has made some kind of impact since v1.2 in July 2007 launching by securing 45 satisfied clients and currently launching it latest v3.0 Apart from providing excellent technical support service, our professional consultants provide training and consultancy packages to our clients.

4.1 The Plus Factors of Timetabler Plus

Timetabler Plus offers various constraints setting for each/all:

Table 2 Summary of T-Plus Features Benefit

Features	Benefits
Enables simultaneous constraints setting for Program, Batch, Subject and Lecturer	Fast, efficient optimization with complete user control
Managing the allocation of specific constraints in subjects and lecturers using the TREE STRUCTURE terminology (PARENT and CHILD concept)	High flexibility and reusability in the creation producing better result for any type of educational institution
USER-FRIENDLY interface with native Windows icon	Easy to learn and use even for non- computer literate
Integration with MS EXCEL for database and printing purpose with various printing format available	Transferring of the current database for new user made easy. Final output simple to customize
AUTOMATED GENERATE button function based on constraints settings PLUS the option of Manual Allocation	Reduction in Cost, Time and Manpower. Clash Free Result (Auto-Generate) and Reminder (Manual Allocation)
(Latest Edition v2.8) Print preview for Individual Lecturer, Room, Day and even Batch of Student with customized Header and Footer	Faster in Analyzing the output and Quicker in Producing the Final Output

1. Lecturer(s) Constraints/Preference:
   - Specific time or day he/she does not want to teach and
   - Preferred Room/s
   - Avoid Teaching Consecutively
2. Constraints for Each Program/Department/Faculty/Batch of Student/ Subject(s):
   - Specific time or day to Include/Exclude
   - Preferred Room/s
   - Same Subject (different day) or Merging Request (Same Time Slot)
   - Adjacent Classes
   - Same Room for a particular group of students
3. Room Constraint/ Specific Room Types:
   - Each Room Capacity
   - Each Room Function/Equipment
4. Style of Generating Output (Generate Constraint):
5. Prefer to start the class in the early morning, afternoon or evening
6. Prefer the 1st class to start on 1st Day of the week (Monday) or last Day of the Week (Friday)
7. User can have the choice for 2 hours class or more to span across the General Break
8. Rooms can be optimized according to the capacity

Timetabler Plus also offers various output formats for individual/master timetable for:

- Lecturer
- Batch of Student
- Room
- By Day

There are three ways for users to print their finalized timetable:

1. Exporting to Microsoft Excel
2. Print directly from Timetabler Plus
3. Export to HTML

4.2 Timetabler Plus W Concept

Any good timetable software is measured by its ability to minimize the W cycle, ideally achieving a single W. Other than the W concept, there is not known type of measurement or benchmark that can act as a tool to differentiate the quality amongst timetable software. We have committed our development of Timetabler Plus software in line with the single W concept, trying to automate every process in the W cycle. While we understand that a perfect W difficult to achieve (unless it is a small organization with small timetable), our goal is to help organization minimizes the W cycle. Table 3 outlines the timetable creation processes and activities using Timetabler Plus in relation to improving every process in the 'W' cycle.

5 Timetabler Plus Strategies

One of the key misconceptions of the users is that a timetabling solution is like a silver bullet to their timetabling problem. This is indeed untrue. Many of the problems lie in the human aspects as well as in planning and management. Our experience with timetabling consultation since July 2007 made us realized that almost all timetable problems could not be solved by many timetable solutions due to lack of understanding between timetable coordinators with the timetable solutions and vice versa. Therefore in order for Timetabler Plus to make a good impact, we have devised a structured strategy to deal with each of our clients in three Cs steps: **C**onsultation, **C**ustomized Timetable, and **C**ustomized Training.

5.1 Consultation

Consultation is the key to understanding clients problems. Understanding our clients unique scenario is the key to successful implementation. For that reason, our consultant will first be consulted about all their timetable issues. Our experience has shown that understanding just their demand and constraints are not sufficient to make a quality timetable. We will need to understand the traditional way or sometimes what we called customary way of creating the timetable. Even within the same organization, different department may have different set of customary demand which may even differ when new person takes over as a new Head of Department (micro level of constraints). Based on the understanding of clients problem, Timetabler Plus will be customized to cater to the needs of our clients. The crux of the challenge lies in the ability of Timetabler Plus to adapt to different clients scenario without having to customize the software at the code level.

Table 3 Process of Timetable Creation using Timetabler Plus

Processes	Activities
Planning by management and Database preparation	- Conventional planning and database preparation applies - Database can be easily exported from the ready-made Timetabler Plus Excel - Course, lecturer and room list can be categorized according to department. - Future editing can be done easily - Construct the general timetable structure; general timeline and list of room. - High flexibility in designing timetable structure.
Data entry and Constraint Setting	- Modeling data and constraints in tree structure with parent and child relationship. - Future editing will be easily done in the existing tree structure. - Apply constraint simultaneously for each faculty, program, and batch of student, subject, lesson and lecturer according to individual demand.
Auto-Generate	- Generating the timetable according to preference and priority - Various generating options, offsetting between fast and quality output. - Incremental generating of output to allow timetable coordinator to build timetable incrementally.
Analyze the result and Edit (if required) simultaneously	- Fast track on changes with quick analysis results - Locking feature to preserve desirable output - Various analysis techniques to analyze various data
Output and distribution process	- Offer various formats of output; Native output, Excel, and Html - Customizable output to suite to different organizations needs - Use independent file concept to enable easy management of multiple scenario. - Multiple file concept also make timetable process easily recyclable.

5.2 Customized Timetabler Plus

Based on our consultation inputs, we will try to accommodate all the demands and constraints by creating a custom timetable structure for our clients. Here lies the key strength of Timetabler Plus. As Timetabler Plus uses a model driven approach, it is really up to the creativity of the timetable consultant to model the constraints. Similar problem can be model in different way as long as the end result is the same. Not all features of Timetabler Plus is required by all client, therefore we only expose those that are of their concern.

5.3 Customized Training

Finally, we construct a Customized Training program for our clients to enable their timetable problems and demands to be solved using Timetabler Plus software. This may need a proper guidance by our consultants and priority is given for older generation

while more attention to bigger organizations. The training programs may even differ for each department in the similar institutions as demands may vary from one another. We usually customized the training program based on demand driven and the uniqueness of each individual/department/institutions. Our main objective at the end of the training session is for our clients to enjoy constructing quality timetable using Timetabler Plus by optimizing the full functions of the software.

6 Impact of Human Character in Timetable Software Implementation

In each of our clients, that we have consulted and trained, we usually faced some problems that can be described as consistent problems which are found in most institutions. These problems usually cause some kind of irregularity in the flow of our training program in relation to timing and productivity. It is called the Human Character and even though each human portrayed different set of characters, they usually have a common element; Demand. This consistent Demand problem in Human Character will affect the single W concept as planned in our training packages. From our findings, Demand problems are commonly categorized accordingly:

- **Demand for Attention**: Usually for older generation who is resistant to change. This is due to lack of confidence in using the computer as they are prone to do anything manually.
  Impact: Too much time spent on them while other coordinators may be accidentally being ignored or lesser time attended to cater their needs.
- **Demand for Excuse from Attendance**: Usually occurs for those that are forced to be inside the training program. Involuntary participants are being obliged to follow orders from higher authority or to gain credit hours for attending the training program.
  Impact: This kind of character usually gives a negative impact to the others and at times they may even create unnecessary scenes that can result in depress or moody environment. The matter can be worst if each department timetable is dependent on other department progress rate in completing the timetable output.
- **Demand for Reasoning**: This usually happen when someone is too enthusiast with the training program or even the Timetabler Plus. He or she will usually wants his/her voice or opinion being heard and sometimes can even dominate the entire training sessions.
  Impact: This character can have both positive and negative implications. On the negative side, all the participants may be frustrated with too many questions being posed by the individual and it can also create some kind of ill feeling due to the limelight being extensively shown to him/her.
- **Demand for Retraining**: This behavior is normally occurred when there is new timetable coordinator replacing the experienced ones. This is a common scene in any institutions due to the rotation of staff responsibility, change in management, higher posting and staff turnover. There are two main issues in connection to new coordinators. Firstly they need to understand the technique of creation of timetable in accordance of their department/institution demands, constraints and customary way. The other is the need to understand the functions of Timetabler Plus software. Another scenario is when existing timetable coordinator who has been trained once will request for another retraining program due to the fact that they only construct

the timetable twice annually and in between there could be a minimum of three months lapse. By the time they need to create new semester timetable they might have forgotten all the functions of Timetabler Plus.
Impact: Main impact of this kind of human behavior will contribute to the quality of the new semester timetable output. Not all institutions have annual budget for the retraining program or there is even some cases whereby there is no unanimous decision by all timetable coordinators to spend time and money for another session of training. The critical impact may even cause some timetable coordinators to abandon Timetabler Plus and even return the conventional (manual method) way.

The human character if ignored may have negative consequence on the success of timetable software implementation. It will be unfortunate if a timetable solution is rejected, not because it cannot cater for the organization needs, but due to the negative image portray by human as a result of their resistive behavior.

7 The Future of Timetable Solution

In our 3 years of timetable research and consultation within the Malaysian Higher Education institutions, the human behavior factors which were described earlier as consistent problems need to be eliminated in order for the single W concept to be achievable. In contrast, many software houses will argue that a good algorithm, friendly interface and informative output may solve all the timetabling problems. Even till this stage, our consultants have been working very hard to make Timetabler Plus the ultimate timetabling solution in the market. However, as time goes by and all the efforts are being done to accomplish this mission, our Timetabler Plus is still unable to solve the consistent problem caused by human behavior.

Therefore, we have come out with another solution which will result in the single W concept to be more achievable. Quality timetable is a perfect single W concept and we believe all efforts need to be aimed in achieving this mission. In this paper and conference, we will like to propose a service oriented timetable solution called Timetable Hub that will eliminate the problems caused by Human behavior thus resulting in a quality timetable output within the single W concept.

7.1 Timetable Hub: New Era of Timetable Solution

Every timetable software in the market including Timetabler Plus has been trying hard to be the ultimate timetabling solution with better algorithm, interactive interface and customized output while forgetting that a single W concept can only be achieved if there is no barrier in the human behavior. In every timetable process, each stage indicated at the single W concept outline the importance of every stage connecting one to another in order to produce a quality output. Normally the relationship between human behavior factors and software is critical at the Construct stage whereby any human elements may influence the result of the subsequent stage; Analysis and Output. Therefore it can be claimed that the quality of timetable lies heavily at this Construct stage which will then acknowledge the importance of relationship between human factors and the software itself. Table 4 outlines our observation on some of the most common human behaviours that have impact on software implementation (particularly in timetabling software).

Table 4 Human Behavior Factors vs Software

Human Behavior Factors	**Impact to Software**
Age Factor	How do the icons, language and functions of software be able to accommodate different types of young, middle and old age contributors?
Attitude	Any software in the market will not be able to directly influence the attitude of the user. Definitely those with positive attitude will be able to finish the Construct process better and quicker than those with negative attitude
Computer Literacy	Will the system be able to provide valuable assistance (Video Tutorial, Easy Manual, Trainer and Help Support) with user friendliness of icons and functions for all level of computer literacy?
Environment Pressure	Will the system be able to influence how each individual in the timetable committee within similar institutions react to one another and thus contribute to the peer pressure? Will the system be able to eliminate ill feeling between colleagues? Will the system be able to reduce the pressure of timetable coordinators due to the Macro and Micro issues in the institutions?

Based on our survey, it is understood that within some of our clients, Timetabler Plus was able to gain a single W concept as they are able to anticipate micro, macro and human behavior issues at the planning stage. Commonly, just like any other software, human behavior will usually influence the Auto-Generate process in the Construct stage in the Timetabler Plus W cycle. Therefore, the introduction of Timetable Hub will be able to fill the gap between human behavior factors and software in the Construct Stage within the single W concept. Timetable Hub will be an application based on Software As a Service (SAS) where clients who engaged the service will be required to input the necessary data, constraints and requests (demands and customary behavior), while the 'Construct' stage will be handle by the service provider. The aim is to take out what the end users is not concern with. The system will Auto-Construct to produce few outputs that will then allow the user to choose from varieties of them based on their preferences. From the chosen output, user will then analyze with their preferences. Changes or request can be made in the given template. Once confirmation about the result, the output will then be able to be printed out in different type of format requested by the user. Further changes in the timetable can be accessible through the given templates and the process of output will then be repeated again.

Since the system will be 100% responsible (without any human interference) for the Construct stage, it will be able to eliminate the Human Factor vs System problem as discussed before. Even the changes to be made can be done easily and the output can be customized according to the clients demands. Furthermore the upgrading of the software or system will not hinder the clients ability to indulge in the creating quality timetable without even learning about the latest upgraded functions.

8 Conclusion

As we are currently living and highly dependent on technology to solve our daily problems, we sometimes failed to realize the importance of human behavior factors in influencing the quality of any kind of output. Constructing timetable has always been a challenge for software house to produce solution comprise of good algorithm, friendly interface and informative output. A single W concept can be the tool to measure the qualities of any kind of timetabling solution in the market. From a timetable consultant point of view, the new era for timetable solution is called Timetable Hub which is able to eliminate the human behavior in the single W concept cycle.

For more information about Timetabler Plus, please visit
http://www.timetablerplus.com.

Cross-Curriculum Scheduling with Themis

A Course-Timetabling System for Lectures and Sub-Events

Abstract We report on a practical implementation of a curriculum-based course-timetabling system for pre-enrolment scheduling that is successfully used in our university. The implementation is based on a sophisticated model that captures essential real-world requirements in terms of course-structure modelling. Our tool Themis allows to handle courses that have sub-events and that are shared between different programs of study. It can also consider whether a shared course is mandatory or optional in each curriculum. Themis supports a cyclic and interactive workflow and offers comfortable means for editing model data and timetables.

Keywords system demonstration · course-timetabling system · real-world model · practical implementation · cross-curriculum scheduling · sub-events

1 Specific Challenges in Real-World Course-Timetabling

Various constraints of different type, uncertain information and competing goals turn curriculum-based course timetabling for real-world settings into a challenging task [2]. In case of our department we observe that many aspects of this scheduling problem can be modeled using typical entities, constraints and cost components. In particular, courses are attended by students from different programs of study and each program has its own curriculum. E.g., the course *Theoretical Computer Science* has first-year students from the two Bachelor programs *Computer Science* (CS) and *Internet-based Systems* (IBS), and second-year students from the Bachelor program *Digital Media and Games* (DMG). For a standard model that covers most aspects of our setting we refer to CB-CTT from [1]. However, to obtain practical solutions we also need to consider some additional requirements:

– Each course has not only lecture events but also a number of smaller sub-events associated with it, like tutorials or laboratory classes. All students attending a lecture are partitioned into these sub-events, each of limited size.

Heinz Schmitz (corresponding author), Christian Heimfarth
Trier University of Applied Sciences
Department of Computer Science, Schneidershof, 54293 Trier, Germany
E-mail: schmitz@informatik.fh-trier.de

– The same course can be mandatory for some students but optional for others, depending on their program of study. E.g., students from IBS have to take the course *Web-Technologies*, while CS students may choose this or some other course to fill one of the placeholders in their curriculum. Collisions between optional courses should be avoided to offer students a large number of possible choices.
– Lectures and sub-events can require more than one timeslot. In some cases, even sub-events of the same course have different numbers of timeslots to account for different skill levels.

As a consequence, we have strong dependencies in terms of clashing constraints across different curricula. Moreover, we need to construct a timetable for each term *prior* to student enrolment. So there is only limited information about what students from which program attend what lectures, and we have no information about sub-event enrolments. Also, several other organisational requirements have to be taken into account: No disruption or noticeable re-scheduling during a period is wanted, and, on the other hand, there is strong need for manual editing and updating, especially during the first weeks. The typical quantity structure of a problem instance for our department has about 800 students, 25 teachers and 140 events to be scheduled in 27 timeslots and 15 rooms. We must consider curricula of three Bachelor programs, two Master programs and some other post-graduate training programs, and we expect that the number of programs and events increases in the next years. Altogether, we are faced with a complex scheduling problem for which it seems nearly impossible to obtain feasible or even optimized solutions without strong tool support.

2 Overview of Themis

The ambitious goal of Themis is not only to implement some experimental algorithms but to provide a reliable and comfortable software system for our schedulers that *really* solves the *real* problem. In this sense Themis can be understood as a contribution to the research agenda set up by McCollum in his paper [2]. We started Themis in 2006 and it is under continuous develoment since then, including a complete re-implementation in 2009 to account for the lessons learned. Right now Themis is successfully used to produce workable and optimized timetables in our department and in other departments of our university.

Inspired by the manual work of our schedulers prior to Themis, the tool supports an interactive and cyclic workflow consisting of the steps (1) management of model data, (2) allocation of anonymous groups of students to lectures and sub-events, (3) automatic timetable generation, (4) manual editing of timetables, (5) presetting of (parts of) a timetable, and returning to (1), (2) or (3).

2.1 Model Data (Step 1)

An independent set of model data, usually one per scheduling period and institutional unit, is organized in a *project*, typically called (*CS-Department*, *SummerTerm2010*), (*EngineeringDepartment*, *WinterTerm2009*) and so on. This structure allows to model different scenarios for the same period independently. The user can copy existing projects and reuse model data.

Information Modelling. THEMIS allows to handle the typical main entities in course timetabling, as there are timeslots, lectures and their sub-events, teachers and rooms, all with a number of specific attributes and relations among each other. For example, a course has a lecture event and a number of sub-events of a maximal size; an event requires one or more timeslots, has one or more teachers and requires or excludes a number of resources offered by rooms (e.g. computer workstations); teachers have, among other attributes, *preferred*, *available* and *not available* timeslots, and so on. Moreover, we have introduced an entity called *curriculum-semester-combination* (CSC) to model groups of students that need to follow a certain set of lectures determined by the curriculum they are enrolled to. Typical values are *BachelorCSSecondSemester* or *BachelorIBSFifthSemester* and the like. We determine the number of students in each CSC and assign one or more CSCs to each course to express its multiple usage in different curricula.

Solution Modelling. As usual, we distinguish between *hard constraints* that a timetable must fulfill to be feasible, and *soft constraints* that it should additionally fulfill in order to be 'good'. Right now, our list of constraints includes typical hard constraints, like 'no two events in the same room at the same time' or 'if two events are modelled in *mustFollowTo*-relation, then the timeslot of the second event must immediately follow the timeslot of the first event on the same day'. THEMIS also knows a rather large number of soft constraints that can be used to optimize feasible timetables. Each violation leads to penalty points that are accumulated for a timetable. Examples are 'minimize free timeslots between events for students of the same CSC', 'use preferred timeslots of teachers' and 'a sub-event should not be the only event on a day for a CSC'. Clearly, accumulating penalty points blurs the boundaries between different optimization objectives. So it is important to visualize for the user how the sum of penalties of a timetable is composed. THEMIS offers a *tree view* that clearly presents all details of a timetable score. Moreover, the user can choose weights to assess different objectives, up to the possibility to exclude objectives from optimization by choosing weight zero.

Each project memorizes the list of all timetables that have been generated so far in this project, so it is always possible to go back to earlier attempts. Each run of a generating algorithm adds a new timetable to this list. All timetables in a project are dynamically evaluated with respect to the current set of model data, i.e., in case of an update, all timetables in the project are automatically re-evaluated to determine feasibility and penalties.

2.2 Lectures and Sub-Events (Step 2)

Lectures and sub-events together with their associated mandatory and optional CSCs impose extra complexity to timetable scheduling. We briefly describe our approach to this problem with help of an idealized and reduced example.

Example. Suppose the course *Web-Technologies* is mandatory for the CSC *BachelorIBSThirdSemester* with 50 students and optional for the CSC *BachelorCSFifthSemester* also with 50 students. We estimate from past terms that 25 students from *BachelorCSFifthSemester* will choose this course and introduce four sub-events for it, each with a limit of 20 students.

After these steps are carried out for all courses in the project, we partition each CSC in anonymous blocks of students and map these blocks to the sub-events of the courses.

This is sufficient if a course is mandatory for a CSC since all of its students attend the lecture. In case of an optional course for a CSC, we partition only the estimated number of attending students into such blocks and map them to the sub-events of this course as well.

Example. (continued) The number of 50 students from *BachelorIBSThirdSemester* is partitioned into blocks of 20, 20 and 10, the number of 25 students from *BachelorCS-FifthSemester* into blocks of 10, 10 and 5. As a result we get two sub-events of *Web-Technologies* each with 20 students from IBS, one sub-event with 15 students from CS only, and one mixed sub-event of 20 students.

THEMIS has algorithms that support these partitioning and mapping steps. To partition the number of students it chooses the smallest possible number of blocks, each having a size from a user-defined range. These blocks are the basic units to allocate students to sub-events. The mapping is done by a greedy algorithm that assigns blocks to sub-events such that the heterogeneity with respect to different CSCs is minimized. The algorithm considers blocks in descending order w.r.t. their size and assigns them to sub-events in a best-fit manner. We do so because we expect less clashing conflicts during timetabling if CSCs share only few events. Partitioning and mapping is carried out as a preliminary step before timetable generation as part of the model data. Clearly, this partitioning and mapping step is a non-trivial optimisation problem on its own that deserves further investigation. Blocksizes and mappings can also be edited manually during the overall interactive workflow.

After this step we have information in our model about what students from which CSC attend what lectures and sub-events. This is further exploited to determine clashings of events during timetable generation by specific hard and soft constraints. An example of such a hard constraint is 'no two sub-events of two mandatory courses in a CSC with the same associated block in the same timeslot'. Penalties result, e.g., from 'two sub-events of different optional courses of a CSC in the same timeslot'.

2.3 Timetable Generation and Editing (Steps 3, 4 and 5)

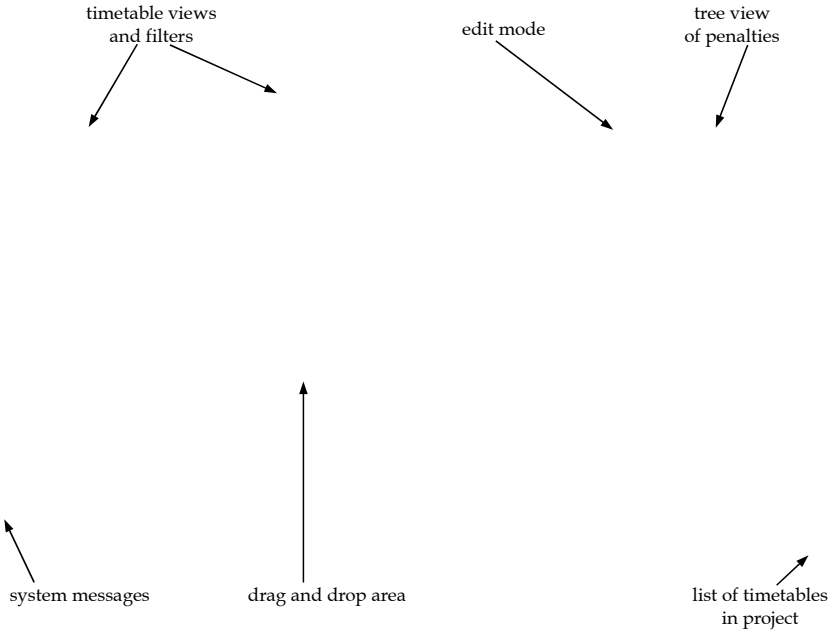
The user can choose to call an algorithm from scratch or to select any existing timetable in the project as an initial solution for some timetable-generating or improving algorithm, respectively. So far we have implemented the following set of algorithms (for common algorithmic approaches to this kind of problem see, e.g., [3]):

1. A constraint-based algorithm to obtain a feasible timetable (an efficient implementation of backtracking with forward-checking, degree heuristic, minimum-remaining-value heuristic and least-constraining-value heuristic).
2. A variant of algorithm 1 where the order in which the (timeslot, room)-values for each event are chosen depends on the penalty of the resulting partial timetable.
3. A local-search procedure with various parameters to improve feasible timetables.

All algorithms display their current best values and can be interrupted by the user. Algorithms 1 and 2 are based on a careful analysis of all hard constraints to reduce the range for the variables in advance and during backtracking. It turned out that in particular algorithm 1 is useful to reveal inconsistencies in model data very early. While the resulting penalty after algorithm 1 is fairly high, we obtain optimized timetables with small penalties from algorithms 2 and 3.

THEMIS has comfortable drag&drop-support for editing timetables. In the *free mode* events can be moved arbitrarily to any timeslot. However, schedulers find it very helpful to work with the *supported mode* of THEMIS during timetable editing. After choosing an event all timeslots are coloured red or green, depending on whether a move of this event to that timeslot results in a feasible timetable or not. Moreover, when dragging over red timeslots, the user is provided with information about what constraints are violated. In case of a green timeslot the new penalty is displayed in advance. Changing the room of an event is assisted by a similar mechanism in this *supported mode*. It is also possible to manually delete and insert events into an existing timetable.

Also other features of THEMIS turned out to be useful in practice. To display only specific aspects of a timetable it is possible to use filters, e.g., to show the timetable for a certain CSC, a certain teacher or a certain room. Timetables can be exported in a universal format for further publishing. Moreover, in order to support an incremental approach THEMIS allows the user to freeze parts of a timetable. As a consequence, all algorithms must maintain this presetting. Schedulers use this feature to produce similar timetables when single entities are added or updated. The following figure gives an impression of the screen for editing timetables.



2.4 Software Architecture and Engineering Aspects

The current release of THEMIS is realized as a pure java application based on the frameworks Hibernate[1] and Docking Frames[2]. It has a modular architecture with separate components for algorithms, graphical user interface and data management. Deployment

[1] http://www.hibernate.org
[2] http://dock.javaforge.com

is rather easy since THEMIS comes as a single jar-file, already including its database HSQLDB[3] (which can easily be changed to any other database working with Hibernate). We want to point out some critical aspects that we have paid attention to while developing THEMIS, but which do not deal with algorithm design in particular:

- Special care must be taken to maintain system-wide *data consistency*, i.e., due to complex dependencies between model entities, referential integrity must be carefully controlled when edit and update actions are preformed. This also includes some thoughts on storage management for the persistent entities in the model.
- There are parts in the code that are frequently executed and where the user expects very fast response times. Among others, *efficient implementations* of feasibility checks are needed. This is usually carried out on the data-structure level and cannot be discovered in some abstract pseudo-code from a research paper.
- Common *software-engineering principles* like design-patterns, encapsulation and no-duplicate-code must be strictly followed. Especially model entities and code to check constraints tend to spread all over the source code with the consequence, that maintenance and further development of the system become impossible.
- It is helpful to work with a single programming environment and language which leads to *seamless debugging* of the complete application. The latter is often problematic when different programming languages are used at the same time. We found that algorithms can be implemented in java reasonably fast (compared to other languages) when restricted to native data types.
- Due to the complex nature of the application domain there is strong need for *quality assurance* in the development project.

From our experience, disregarding a single of these aspects can make the difference between a working system and an instable prototype which cannot be used in practice. As a consequence, there is need for various expertises in the development team which makes such a project attractive also from an educational point of view (for Computer Science students). Luckily, we observe a high motivation of students to contribute to a system that affects their own academic calendar.

3 Future Work

THEMIS is primarily designed and used to produce timetables for single departments in universities. This is an appropriate approach in our case since only a small number of rooms is centrally owned and must be shared between different departments. However, we observe that there is an increasing number of programs in preparation that are offered in cooperation between two or more departments which implies that a common timetable is needed. We will investigate how THEMIS behaves on these larger instances and what new requirements arise.

Moreover, we want to further investigate algorithmic and modelling aspects of cross-curriculum scheduling of mandatory and optional courses and their sub-events (section 2.2). One aspect is that it seems to be difficult to generate timetables that guarantee a certain minimal number of non-overlapping optional lectures and sub-events in each CSC.

[3] http://hsqldb.org

Another aspect is the presence of uncertainty in the input data which cannot be avoided in pre-enrolment scheduling. This becomes even more problematic if students from different programs need to be allocated to common sub-events of optional courses. Themis in its current version offers means to control this allocation in terms of block mappings as described above. However, the critical part in this approach is that it is based on the educated guess of the scheduler about how many students of a CSC will presumably attend what optional courses. In practice, this uncertainty is now handeled by scheduling small over-capacities (an approach with definitely tight limits) and manual editing after enrolment, e.g., by introducing additional sub-events. We think that there is strong pratical motivation to further investigate models and algorithms for timetables that are *robust* with respect to this uncertainty. A first step could be to identify and select scenarios that represent situations where students have chosen other optional courses than expected. Timetables should then be evaluated with respect to their ability to remain feasible in different scenarios.

Acknowledgements We are grateful to the anonymous referees for their helpful comments. We would also like to thank all other current and former members of the Themis development team for their contributions which are F. Hermes, P. Kranz, J. Pauken, P. Schiffgens, B. Schumacher, J. Sonntag, S. Stoffel, M. Stüber, M. Weiser. We are thankful to the Nikolaus-Koch-Foundation for their financial support.

References

1. F. De Cesco, L. Di Gaspero, and A. Schaerf, *Benchmarking Curriculum-Based Course Timetabling: Formulations, Data Formats, Instances, Validation, and Results*, In: E. Burke and M. Gendreau (eds.), Proceedings of PATAT '08 (2008).
2. B. McCollum, *University Timetabling: Bridging the Gap between Research and Practice*, In: E. Burke and H. Rudov (eds.), Proceedings of PATAT '06 (selected papers), Lecture Notes in Computer Science 3867, pages 3-23 (2007).
3. A. Schaerf, *A survey of automated Timetabling*, Artificial Intelligence Review, volume 13(2), pages 87-127 (1999).

The Perception of Interaction on the University Examination Timetabling Problem

J. Joshua Thomas Ahamad Tajudin Khader Mohammed Al- Betar

Bahari Belaton

Artificial Intelligence Lab, School of Computer Sciences, University Sains Malaysia, Penang, Malaysia

joshopever@yahoo.com, tajudin@cs.usm.my, mohbetar@cs.usm.my

bahari@cs.usm.my

Abstract. In real-world perspective, educational institutions have come across a mixture of formulation for the examination timetabling problem and still semi- automated scheduling systems are in practice. In this paper, we look into the knowledge abstractions techniques to reduce the complexity of problem solving for university examination timetabling problem. The methods consists of *recapitulate, visual analysis heuristic (VAH), and specification.* The recapitulation groups the successive components, thus reducing the size of the problem. The clustering heuristics, partitioning the problem into easy and difficult components interacting through abstracted pools. The hierarchy of pools allows the user to intervene in conflict resolution at the most appropriate level of abstraction. The specification simplify the decouple components to aid the user in assessing the stiffness of the problem. We propose an algorithm that interleaves these processes. The combinations of these three techniques are evaluated with the real-world examination timetabling room allocation problem scenario. The merits of our approach are minimizing the need for backtracking, provides interactive visual models framework to understand the conflict resolution and offering a comprehensive direction to feasible solution.

Keywords: Keywords: Examination Timetabling, Visualization, Conflict resolution

1 Introduction

The common feature of constraint satisfaction problems is the fact that each variable ranges over a finite domain. Problems in this class are theoretically desirable. A simple algorithm can be exhibited that eventually finds the solutions, if any, and terminates. The real problem is efficiency that is, finding effective feasible solution. Scheduling, assignment, planning are closely related problem. The main reason is they intercede at different time scales and we are looking at that class of problem in the paper.

The scheduling of examinations to time periods is a problem faced by many educational institutions at the end of the academic semester (Lewis et al. 2005a, Burke and Kingston et al. 2004). The basic form of the examination timetabling is tackled by assigning a set of examination to predetermined time periods so as to satisfy the predetermined constraints. The constraints are either hard or soft. The former must be satisfied in order to come up with a feasible timetable while satisfying the soft constraints is desired by not essential. The conventional research objective is to minimize the total number of soft constraint violations in a feasible timetable.

Assorted and modern approaches such as hyper heuristics (Burke, McCollum et al.2007), tabu search (Chiarandini et al.2006), evolutionary algorithms (Doyle 1979, Cote 2005, Burke and Newall 1996) and simulated annealing (McCollum and McMullan 2008), particle swarm algorithms (Erben 2001), and harmony search algorithms (Al-Betar et al. 2008) are in the list on solving the examination timetabling problems. Graph based techniques are beneficial to constructing solutions by ordering the exams that have not yet been scheduled according to the obvious difficulty in scheduling that exam into a feasible timeslot. In the newest dataset establish by ITC-2007 or other real world case, finding a feasible solution by using graph colouring heuristics becomes implausible.

The work carried out by (Lewis and Paechter 2005a, Lewis and Paechter 2005b, Lewis et al. 2007) can be good evidence that the real-world examination instances normally cannot be feasibly tackled using classical graph colouring approaches. The author employed a grouping genetic algorithm in order to solve the first-ordering constraints against 60 test problem instances establish by him for post-

enrolment course timetabling. Eventually, the researcher was not able to find feasible solutions for all problem instances.

In fact, the dataset established by the first international timetabling competition (TTComp2002) for course timetabling problem has been not concern in the difficulty of finding a feasible solution. It is awarded the participants based on those who obtained a feasible solution with the least number of sort constraint violations (Chiarandini Birattari et al 2006). Lately, the attentions of the timetabling research communities have been turned toward closing the gap between the fabrication datasets using in research and the real-world dataset especially when the influential research carried by (McCollum et al. 2007) was published. As such, the newer dataset released by ITC-2007 (McCollum et al. 2009) for the post enrolment was more realistic in which the term 'distance to feasibility ' is introduced as a factor of evaluating the solution obtained and thus the competitor are win when they find a feasible schedule for some problem instances.

In extending and explaining the techniques behind the Visual Analysis Heuristics (VAH), it is beneficial to look at the reasons for investigating the topic, into a broader research context. It is essential to clearly define the subject of interactive visualization as a basic formation. In real-life examination scheduling it becomes highly complex making valid solutions where the visual representations can be used as a guidance measure to reduce the complexity. Our previous work has shown a visual analysis framework on the pre-processing (J.J. Thomas et al. 2008, J.J. Thomas et al. 2009) over the examination timetabling problem.

There are a number of software commercial systems available for examination scheduling (Erben 2001) each of which use apt user interfaces which allow user to address the design and implementation issues of the search heuristics in a more standard way. These computing system need human knowledge to intervene few processes to ease the process and learn visually.

The interest in this research work is on solving the examination allocation problem, an assignment usually to one or more human machinist. For example human schedulers or human decision makers who applied a heuristic assignment procedure, based on the knowledge and with little guidance from computer software to avoid clashes in relation to solution the measurement of evaluation function is not feasible

or ideal and it was allowed to adjust the weight to enable the solution (Cumming et al. 2006). In this paper we cast the examination timetabling problem and the organization of the paper is the subsequent paragraph.

The paper is organized as in sections; Introduction and problem definitions are in Sections 1 and 2. Section 3 illustrates solution overview. Section 4 has the knowledge abstractions and introduces two techniques *recapitulate* and *specification*. Section 5 discusses how the knowledge abstractions are used to reduce the size of the problem, the use of computation on *recapitulate.* In Section 6 we suggest a new heuristic method to divide the problem into subcomponents and classify, easy to solve and hard to solve (over-constrained) the subcomponents are interactive with a tree structure, these interaction explains the conflicts. Section 7 elaborates the conflict detection. In Section 8, we introduce an inadequacy of the disintegration strategy, namely *wide exam result*, and introduce to overcome the problem by defining *associations* among the conflicting problem components where the users can intervene in problem feasible solution and contribute preferences interactively in Section 9. We have applied the methods introduce earlier on to the framework to a real-life examination timetabling problem in Section 10. Section 11 draws conclusion and discussion.

2 Problem definition

Examination timetabling problem has to assign exams to periods (timeslots) and to rooms. This would be easy, except for the constraints that need to be valued as much as possible. E.g. a student cannot take 2 exams at the same time.

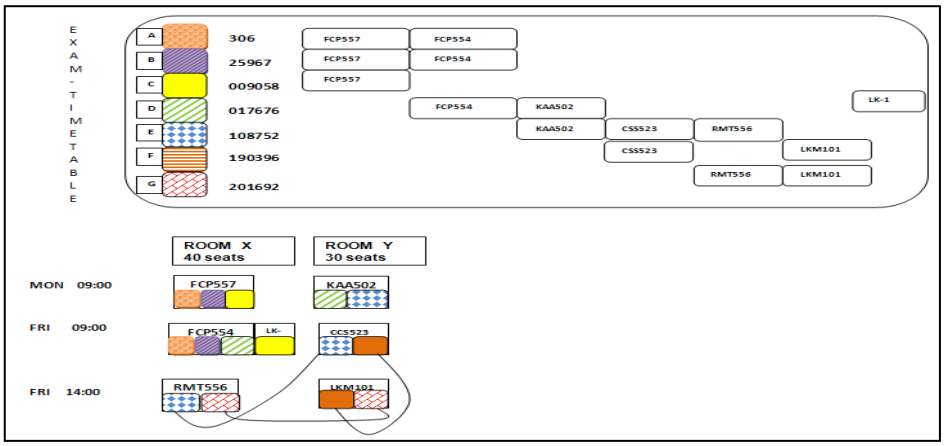


Fig. 1. Assignment of seven examinations to periods (timeslots) to rooms.

The above diagram illustrates an examination schedule in which students take which exams. For- example, 306 (student A) takes FCP557 and FCP554. But 25967 and 009058 also take the FCP557 exam. There are only 3 periods (Monday (AM), Friday (AM) and Friday (PM))) and 2 rooms (X with 40 seats and Y with 30 seats) available.

In a normal heuristic manner the algorithm orders and schedules exams to time periods and rooms based on the difficulty level, but it cannot guarantee the feasibility for all exams. 201692 (student G) has to take the RMT556 and LKM101 exams at the same time. And both 108752 and 201692 aren't too happy because they each have 2 exams on Friday.

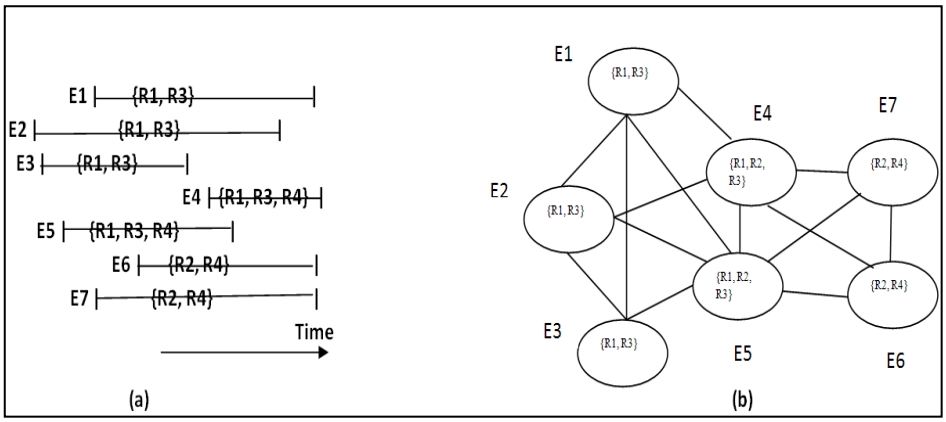


Fig. 2. (a) Examination Schedule of seven Exams with start time and duration during the scheduling phase. For each of the exams possible rooms are shown. (b) Constraint graph on the corresponding schedules.

The examination scheduling problem in Fig. 2(a) can be expressed as a discrete constraint satisfaction problem (CSP). To identify the CSP graph where the nodes are represented as exams to be assigned, Rooms are the resources, and arcs link nodes that intersect in time and indicate that one room cannot be assigned to more than one exam at a time. If it is the case, the constraint graph Fig. 2(b) generally highly connected and often has no feasible solution which can result as conflicts. Examination assignment can be easily mapped with a list coloring problem which is a category in graph coloring. The usual graph coloring problem in interval graph is known to be liner (Gupta et al. 1979) list coloring is NP-complete. (Arkin and Silverberg 1987).

3 Solution Overview

Examination timetabling has considered being NP-complete problem, to generate a solution method it is necessary to develop a structure graph which are more likely to fit the combinatorial problem. We suggest an architecture in which we use three useful techniques: *recapitulate, visual analysis heuristic (VAH)* and s*pecification.*
Recapitulate groups' successive constraints which can be executed by the same resource into a single one, thus reducing the size of the problem. This can be done when you look at the preprocessing of the problem (J.J Thomas et al. 2008).

In general, a *recapitulation* simplifies the problem and it might ignore feasible solutions. To avoid that we first steadily process the original problem to intermediate abstraction levels until confined *recapitulation* can be applied. The tree of splitting which reflect safe recapitulations, illustrated in Fig. 3. It explains the construction of the problem decomposition strategy called VAH (Visual analysis Heuristic). The problem division strategy iteratively decomposes the room allocation problem and expressed as in a CSP, (Constraint Satisfaction Problem) into tree structure consists of interacting sub-components. This clustering process, tests are carried out to check whether a *recapitulation* of the sub-component could yield a feasible solution, if not further refinement is applied. The box in Fig. 3 symbolizes the description of the diagram and its components.

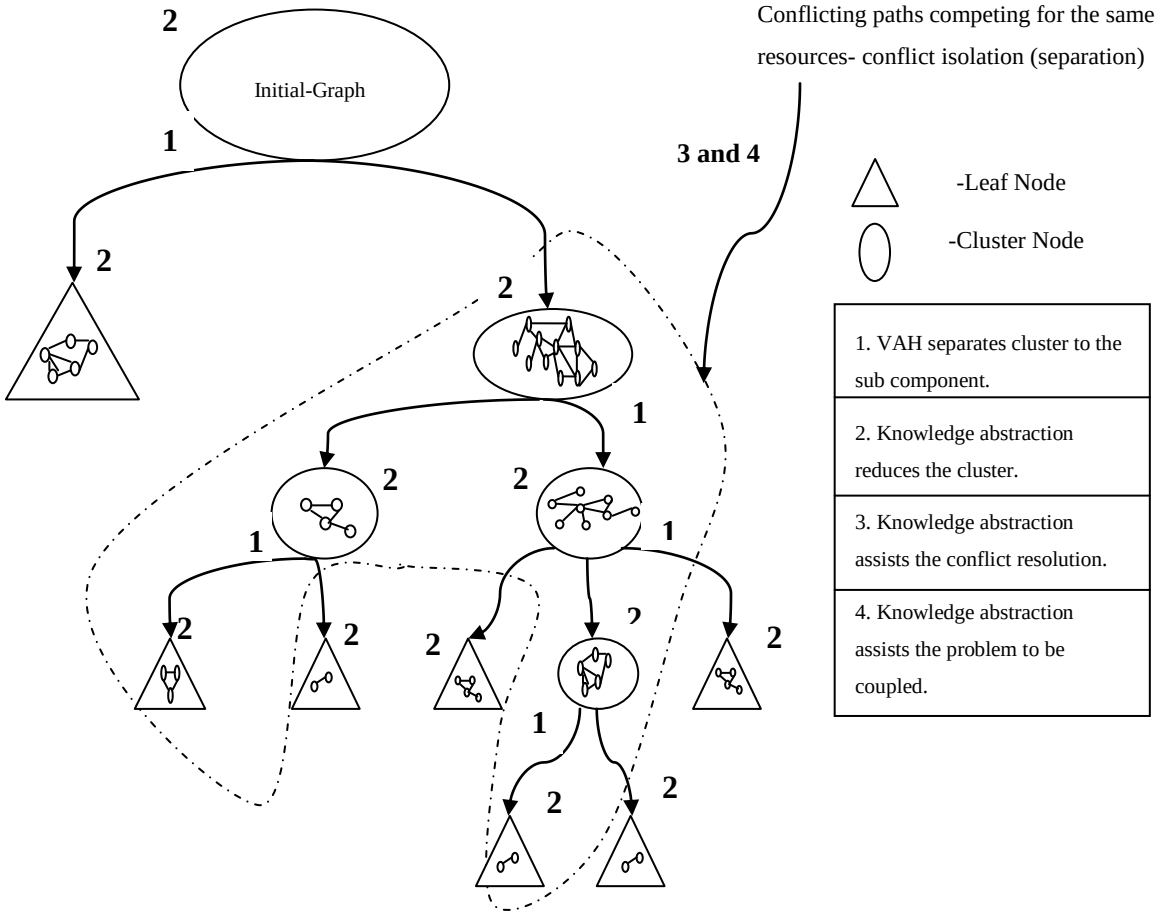


Fig. 3. VAH heuristic and knowledge abstractions in solving examination problem.

At the end of the clustering process, the leaves of the generated tree form isolated components of the initial problem. Some components are under-constrained and can be solved independently, for example room related constraints some other are over-constrained and interact among each other along the tree structure. There are interactions among the conflicts with the resource specific constraints as it treated as resource pools. It is located on the trunk of the tree. As shown in Fig. 3, the detachment strategy provides a framework (J.J. Thomas et al 2008), that is well suited for conflict isolation and for interactive problem solving, in which the human

operator can intervene to select between conflicting paths in the clustering tree. This is very important feature, especially in heuristic application domains, for which preferences are difficult to formulize i.e. semi-automated schedulers.

A second knowledge abstraction technique named *specification* and is based on concept of specification (Dietterich, T.G. Michalski, J.C 1984). The process of specification generates abstractions bottom up along the tree structure in Fig. 3. It starting at the conflicting leaf nodes identified by the conflict resolution procedure. The two-level approach can easily extend into multiple levels to give constraint hierarchies (Boring et al. 1987). The specification procedure forms the most specific common specifications of the conflicting assignment. This is used as a feedback support for the visual analysis heuristics and evaluation of conflicts and problem solidity.

4 Knowledge Abstraction

Abstraction techniques have been proposed as promising methods to reduce the complexity of the problem solving and have been applied to large number of domains. In the examination timetabling problem, multi-user distributed environment with various cohorts of schools and department who often operate quite autonomously. It has been studied (Dimopoulou et al. 2001, Dietterich et al. 1984, Dean et al. 1987, Doyle 1987), much more work is required on understanding the issues involved and the interplay between user interaction and managing the information with the goal of producing a workable solution and the extent to which techniques can be used in an automated process.

Defining the abstraction is as follows: (Golumbic 1984)

Abstraction is the mapping of a problem representation into a simple one that satisfies some desirable properties in order to reduce the complexity of reasoning. The problem is solved in a simplified abstract space and the solution is then mapped back to the complex ground space.

Abstraction techniques shows potential to reduce the complex problem, and it have been applied to wide number of domains. We suggest two processes of knowledge abstractions *recapitulate* and *specification.* Recapitulations group the

knowledge space, and are considering only time related information (periods), while specifications group classes of similar within one proposition to form a more general solution. These methods are independent of adopted sequential formalism, we highlights the Time Map Manager (Dietterich et al 1984). The unit of TMM is sequential tokens, and it is associated with the event proposition of a time interval. In general abstractions are rough calculation and might lose little information related with the detailed descriptions of sequential token (constraints). We organize the tokens by using the arrangement used in TMS (Truth Maintenance System) (Dean et al. 1987) the user has access to the sequential tokens (constraints) wherever it is necessary.

4.1 Recapitulate

Recapitulation is the substitute of a compilation of time tokens. The interval of the recapitulations is the smallest period that includes all intervals of the component tokens. The property of the recapitulation is a combination of the properties of the essential tokens, see Fig..4

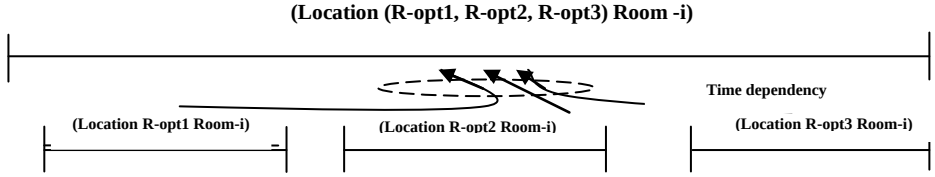


Fig. 4. Recapitulation of interval resource based constraints.

In the examination timetabling the allocation of rooms to exams are recapitulate such as opt1 takes place in room-1 followed by opt2 and opt3 in the same room (Institutional based soft constraint). All the examination must assigned to rooms within the time period (interval) during the exam week. The solution can be of constituent priorities. The priority does not necessary hold for every sub-interval. Some exams may need specific rooms, only exams of similar lengths are scheduled at the same timeslot in the same room.

After recapitulation, the abstract space comprises a subset of the initial set of possible solutions and any solution found in the abstract space can be safely mapped

back to a concrete solution. In Fig.. 4 the assignment have been grouped to the same room, for alternative solutions. For example, assigning a different room for exams has been purposely ruled out and dropped from the solution space.

4.2 Specification

Specification is base on domain, depends on background knowledge in the form of concept hierarchy. A time table designer or administrator who has good experience in the hierarchy. For example an examination timetable designer must aware of what are the subjects are currently in the semester. To avoid clashes in exams, the basic knowledge is to allocate exams first for those who register for particular subjects in the current exams should be of fewer conflicts. Specification operates only on its propositional expression or property it provides a way to replace a disjunction of terms within one proposition, by a single more general term (Dimopoulou et al. 2001).

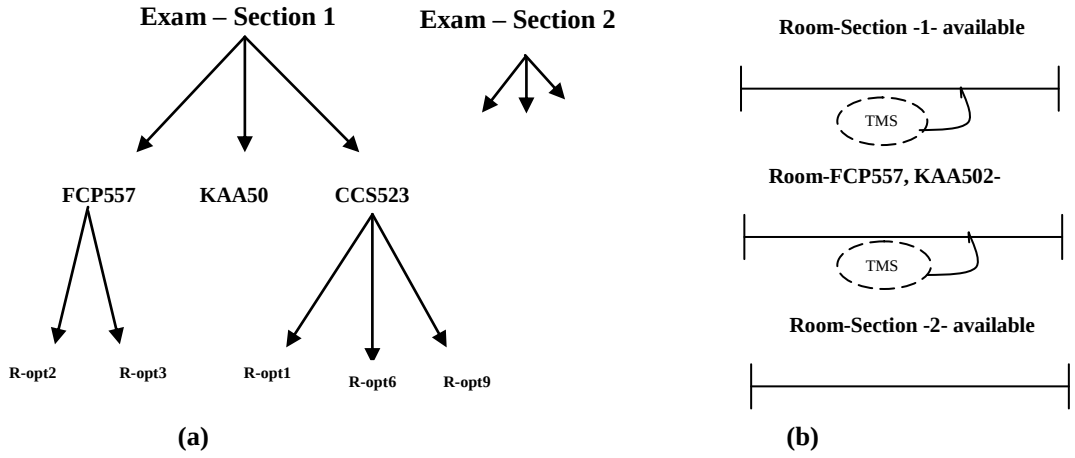


Fig. 5. (a) & (b) Specification using background knowledge structure

give another example of specifying a proposition obtained by recapitulation using specification hierarchy shown. For instance, the timetable designer recapitulates exam section -1, exam section-2 and so on and allocate to the timeslots (periods) into rooms with less conflicts. Specifications suppress the detailed descriptions of specific entities by mapping them to more generalized categories. Thus decisions made on the

basis of the general categories alone might violate constraints that refer to the specific entities.

5 The use and computation of Recapitulate

The goal is to simplify the problem by reducing the number of assignment and thus the size of the search space, at the expense of reduced flexibility. In here, we discuss how recapitulation are applies in examination allocation. Each leaf cluster identified by the VAH is a set of assignments with similar room requirements. Recapitulations are applied to the leaf clusters in order to reduce their size. A leaf cluster "contains" a collection of intervals such as those shown in Fig. 6 (a). Groups of intervals which do not overlap are identified and replaced by exclusive interval. A regular allocation procedure can finally apply to assign one value per "time".

The first step is to arrange the time interval, by topological sorting, in the directed graph reflecting their knowledge abstraction succession Fig. 6 (b). Nodes denote time internals, links are built between nodes that do not overlap, and arrows reflect their precedence order. Every path in this graph represents a possible sequence of assignment that can be carried out by one and the same room, and thus a possibly useful recapitulation.

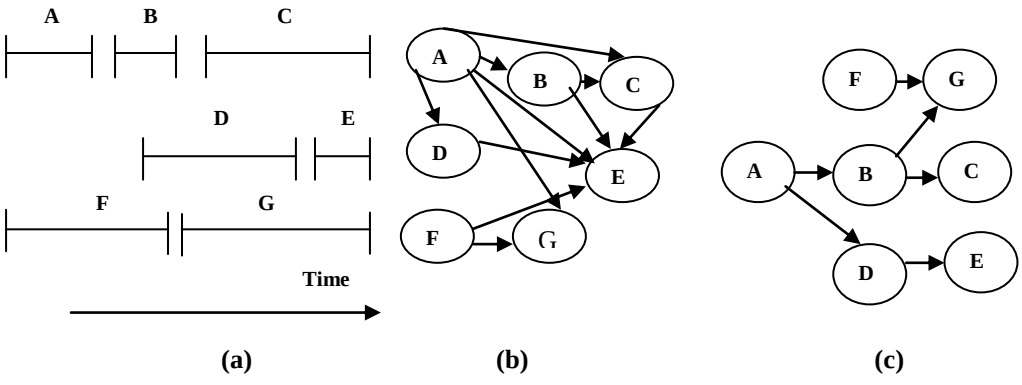


Fig. 6. (a) Token representation of timeslot located in one cluster. (b): graph comparability. (c): After eliminating arcs of the transitive priority.

The user may control the demanding constraints maximum, interval between correlated assignment and the arcs that exceed this interval are omitted from the graph. Competing recapitulations are apparent in the graph as different paths between the same nodes. Those that arise as a result of transitive priority are eliminated to simplify the graph.

If the users have some unquantifiable criteria (subjective preferences) for grouping assignments, they can be presented with the graph in Fig. 6 (c) so, they can interactively select the most suitable recapitulations. They can thus apply criteria which are difficult to formalize in a computer program, and obtain solutions which are more acceptable in practice.

The algorithm is to replace the set of intervals by the set recapitulated intervals of minimum cardinality. This can be done by coloring the intervals with the minimum number of colors then recapitulate those intervals with the same color. We consider the two criteria for establishing this order.

1. maximize the minimum distance between consecutive timeslot (allocating the exams, the length of the timetable)

2. minimize the maximum distance (the total number of students in the same room must be less than the capacity of the room).

6 Visual Analysis Heuristic (VAH)

This section proposes a new heuristic called the Visual Analysis Heuristic (VAH) useful for solving list coloring problems expressed in Section 2 Fig 2(a) & (b). Here we are providing a node value called *delay*. This means that nodes linked cannot be given the same value. In interval graphs, two nodes are linked if and only if their time interval intersects and they have at least one value (i.e. resource) in common. Since, in real-life examination applications, many constraints are to be executed at the same time and many rooms are seek by many time slots, the corresponding constraint graphs tend to be densely connected.

The clustering process has happened in the nodes shared most common values in graph the elimination of arcs are applied to the constraint graph and step by

step separating it to clusters. We demonstrate it with a simple illustration. In Fig. 7 (top), we show simple example of a list coloring problem with four nodes: N1, N2, N3 and N4. Each node has a set of values it can be assigned. It indicates the values assigned to the nodes are not the same. The backtracking procedure applies to the significant nodes and using iterations with respective of delay values to initial cluster and to the generated cluster in the bottom of Fig. 7.

Using the value-delay heuristic, the most common value, is delayed, thus splitting the initial cluster into two parts. In Cluster- 1, one can assign value {e} to N2 without hesitation. In Cluster2, the three nodes are still competing for common values, and they share, as a reserve, a set of delayed values, delay ={a}. At the next step, value {b} is delayed, and unsolved is split further, into two parts. One of these, Cluster3, can be assigned a value without further delay, and the remaining cluster is left with an empty set of possible values. It claims the delayed values “a” and “b”. A subsequent conflict resolution procedure assigns one delayed value to each unsolved node in this cluster.

The VAH heuristic provides a visually relevant, dynamically built, hierarchical structure to evaluate and it is an abstraction technique.

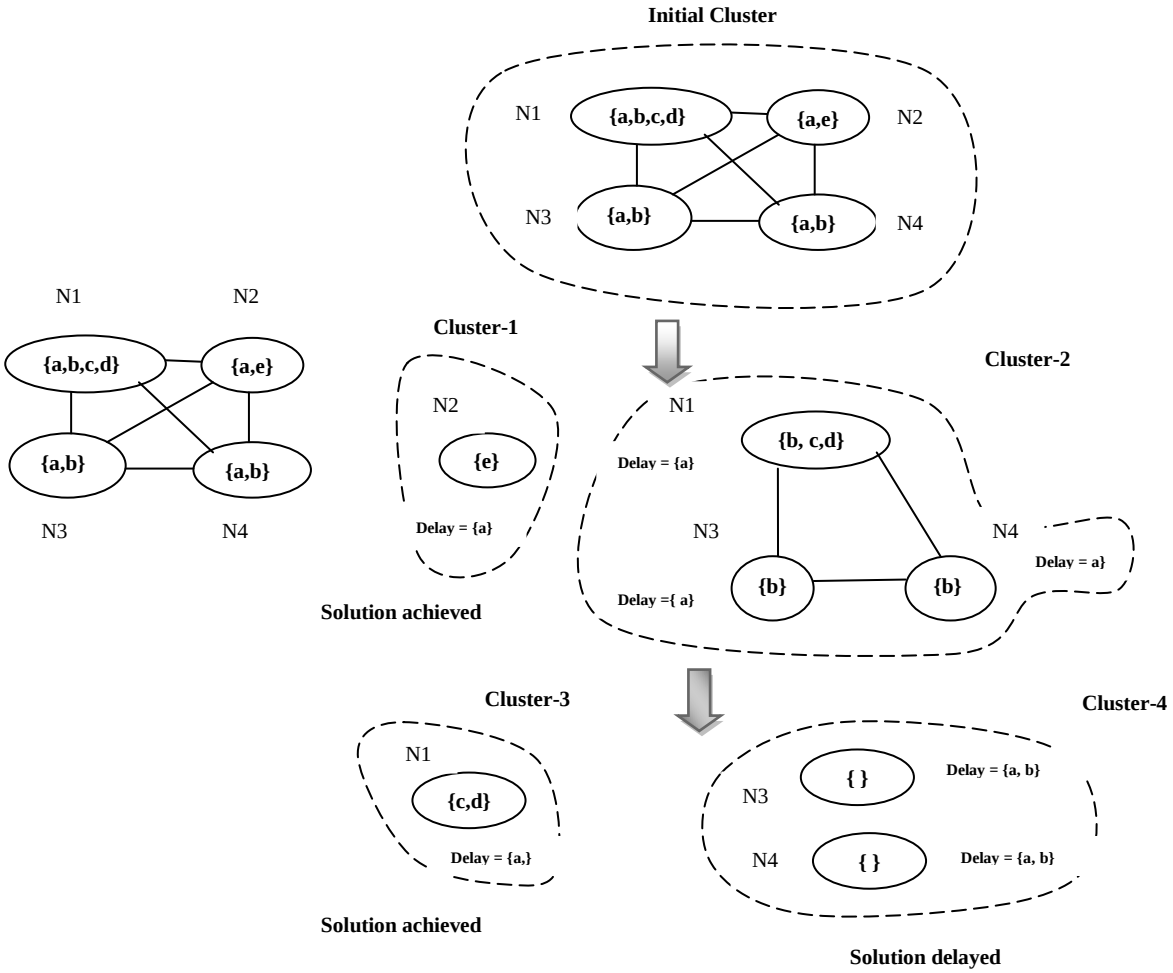


Fig. 7. (a) List coloring problem (b): Applying Visual Analysis Heuristic to example

7 Conflict detection

Fig. 8 has three clusters namely Cluster 1, 3 and 4. Each cluster maintains values {a} and {b} and it can be solved independently. The root node has a delayed value {a} with two levels, the first level cluster (leaf cluster-1) are solved. Cluser-4 is claiming values {a} and {b} and the solution is delayed (not solved). Each set of delayed values maintain by one or more unsolved cluster called *conflict.*

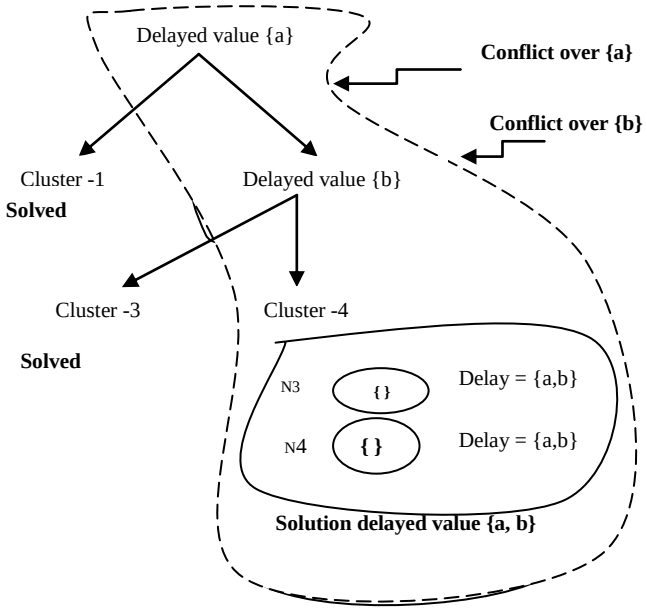


Fig. 8. Conflict Separation

The detection of conflict has been achieved by testing all conflicts by a conflict detection procedure. The conflicts are visually available for the timetable designers if it has preferences he or she can interact with the conflict resolution and modify the value assignment.

8 The Wide Exam Result and its resolution

The heuristics (or constructive) approaches are often stemming from a graph coloring heuristics. The basic timetabling problems can be modeled as graph coloring problem and the above section explains about the list coloring problem is a part of graph coloring problem. The conflict procedure clusters of a given branch node in the hierarchy tree might compete for values delayed however, the time interval does not interconnect. This will happen to all the clusters have competing same delayed values. We call this procedure as *wide exam result* it covers all instances of the clustering procedure.

When the exams in two apparently competing clusters, do not all intersect in time the two clusters defined a possible *association*. They can be merged and the non-intersecting exams can be recapitulated thus reducing the overall space contention.

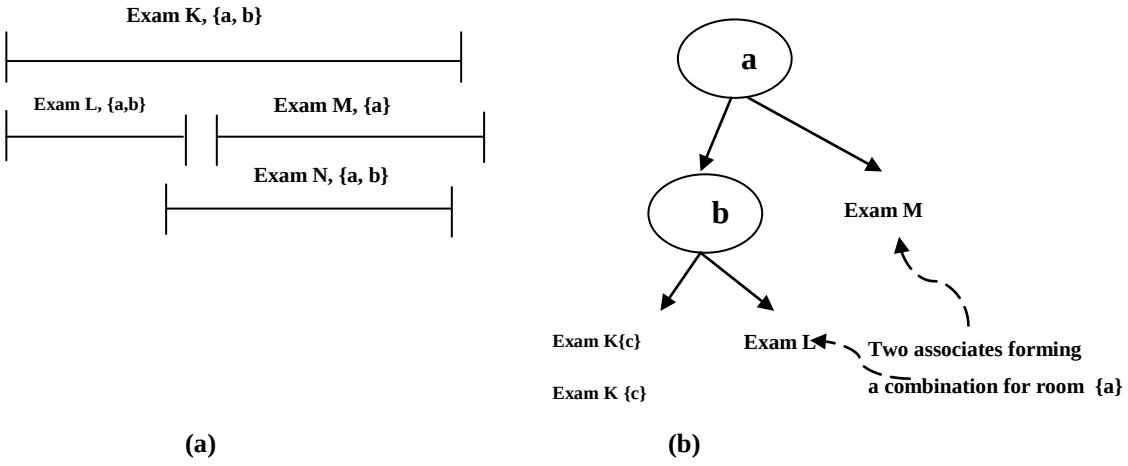


Fig. 9. (a): Four timeslots and their possible examinations. (b): illustration of Wide Exam Result method.

The example of Fig. 9 (a) is a very simple illustration of the wide exam result effect. Exam K has possible association with rooms {a, b} and Exam L with {a,b}, Exam M {a} and Exam N with{a,b} respectively. Fig. 9 (b) value {a} and {b} are the *association* forming the combination for room {a}. One way to overcome the wide exam result is to find possible grouping among leaf clusters. Upon user's request, the

conflict detection procedure identifies all grouping and measures the corresponding decrease of exam conflict.

9 Automatic conflict resolution

Although our goal is to allow the users to selectively participate in the conflict resolution by viewing and manipulating conflicts and *associations* determined in the previous steps, we also provide an automatic conflict resolution procedure. This procedure may integrate domain dependent knowledge and adopt any of the following strategies:

9.1 Conflict resolution procedure

1. Allocate a delayed value to the first node encountered going in the reverse order of that used to create the tree (i.e. from the last visited leaf up to the root).
2. When two exams are competing for a delayed value, give it to the exam that temporally contains the other, or the one with the longest duration.
3. Allocate a delayed value to the exams that participate in the least number of associations.
4. Use domain dependent heuristic knowledge to distribute delayed resources over unsolved clusters. For instance, one may want to allocate a room to those exams that show closeness or distances.

These strategies are implemented with visual representation which will be depicted in Fig.10. Further investigation into conflict resolution is necessary. The conflict resolution procedure exhibits exponential complexity. However, since parts of the problem may have been solved by the previous procedure.

10 Real-world Examination Timetabling Problem: an example

In this section, we propose an *interactive visual model* for solving real-life examination timetabling problems that integrates the techniques introduced and discussed above. This solution method is made of three components, namely:

(1) a clustering algorithm based on the composition of the VAH heuristic and of knowledge abstraction.

(2) a conflict resolution procedure where conflicting needs of groups of tasks for the delayed values are identified and, either interactively or heuristically, solved, and

(3) a specification procedure carried out over unsatisfied constraints. This last procedure provides an assessment of problem rigidity. Here, we apply the solution method to a real-life examination timetabling as a case study: The allocation of examination to rooms based on standard constraints.

In this section, we demonstrates from Fig. 10 to 12, how a solution could be completed successfully. Firstly, a node consistency check is run to determine the rooms that can be allocated to each examination. Then the initial constraint graph is built and separated into independent connected graphs. Nodes that are not linked to the rest of the graph are isolated. Here we use three variables rooms, period and exams to visualize the exam to rooms and to period's conflicts and allocation. Fig. 10 has shown the possible conflicts, between rooms, examination and timeslot (periods).

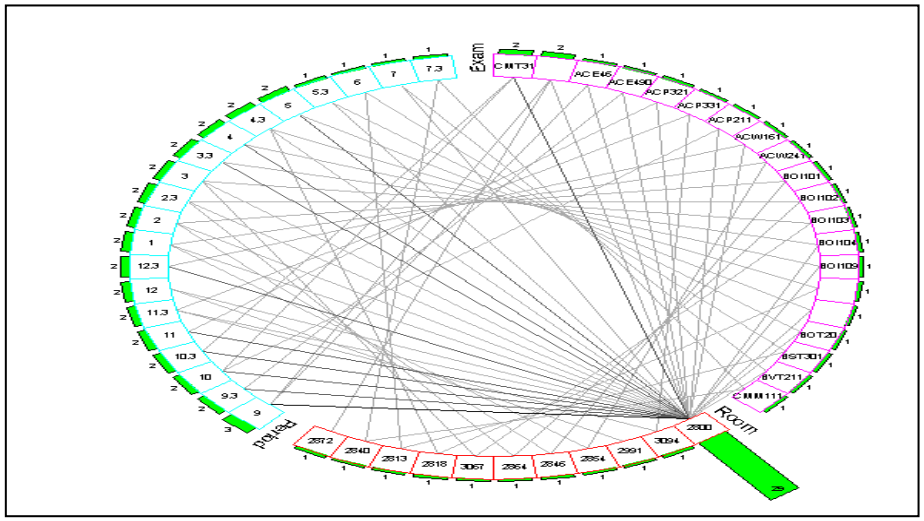


Fig. 10. The constraint graph of examination clashes between the exams to rooms and period.

The clustering procedure is applied to each of the remaining components. It uses the VAH heuristic, recapitulation; (Cheeseman et al. 1991)-Fig. 11 illustrated the initial problem discussed as in Fig. 3, the color symbolized the clustering groups and links between each examination. This visualization assists the problem to separate into two levels in later stage to apply the visual analysis heuristics techniques together with evolutionary algorithms.

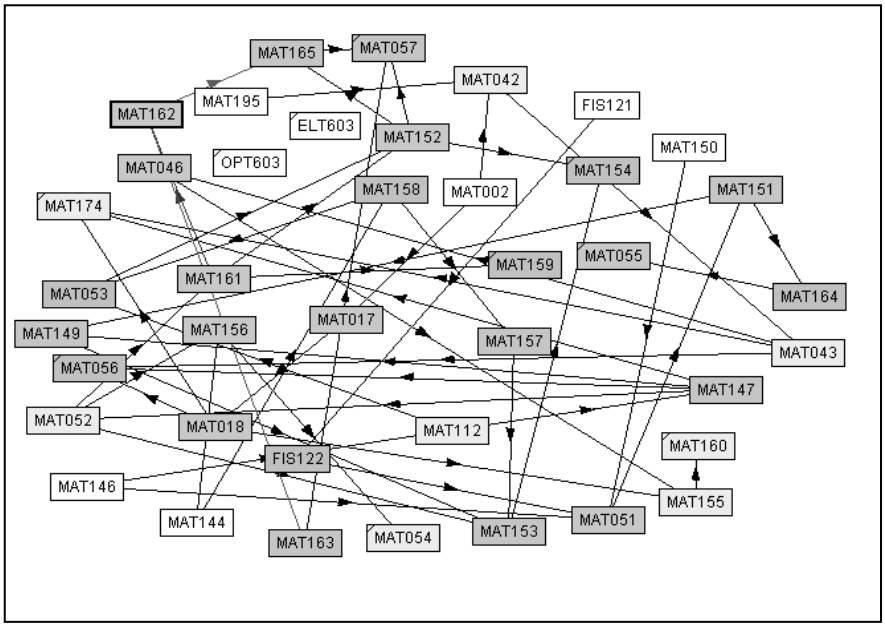


Fig. 11. Sample dataset color coded based on the exams clashes in a particular school.

The remaining in a leaf cluster (OPT603, ELT603) is solicited equally by the allocation of room in the cluster. The recapitulation process described is Section 4.1 is applied to these assignments to reduce the size of the corresponding leaf cluster. Fig. 12 shows the tree after the leaf clusters have been reduced by recapitulation; hence the clusters in this diagram are smaller than the corresponding ones in Fig. 11.

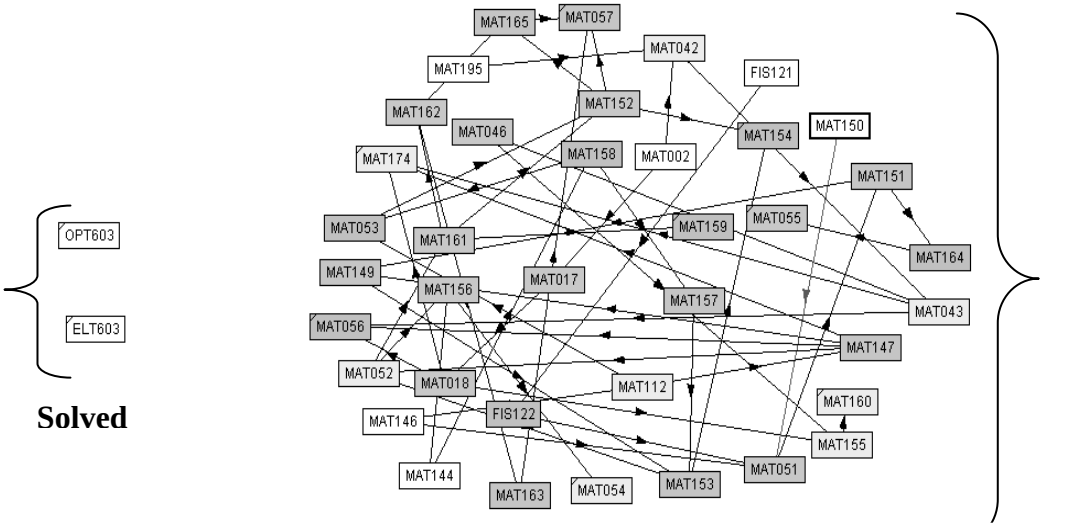


Fig. 12. Leaf nodes have been recapitulated and conflicts detected (Decomposition by VAH).

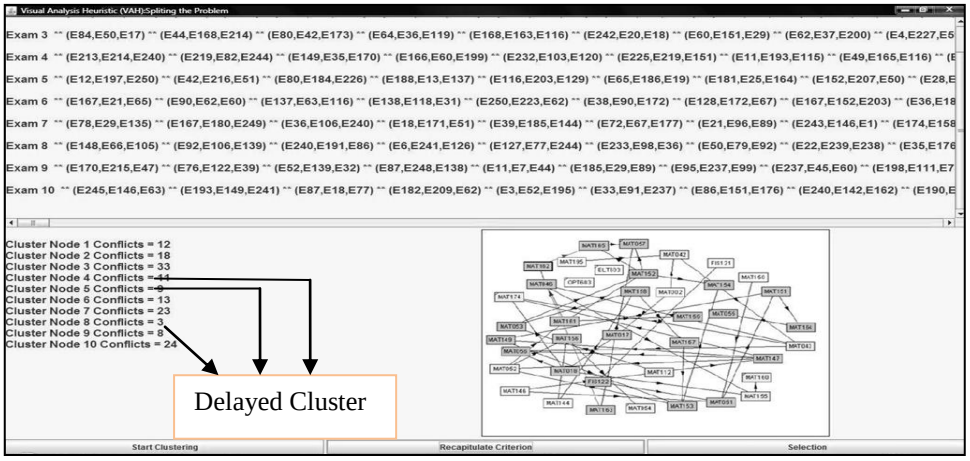


Fig. 13. Assignment tasks delayed values (Cluster 5, Cluster 8 and Cluster 4) using evolutionary algorithm

At this case, the user can see the current conflicts based on the delayed value assigned. For example here the delayed examination clusters are 5, 8, 4. The user may interact in the resolution of any conflict in allocation of examination to rooms.

However the solver is enable from the human scheduler by formation procedures and identifies the groups between the clusters claimed as delayed.

In Fig. 14. Visual Analysis Heuristic (VAH) solution split the problem into easily solved cluster and difficult solved clusters. By the method of *grouping* procedure identifies all possible clusters claiming the same delayed assignments. Finally, in Fig..14, the automatic conflict resolution succeeded in allocating exams to the rooms and all the grouped clusters are assigned with two assignments. E.g. Room No (1) assigns to Exam No (172), (107) same room used for two different examination in different period.

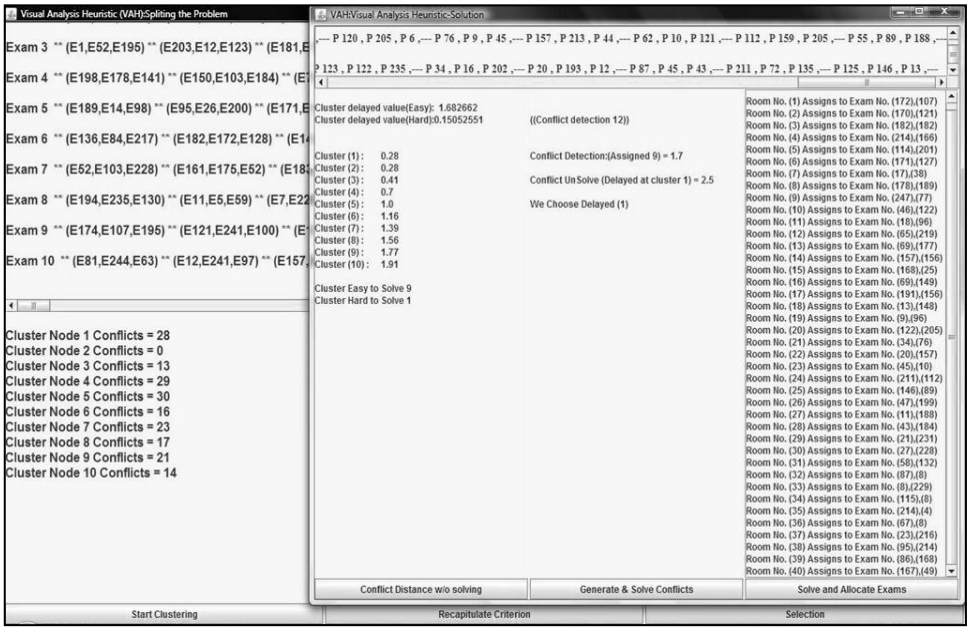


Fig. 14. Examination Assignment to rooms has been heuristically solved.

11 Conclusion and Discussion

We have presented a new interactive approach of the examination timetabling problem as it would appeal to many institutions. In providing this formulation, it is pointed out that minimizing the number of periods is not considered to be an effective way to solve the real-world examination timetabling problem. The discussion below is fruitful on helping the issues.

Several approaches have been used for solving the examination timetabling problems are based on variations to build an automatic solver. Many on backtrack search algorithms, mechanism are effectively implemented by underline the risks of combinatorial explorations. Majority of the times, it focuses on algorithm efficiency and the CPU time to solve the problem is discussed, importantly they fail to explain the reasons for the failure when it occurs.

Few remarks that have to be including in this paper, the proposed approach together with algorithms are creating a new breakthrough on the timetabling research. Interactive techniques and visual perception are work together to solve combinatorial problems with minimal violation of constraints especially to solve real-world timetabling application in general.

There are few approaches are highly suitable for interactive application *concept formation* rather than just writing on implementation with results. The is one of the good and advantages method is the reason provide to the reader it give visual pertinent hierarchical tree structures that give the user a closer walkthrough on the assignment of resources (real-world applications) and , through *recapitulation* and *specification* over conflict sets. It might assist the user to generate knowledge based abstraction for rescheduling.

Visual Analysis Heuristic (VAH) is assisted with delayed value to the clusters. Each group of clustering are formed is based on certain constraints (soft constraints) in real-world examination timetabling and with institutional model (Cumming et al. 2006) it is necessary to analyze the constraints and split to work towards a feasible solution. This technique is used for all type of list coloring problem which is a category of graph coloring problem always mapped to examination timetabling problem.

In many constraint based systems, contradictions are solved by relaxing one or more constraints choosing the minimum set of constraints known to be NP-Complete. But when we describe about feasibility and optimality, primarily the interested area is feasible solutions. Optimality is considered highly complex once the problem is in real-world environment (real-world examination timetabling). The clustering method we proposed will find a better near-optimal solution because we have avoided the situation where a few local minima are numerically compensated.

The prototype application has to be integrated to a single GUI and that will provide the user a greater flexibility. Furthermore, the benchmark dataset are to be tested to understand the proposed heuristics.

12 References

Al-Betar, Mohd., Khader, A.T., Gani, T,A.: A Harmony Search Algorithm for University Course Timetabling. In: Burke, E., Michel Gendreau (eds) Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling. Université de Montréal, August 18 - 22, 2008.

Arkin, E. Silverberg, E. Scheduling Jobs with Fixed Start and End Times. In.: Discrete Applied Mathematics, Vol 18, pp. 1-8(1987)

Burke, E.K., Kingston, J., de Werra, D.: In: Gross, J., Yellen, J.(eds) Applications to timetabling. Handbook of Graph Theory, pp. 445-474. Chapman Hall/CRC Press (2004)

Burke. E. K., Newall, J.P., Weare, R.F.: A Memetic Algorithm for University Exam Timetabling. In: Burke, E.K., Ross, P. (eds) Practice and Theory of Automated Timetabling. LNCS, vol. 1153, pp. 3-21. Springer, Heidelberg(1996)

Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristics for educational timetabling problems. European Journal of Operational Research 176, 177-192(2007)

Burke, E.K., Marecek, Parkes, Andrew J. Rudova, Hana. Decomposition, reformulation, and diving in university course timetabling. Computers & Operations Research, Amsterdam, Elsevier, The Nederlands. ISSN 0305-0548, 2010, vol. 37, no. 3, pp. 582-597.(2010)

Benson, B. W. and Freuder, E.C. Interchangeability Preprocessing Can Improve Forward Checking Search. In.: Proceedings of the 10th ECAI, pp. 28-30, Vienna, Austria.

Berry, P.M. A Predictive Model for Satisfying Conflicting Objectives in Scheduling Problems. PhD thesis, Department of Computer Science, University of Strathclyde, Glasgow, UK.

Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, A., Woolf, M. Constraint Hierarchies. In.: Proceedings of the conference on Object-Oriented Programming Systems, OOPSLA-87, pp. 48-60, Languages and Applications, Orlando, FL.

Bose, P. An Abstraction-based Search and Learning Approach for Effective Scheduling. In.: Famili, A., Nau, D.S., Kim, S.H (eds), Artificial Intelligence Applications to Manufacturing, pp. 187-197. AAAI-Press/The MIT-Press, Menlo Park, California.

Cheeseman, P. Kanefsky, B. Taylor, W.M. Where the Really Hard Problems Are. In.: Proceedings of the 12th IJCAI, pp. 331-337 Sydney, Australia(1991)

Cote, P.: A hyprid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. In: Burke, E.K., Trick, M.A.(eds) PATAT 2004. LNCS, vol. 3616, pp. 294-312.Springer, Heidelberg(2005)

Cumming, A. Paechter, B. Rankin, R.C. Post-Publication Timetabling, In: Proceedings of the 3rd International conference on the Practice and Theory of Automated timetabling, pp. 107-108 (2006).

Chiarandini M, Birattari M, Socha K, Rossi-Doria O. (2006) An effective hybrid algorithm for university course timetabling. J. of Scheduling 1094-6136 9: 403-432.

Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E., Erben, W.(eds) PATAT 2000. LNCS, vol. 2079, pp. 104-117. Springer, Heidelberg (2001).

Dimopoulou, M. Miliotis, P. Implementing a University Course and Examination Timetabling System in a Distributed Environment, In: Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling. pp 148-151(2001).

Dietterich, T.G. Michalski, J.C. A Comparative Review of Selected Method for Learning from Examples. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M.,(eds) Machine Learning: An Artificial Intelligence Approach, pp. 41-81. Springer, Berlin, Heidelberg(1984)

Dean, T. McDermott, D. Temporal Data Base Management. Artificial Intelligence, Vol. 32, pp. 1-55 (1987)

Doyle, J. A Truth Maintenance System. Artificial Intelligence, Vol. 12, pp. 231-272 Morgan Kauffman San Francisco, CA, USA (1979)

Erben, W,: A grouping genetic algorithm for graph coloring and exam timetabling. In: Burke, E., Erben, W. (eds) PATAT 2000. LNCS, vol. 2079, pp. 132-158. Springer, Heidelberg(2001)

Eley, M.: Ant algorithm for the exam timetabling problem. In: Burke, E.K., Rudova, E.K.(eds) Practice and Theory of Automated Timetabling LNCS, vol. 1153, pp. 167-180. Springer, Heidelberg (1996)

Ellman, T. Abstraction via Approximate Symmetry. In.: Proceeding of the 13th IJCAI, International Joint Conference on Artificial Intelligence, pp.916-921, Chambery, France(1993)

Ellman, T. Synthesis of Abstraction Hierarchies for Constraint Satisfaction by Clustering Approximately Equivalent objects. In.: Tenth International Conference on Machine Learning, Amherst, MA (1993)

Feldman, R. Golumbic, M.C. Optimization Algorithms for Student Scheduling via Constraint Satisfiablilty. The Computer Journal, 3 No. 4:356-364.

Fox, M.S, Smith, S.F. ISIS: A Knowledge-Based System for Factory Scheduling. In.: Allen, J.F. Hendler, J. Tate, A (eds) Readings in Planning , pp. 336-360. Kaufmann, San Mateo, CA.

Gupta, U.J. Lee, D.T. Leung, J.Y. An Optimal Solution for the Channel Assignment Problem. IEEE Transactions on Computers, C 28 (11): 807-810 (1979)

Golumbic, M.C. Algorithmic Aspects of Perfect Graphs. In.: Annals of Discrete Mathematics. Vol 21, pp. 301-323(1984)

Giunchiglia, F. Walsh, T. A Theory of Abstraction. Technical Report IRST 9001-14, Istituto per la Ricerca Scientifica e Tecnologica, University of Trento, Trento, Italy (1990b).

Hart, E.Ross, P. Gavel - a new tool for genetic algorithm visualizationIn.: IEEE Transactions on Evolutionary Computing, Vol. 5, pp. 335-348.

Haselbock, A. Exploiting Interchangeabilities in Constraint Satisfaction Problems. In.: proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI, pp 282-287, Chambery France (1993)

J.J Thomas, Khader, A.T. Belaton, B. A Visual Analytics Framework for the Examination Timetabling Problem. In.: Proceeding of the 5th International Conference on Computer Graphics, Imaging and Visualization CGIV08, pp. 305-310, August 26-28 (2008).

J.J. Thomas, Khader, A.T. Belaton, B. Information Visualization Approach on the University Examination Timetabling Problem. In.: Book Visual Information Communication ISBN 978-1-4419-0311-2 pp. 255-264. Springer USA (2009).

Lewis R, Paechter B. (2005a) An empirical analysis of the grouping genetic algorithm: the timetabling case Evolutionary Computation, 2005. The 2005 IEEE Congress on, pp. 2856-2863 Vol. 2853.

Lewis R, Paechter B. (2005b) Application of the Grouping Genetic Algorithm to University Course Timetabling Evolutionary Computation in Combinatorial Optimization, pp. 144-153.

Lewis R, Paechter B, Rossi-Doria O. (2007) Metaheuristics for University Course Timetabling Evolutionary Scheduling, pp. 237-272.

McCollum B. (2007) A Perspective on Bridging the Gap Between Theory and Practice in University Timetabling Practice and Theory of Automated Timetabling VI, pp. 3-23.

McCollum. B, P.McMullan, E.K.Burke, A.J.Parkes, R.Qu, A New Model for Automated Examination Timetabling . In. Endre Boros (eds) Annals of Operational Research, 2008-accepted.

McCollum, B.: University timetabling: Bridging the gap between research and practice. In: Burke, E.K., Rudova, H (eds): Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling. 30th August – 1st September 2006, Brno, Czeh Republic, pp. 15-35 (2006)

McCollum B, Schaerf A, Paechter B, et al. (2009) Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS JOURNAL ON COMPUTING: ijoc.1090.0320.

Muller, T, Barak, R. Interactive Timetabling: Concepts, Techiniques and Practical Results, In: Proceedings of the 4th International Conference on the Practice and Theory of Automated timetabling, pp. 58-72.

Ow, P.S. Smith, S.F. Viewing Scheduling as an Opportunistic Problem-Solving Process, Annals of Operational Research, vol 12, pp 85-108. (1988)

Piechowiak, A. Ma, J. Mandiau, R. An Open Interactive timetabling Tool, In: Burke, E.K. Trick, M. (eds) selected papers from the 5th International conference, Pittsburgh 2004, Springer Lecture Notes in Computer Science, Vol 3616, Springer 2005.

Qu, R, Burke, E., McCollum, B., Merlot, L.T.G Lee, S.Y.: A survey of search methodologies and automated approaches for examination timetabling. Technical Report No. NOTTCS-TR-2006-4, School of Computer Science & IT, University of Nottingham (2006)

Rubio, R.G. A Timetable Production System Architecture for Course and Exams. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pp. 342-350.

A 5.875-Approximation for the Traveling Tournament Problem

Stephan Westphal · **Karl Noparlik**

Abstract In this paper we propose an approximation for the Traveling Tournament Problem which is the problem of designing a schedule for a sports league consisting of a set of teams T such that the total traveling costs of the teams are minimized. Thereby, it is not allowed for any team to have more than k home-games or k away-games in a row. We propose an algorithm which approximates the optimal solution by a factor of $2 + 2k/n + k/(n-1) + 3/n + 3/(2 \cdot k)$ which is not more than 5.875 for any choice of $k \geq 4$ and $n \geq 6$. This is the first constant factor approximation for $k > 3$.

Keywords Sports Scheduling · Traveling Tournament Problem · Approximation Algorithms

1 Introduction

During the last decades professional sports leagues worldwide have turned into million or sometimes even billion dollar businesses. Soccer in Europe as well as American Football, basketball, baseball or ice hockey in North America absorb thousands of fans inside the stadiums and millions of spectators around the world. A crucial contribution to the success of a season lies in the timetable or schedule of the league which determines what games are arranged when and at which arenas. Thereby, the planers of those leagues have to balance not only the expectations of the fans but also many requests stipulated by clubs and TV stations. Created by hand in the past, nowadays most schedules of professional sports leagues are obtained by computer-based applications of sophisticated mathematical models and tools.

In this paper we will focus on the Traveling Tournament Problem (TTP) introduced by Easton et al. [6]. It is a quite well-known and practically difficult optimization problem inspired by Major League Baseball. North American sports leagues have an incentive to minimize the travel distance of the participants of a tournament due to the vast expanse of their continent.

The task of the TTP is to find a schedule for a double round robin tournament (where each team plays every other team twice: once at its home venue and once at the other team's

University of Kaiserslautern, Department of Mathematics, P.O.Box 3049, Paul-Ehrlich-Str. 14, 67653 Kaiserslautern, Germany.
{westphal, noparlik}@mathematik.uni-kl.de

venue) which minimizes the overall travel distance of all teams in a sports league under two specific constraints.

These constraints are the no-repeater constraint, enforcing that game A-B (B travels to A's venue) must not be placed directly after game B-A took place and the restriction on the number of consecutive home games (*home stands*) and also on the away games (*road trips*). This is due to economical reasons since the supporters might be bored by a too long home stand a well as impatient during a long road trip.

1.1 Sports Scheduling and the Traveling Tournament Problem

Sports Scheduling in general deals with the design of tournaments. A *single round robin tournament* on n teams where n is an even number consists of $(n-1)$ days (also called *slots*). Each day $n/2$ games which are themselves ordered pairs of teams take place. Every team has to participate at one game per day and must meet every other team exactly once. It is standard to assume n to be even since in sports leagues with n being odd, a dummy team is usually introduced and whoever plays it has a day off, which is called a *bye*. A *double round robin tournament* on n teams consists of $2(n-1)$ days and every team must meet every other team twice: once at its own home venue (*home game*) and once at the other team's venue (*away game*). A popular policy in practice is to obtain a double round robin tournament from a single round robin tournament by mirroring, that is repeating the matches of day k for $k = 1,...,n-1$ on day $k+n-1$ with changed home field advantage. Consecutive home games are called *home stand* and consecutive away games form a *road trip*. The *length* of a home stand or road trip is the number of opponents played (and not the distance traveled).

The *Traveling Tournament Problem (TTP)* as introduced in [7] is then defined as follows:

Input:

- a set $V = \{1,2,\ldots,n\}$ of n teams with n even
- an $n \times n$ integer distance matrix D containing the metric travel distances between the home venues of all teams
- integers L,k

Output: A double round robin tournament on V satisfying:

- The length of every home stand and road trip is between L and k inclusive.
- No pair of teams plays both of their matches against each other on two successive time slots.
- The total distance traveled by all teams is minimized

In this paper, we assume that $L = 1$ which is common in literature and means that we forget about L. This assumption is reasonable since it is hard to imagine a sports league planner who will insist on forbidding home stands or road trips of length 1 when facing his many conflictive objectives.

1.2 Previous Work

So far, most efforts concerning the TTP have led to a variety of algorithms aiming to minimize the total distance driven by the teams. Kendall et al. [9] provide a good overview of the work done on the TTP and sports scheduling in general. Just to mention a very few examples, hybrid algorithms with constraint programming (CP) exist by Benoist et al. [3] who additionally use Lagrange relaxation. Easton et al. [7] merge CP with integer programming while Henz [8] combines CP with large neighborhood search. Anagnostopoulos et al. [1] and Hentenryck and Vergados [14] propose simulated annealing algorithms, whereas Ribeiro and Urrutia [12] focus on the special class of constant distance TTP where break maximization is equivalent to travel distance minimization.

The TTP is believed to be *NP*-hard although to the best of our knowledge no proof has been published yet. For scheduling single round robin tournaments a rather general and useful scheme called *canonical schedule* has been known in sports scheduling literature for at last 30 years [5]. One can think of the canonical schedule as a long table at which n players sit such that $n/2$ players on one side face the other players seated on the other side of the table. Every player plays a match against the person seated directly across the table. The next day of the schedule is obtained when everyone moves one chair to the right with the crucial exception that there exists one person at the end of the table who never moves and always maintains the seat from his or her first day. Note that the canonical schedule only specifies who plays whom when and not where.

Miyashiro et al. [10] provide a $2+(9/4)/(n-1)$ approximation for the intensively studied special case $k=3$ by means of the *Modified Circle Method*, a variation of the canonical schedule. In [15] Yamaguchi et al. obtain an algorithm with approximation ratio $(2k-1)/k+O(k/n)$ for $k \leq 5$ and $(5k-7)/(2k)+O(k/n)$ for $k>5$. Again they make use of the canonical schedule, now refined such that the teams are ordered around the 'table' such that most of the distances driven are part of a near optimal traveling salesman tour which clearly has positive effects on the length of many distances traveled. As $k \leq n-1$, they showed this way that a constant factor approximation for any choice of k and n exists. However, they did not show how this factor looks like exactly.

1.3 Our Results

Our aim however is to approximate the TTP by a constant ratio for arbitrary choices of k and n.

Applying the canonical schedule mentioned above, we choose a specific orientation of the underlying graph which ensures that home stands and road trips do not contain more than k matches and for which the total distance traveled is not too long. Whereas it is common practice to derive the second half of the season by repeating the first half's games in the same order but with changed home field advantage, it is not suitable here, as road trips or home stands might become too long. Thus, we derive the second half in a different way. Finally, we show that the plan we construct approximates the optimal solution by a factor of $2+2k/n+k/(n-1)+3/n+3/(2\cdot k)$. For the case of $k=3$ this guarantees an approximation ratio of $5/2+12/(n-1)$ which is actually not better than the ratio of Miyashiro et al. cited above. But for any choice of $k \geq 4$ (and thus $n \geq 6$) this yields an approximation ratio of less than 5.875, which is the first constant factor approximation for $k>3$.

2 Lower Bounds

The objective of the TTP, minimizing the total travel distance of all teams during a double round robin tournament, can be estimated by various bounds. One of them is called Independent Lower Bound (ILB) [6] and consists of finding the shortest tour for each team individually, independent of the other team constraints. (primarily that B has to be at home when A visits B during one of A's road trips). Finding an ILB is equivalent to solving a capacitated vehicle routing problem. In this paper we will use an even coarse version of ILB where we focus only on a traveling salesman tour traversing all venues.

Theorem 1 *Let ρ be the length of a TSP in G. Every solution of the TTP has a total length of at least $n \cdot \rho$.*

Proof Every team has to visit all the other teams. Thus, each team has to travel at least a distance of ρ which gives a total distance of $n \cdot \rho$.

As in [10], we denote the sum of the distances of all ordered pairs of teams as $\Delta = \sum_{i,j \in T} d(i,j)$. Miyashiro et al. [10] showed a lower bound of $2/3 \cdot \Delta$ for the objective function of TTP with $k = 3$. We generalize this result for arbitrary k:

Theorem 2 *Every solution of the TTP has a total length of at least $2/k \cdot \Delta$.*

Proof Consider an arbitrary solution and suppose team i plays $l \leq k$ consecutive away games at teams $t_1, t_2, \ldots, t_l$. The distance $\tilde{d}_i$ driven thereby is

$$\tilde{d}_i = d(i,t_1) + \sum_{j=1}^{l-1} d(t_j, t_{j+1}) + d(t_l, i)$$

Because of the triangle inequality we have $\tilde{d}_i \geq 2 \cdot d(i,t_j)$ for all j and thus we have

$$l \cdot \tilde{d}_i \geq 2 \cdot \sum_{j=1}^{l} d(i,t_j) \quad \Longrightarrow \quad \tilde{d}_i \geq \frac{2}{k} \cdot \sum_{j=1}^{l} d(i,t_j)$$

Summing up over all tours driven yields the desired lower bound of $2/k \cdot \Delta$ for the total distance driven by the teams in any solution.

3 Construction of the Tournament

For $i \in V$ let $s(i) := \sum_{j \in V} d(i,j)$ be the *star-weight* of i. Since $\sum_{i \in V} s(i) = \sum_{i \in V, j \in V} d(i,j) = \Delta$, there has to be one $j \in V$ for which $s(i) \leq \Delta/n$. Let T_{heu} be a tour through all of the teams' venues which has been found by applying the well known heuristic by Christofides [4]. Therefore, we know that this tour is not more than 1.5 times longer than the shortest possible tour. We furthermore assume that the teams are named in a way such that T_{heu} traverses them in the order $1, 2, \ldots, n$ and that n is the team with minimum star weight. Given this tour we construct a solution of the TTP in the following way. For $n = 20$ the games of the first two days of the season are displayed in Figure 1 and 2. The Figures corresponding to other choices of n can be derived analogously. A solid arc (u,v) in this digraph means that team u is playing against team v in the arena of team v. The games of the other days can be derived analogously by changing the positions of the teams counterclockwise. The only arc which

changes its orientation during one half of the season is the arc incident to node n which changes its orientation every kth match. This way, the season starts for team 4 with a tour visiting the teams $16, 17, 18$ and 19 before coming home and then playing against the teams $1, 2$ and 3. Then, it starts off again to play against $20, 5, 6, 7$, and has then a home stand again consisting of matches against $8, 9, 10, 11$. Finally, there is a last road trip including 12 and 13 and a last home stand with 14 and 15. It is clear that no team has home stands or road trips which are longer than k matches. And it is also clear that every two teams have met each other during this first $n-1$ games.

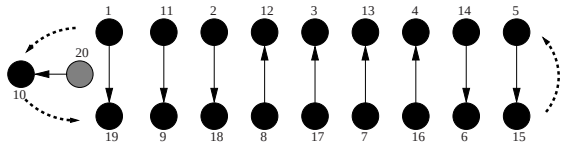


Fig. 1 Example for slot 1 with $n = 20$, $k = 4$ and $l = 2$

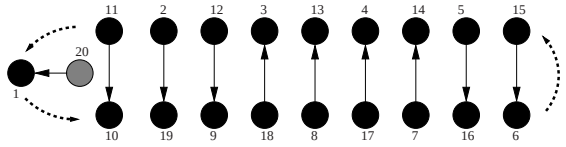


Fig. 2 Example for slot 2 with $n = 20$, $k = 4$ and $l = 2$

In order to construct a full tournament, it remains to construct the second half of the season. If we just repeated the first $n-1$ matches with changed locations (changed the orientation of the arcs), we would obtain a solution, in which every pair of teams met twice and these two games took place at different sites. Furthermore, no half of the season contained a road trip or a home stand longer than k matches. However, this solution could contain road trips and home stands being longer than k. For example, the team 4 we considered above would start into the second half of the tournament with a home stand of length 4 after having ended the first half with two home stands. In order to get rid of this problem, we start the second half with the match of day $n-2$, succeeded by the matches of the days $n-1, 1, 2, \ldots, n-3$ in this order. The double round robin tournament obtained this way contains neither road trip nor home stand longer than k. To see this, assume for the sake of a contradiction that there is a team t which has a road trip longer than k. It is clear from construction that no half of the season completely contains such a tour. Thus, the tour has to include the days $n-1$ and n. In case t has away-games at both of these days, the other matches involving these opponents will be home-games for t. By construction, these games will take place on the days $n-2$ and $n+1$ which means that the road trip had only a length of 2, contradicting the assumption. The case for too long home stands follows along the same lines.

By looking at the figures presented above, one can see that every home stand or road trip is defined by a set of consecutive arcs pointing in the same direction. We call such a set of arcs a *block*. Furthermore, any orientation of the arcs defining the schedule gives rise to a feasible schedule, as long as the blocks do not contain more than k arcs. The leftmost block is not even allowed to contain more than $k-1$ arcs because of the games team n is

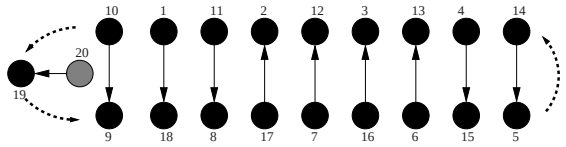


Fig. 3 Example for slot $n-1$ with $n = 20$, $k = 4$ and $l = 2$

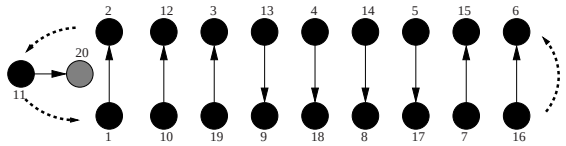


Fig. 4 Example for slot n with $n = 20$, $k = 4$ and $l = 2$

involved in. As long as we obey these rules for the maximum sizes of blocks stated above, we will always obtain a feasible plan for any choice of orientations of the arcs defining the tournament.

In the following, we consider k different orientations. The main difference between them is the width of the rightmost block. For $l \in \{1,\ldots,k\}$ let O_l be the orientation in which the rightmost block has width l, the blocks in the middle all have width k and the leftmost block contains the rest (see Figure 5). In case this leads to the leftmost block containing exactly k arcs, we change the orientation of the edge (u_1, v_1), such that the arc incident to team n cannot prolong the road trips induced by this block to have a length of $k+1$ matches. The left- and rightmost arcs in a block always define the first and the last match of a trip.

4 Costs of the Tournament

In this section we will prove an upper bound for the total length of the tours defined by the tournament constructed in the previous section.

We assume that every team t having an away game against team n will drive home first before driving to team n's site and drives home after having played that match. By construction, t has a home game before or after that game anyway. We just obtain one more visit home this way. By the triangle inequality, the costs incurred this way are only higher than before. Furthermore, we will apply the triangle inequality a second time by assuming that every team drives home after the last game of the first half if it is not already at home. Let the nodes of the underlying graph be denoted as $u_1, u_2, \ldots, u_{n/2-2}$ and $v_1, v_2, \ldots, v_{n/2-2}$ (see Figure 5).

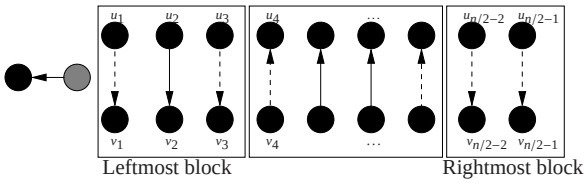


Fig. 5 The blocks defined by the orientation O_2

In the following we will estimate the distances related to the constructed tournament separately:

1. C_h - the costs related to home-games of team n
2. C_a - the costs related to away-games of team n
3. C_s - the costs related to the first days of the season-halves and the costs of returning home after the last days of the season-halves
4. C_l - the other costs incurred by the edge (u_1, v_1)
5. C_r - the other costs incurred by the edge $(u_{n/2-1}, v_{n/2-1})$
6. C_o - the other costs

C_h - The costs related to home-games of team n : Every other team plays against team n once. As we can assume by application of the triangle inequality that all teams come from their home venues to play against team n and return to their home venues after the game, we know that the cost incurred thereby is at most

$$C_h \leq \sum_{i=1}^{n-1} d(i,n) + d(n,i) = 2 \cdot s(n) \leq 2 \cdot \Delta / n$$

where the last follows from the assumption of n being the node with the smallest star-weight.

C_a - The costs related to away-games of team n: Analogously, to the estimation of the home-games of team n, we can upper bound the costs incurred by the away games by first assuming that team n always returns home after each away-game. This way, we derive the same upper bound of $2 \cdot \Delta / n$ for the costs $C_a$incurred by the away-games of team n.

C_s - The costs related to the first days of the season-halves and the costs of returning home after the last days of the season-halves: At the first day of the season, $n/2$ teams have to travel to their opponents. We do not consider the game that team n is involved in, as we have already taken care of these costs above. So, there are $n/2-1$ distances traveled left which correspond directly to the vertical arcs of Figure 1. After the games of day $n-1$ the first half of the season is over, and we assume that all teams drive home. The second half of the season starts with the matches which have already taken place at day $n-2$ and it ends with the second leg of the game of day $n-3$. Observe, that the orientation of the arcs does not have an effect on the total distance driven. It only affects the question who is driving which is not of interest here. In the example mentioned above, for team 4 these are the teams $16, 15, 14$ and 13. If team 4 did not start the season this way but with a match against team 15, then we would need to consider the distances to the teams $15, 14, 13$ and 12. This way we obtain $n-1$ different choices for the first and last trips of the two halves of the season. Furthermore, it is easy to see that each edge of $(\{1,\ldots,n-1\} \times \{1,\ldots,n-1\})$ is part of at most four of these choices. So, summing up the distances of the $n-1$ different possible choices for day 1, we obtain a total of at most

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n-1} 4d(i,j) = 2\Delta - 4 \cdot s(n)$$

So, there has to be a choice for which we can estimate

$$C_s \leq 2 \cdot (\Delta - 2 \cdot s(n)) / (n-1).$$

C_l - The costs incurred by the edge (u_1, v_1): As we assumed that every team's trip to team n starts at the home-site and leads back there after the match, there is always a trip ending or starting with a trip along the edge (u_1, v_1). Apparently, these are always trips between teams being neighbors on the heuristically obtained tour T_{Heu}. As these teams will meet in both halves of the games, the edges have to be counted twice and the cost incurred on that arc can thus be estimated as

$$C_l \leq 2d(n-1,1) + 2\sum_{i=1}^{n-2} d(i,i+1) \leq 2 \cdot d(T_{Heu}).$$

C_r - The costs incurred by the edge $(u_{n/2-1}, v_{n/2-1})$: In the first half of the season, the edge $(u_{n/2-1}, v_{n/2-1})$ always marks the end of a trip, whereas it stands for the beginning of a trip in the second half of the season. The costs incurred in both halves together can be estimated as follows.

$$\begin{aligned} C_r &= 2 \cdot \left(\sum_{i=1}^{n/2} d(i, i+n/2-1) + \sum_{i=n/2+1}^{n} d(i, i-n/2) \right) \\ &= \sum_{i=1}^{n/2} (d(i, i+n/2-1) + d(i+n/2-1, i)) + \sum_{i=n/2+1}^{n} (d(i, i-n/2) + d(i-n/2, i)) \\ &\leq \sum_{i=1}^{n/2} \mathsf{OPT}_i + \sum_{i=n/2+1}^{n} \mathsf{OPT}_i = \mathsf{OPT} \end{aligned} \tag{1}$$

with OPT_i denoting the length of team i driven in an optimal solution of total length OPT. Every possible solution has to contain a trip for any team $i \in \{1, \ldots, n/2\}$ which covers team $i+n/2-1$. For the length of this trip is not longer than $d(i, i+n/2-1) + d(i+n/2-1, i)$ and we can make similar observations for the other teams as well, inequality 1 follows.

C_o - The other costs: As already mentioned earlier in this paper, we do not only consider the orientation of the arcs as displayed in Figures 1 - 3. Instead, we will consider k different orientations. The difference between them is the width of the rightmost block, the block including the arc $(u_{n/2-1}, v_{n/2-1})$ or resp. $(v_{n/2-1}, u_{n/2-1})$. For $l \in \{1, \ldots, k\}$ let O_l be the orientation in which the rightmost block has width l, the blocks in the middle have width k and the leftmost block contains the rest. In case, this leads to the leftmost block containing exactly k arcs, we change the orientation of the edge (u_1, v_1), such that the arc incident to team n cannot prolong the road trips induced by this block to have a length of $k+1$ matches.

In every half, every team i is associated to one of the nodes $v_1, v_2, \ldots, v_{n/2-1}$ exactly once. When it is associated to node v_j it plays against the team $(i+j-1) \mod (n-1)+1$ which is associated to node u_j at that time. In case the edge (u_j, v_j) marks the first or the last game of a road trip in the first or the second half of the tournament, we call this edge a *home-edge* (the dashed arcs in Figure 5). If the home-edge corresponds to the beginning of a trip in the first half of the season, it marks the end of a tour in the second half of the season. Therefore, the distance associated with this edge is driven exactly twice in the corresponding tournament. Let us have a closer look at the costs which are being incurred by teams traveling along the home-edges. Since every direct travel from or to i's home site can only happen via exactly one home-edge, and as there are at most two orientations in which some edge (u_j, v_j) is a home-edge, the overall costs incurred by the home edges is at most 2Δ. It still remains to estimate the distances traveled which are not from or to the traveling

teams' home sites. A trip which visits l teams consists of two drives along home-edges and $l-1$ drives inbetween. By construction, these $l-1$ rides are driven along edges which are part of the heuristically obtained tour T_{Heu}. Let u_j be a node which does not represent the beginning of a trip. Whenever a team i is assigned to this node, there is another team l visiting i after having played an away match at the team $i-1$, the predecessor of i in T_{Heu}. Thus, for any node u_j or v_j which does not represent the beginning of a trip, we can estimate the sum of the distances driven to get to the teams assigned to this node as no more than $d(T_{Heu})$. Since there are no more than $n/2-2$ such nodes, the distances driven here are not more than $(n-2)d(T_{Heu})$.

For there are k different orientations, there has to be one with total distance incurred by the home-edges not more than

$$C_o \leq \frac{2\Delta + (n-2)d(T_{Heu})}{k}$$

5 The Approximation ratio

If we choose the parameters in the above mentioned ways, we obtain an approximation ratio of

$$\begin{aligned}
&\frac{C_h + c_a + C_s + C_l + C_r + C_o}{\mathsf{OPT}} \\
&\quad \leq \frac{2\Delta/n + 2\Delta/n + 2\cdot(\Delta - 2s(n))/(n-1) + 2\cdot d(T_{heu}) + \mathsf{OPT} + \frac{2\Delta+(n-2)d(T_{Heu})}{k}}{\mathsf{OPT}} \\
&\quad = \frac{2\Delta/n + 2\Delta/n + 2\cdot(\Delta - 2s(n))/(n-1)}{2/k\cdot\Delta} + \frac{2\cdot d(T_{heu})}{n\cdot d(T_{opt})} + 1 + \frac{2/k\cdot\Delta}{2/k\cdot\Delta} + \frac{(n-2)/k\cdot d(T_{Heu})}{n\cdot d(T_{opt})} \\
&\quad \leq \frac{4\Delta/n + 2\cdot\Delta/(n-1)}{2/k\cdot\Delta} + \frac{3}{n} + 1 + 1 + \frac{(n-2)/k\cdot 3/2\cdot d(T_{Opt})}{n\cdot d(T_{opt})} \\
&\quad \leq \frac{2/n + 1/(n-1)}{1/k} + \frac{3}{n} + 2 + 3/(2\cdot k) \\
&\quad = 2k/n + k/(n-1) + \frac{3}{n} + 2 + 3/(2\cdot k)
\end{aligned}$$

As $k \leq n-1$, this bound cannot be larger than $5 + \frac{3}{n} + 3/(2\cdot k)$ which is not more than 5.875 for $k \geq 4$ and $n \geq 6$.

References

1. Anagnostopoulos, A., Michel, L., Van Hentenryck, P., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. In Proceedings CPAIOR'03, Montreal, (2003)
2. Ball, B.C. , Webster, D.B.: Optimal scheduling for even-numbered team athletic conferences. AIIE Transactions, 9:161169, (1977)
3. Benoist, T., Laburthe, F., Rottembourg, B.: Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems. In Proceedings CPAIOR'01, Wye College (Imperial College), Ashford, Kent, UK (2001)
4. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, CMU (1976)
5. de Werra, D.: Scheduling in Sports. In: P. Hansen, Editor, Studies on Graphs and Discrete Programming, North-Holland, Amsterdam (1981) 381-395.

6. Easton, K., G. Nemhauser, and M. Trick.: The traveling tournament problem: Description and benchmarks. *Lecture Notes in Computer Science* **2239** (2001) 580-585
7. Easton, K., Nemhauser, G., Trick, M.: Solving the traveling tournament problem: a combined integer programming and constraint programming approach. In E. Burke and P. De Causmaecker, editors, Practice and Theory of AutomatedTimetabling IV, volume 2740 of Lecture Notes in Computer Science, 100-109. Springer Berlin / Heidelberg (2003)
8. Henz, M.: Playing with constraint programming and large neighborhood search for traveling tournaments. In E. Burke and M. Trick, editors, Proceedings PATAT 2004, 23-32 (2004)
9. Kendall ,G., Knust, S., Ribeiro, C. C., Urrutia, S.: Scheduling in Sports: An Annotated Bibliography. Computers & Operations Research, 37, 1-19 (2010)
10. Miyashiro, R., Matsui, T., Imahori, S.: An approximation algorithm for the traveling tournament problem. The 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)
11. Rasmussen, R.V., Trick, M.A.: Round robin scheduling - a survey. European Journal of Operational Research 188 (2008), 617-636
12. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research, volume 179 number 3, 775-787 (2007)
13. Trick, M.: Challenge Traveling Tournament Instances (2009), `http://mat.gsia.cmu.edu/TOURN/`
14. van Hentenryck, P., Vergados, Y.: Traveling tournament scheduling: A systematic evaluation of simulated annealing. In J.C. Beck and B.M. Smith, editors, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 3990 of Lecture Notes in Computer Science, 228-243. Springer Berlin / Heidelberg, (2006)
15. Yamaguchi, D., Imahori, S., Miyashiro, R., Matsui, T.: An improved approximation algorithm for the traveling tournament problem. Mathematical Engineering Technical Report, METR09-42, Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo (2009)

Comparison of Algorithms solving School and Course Time Tabling Problems using the Erlangen Advanced Time Tabling System (EATTS)

Peter Wilke · Helmut Killer

Abstract Seven algorithms are applied to two common time tabling problems: school and course time tabling. The algorithms were implemented using the EATTS (Erlangen Advanced Time Tabling System) which allows to compare and evaluate the algorithms regarding their performance and ability to solve the problems.

Keywords Great Deluge · Harmony Search · Simulated Annealing · Genetic Algorithms · Tabu Search · Immune System · Real World Problems · EATTS · Erlangen Advanced Time Tabling System

1 Introduction

Recently we had to deal with two real world problems, namely school and course time tabling, and jumped on this opportunity to investigate regarding their ability and performance to solve the given problems. All algorithms have been implemented using the current version of EATTS (Erlangen Advanced Time Tabling System [Wil10]), to allow comparison and evaluation.

2 The Problems

2.1 School 2009 Time Tabling

The data for our School 2009 time tabling problem represents an existing school with students from year 1 to 10. In this scenario the classes and their subjects are given, while class rooms and time slots have to be assigned to the events. Teachers can be assigned fixed to class/subject pairs, but don't have to. Here it is sufficient that one student represents the entire class, details given in table 1.

Peter Wilke
Universitaet Erlangen-Nuernberg, Department Informatik, Martensstrasse 3, 91058 Erlangen, Germany
Ph.: +49 (9131) 85-27998
E-mail: Peter.Wilke@Informatik.Uni-Erlangen.DE Helmut Killer
E-mail: Helmut.Killer@gmx.DE

Events:	178
Resources of type TimeSlot:	81
Resources of type Class:	14
Resources of type Student:	14
Resources of type Teacher:	28
Resources of type Room:	37
Resources of type Subject:	78
Resources of type Asset:	0
Resources of type LessonProperties:	178
Resources of type Building:	1
Resources:	431
Resources to be assigned:	118
Number of possible solutions:	$> 10^{439}$

Table 1 The main characteristics of the School 2009 example

2.2 MuT 2009 courses at a university

Our university organises a girl-and-technology (in german: „Maedchen und Technik“, abbreviated MuT) week each year to attract more female students to technical subjects. In this scenario the tutors and time slots for the events are fixed while students (not classes) have to be assigned to the project of their choice, details given in table 2.

Events:	229
Resources of type TimeSlot:	57
Resources of type Subject:	52
Resources of type Girl:	170
Resources:	279
Resources to be assigned:	170
Number of possible solutions:	$> 10^{656}$

Table 2 The main characteristics of the MuT 2009 example

3 Algorithms

The following algorithms were implemented:

- Genetic Algorithms
- Immune System
- Harmony Search
- Tabu Search
- Simulated Annealing
- Great Deluge
- Walk Down Jump Up

Table 3 shows the used algorithms and their main characteristics, indicated by a mark in the corresponding row. The characteristics are:

population In each iteration one or more solution candidates are produced and substitute older solutions.

trajectory In each iteration only one solution candidate exists.

history The algorithms depends (at least partial) on its history of computational steps.

limit At each step of the computation the limit, which may vary during the computation, determines if the newly generated solution is accepted as new current solution candidate.

round based In each round, i.e the iteration step, one or more solutions are generated, but only one solution is selected for the next iteration step.

references For detailed information about the algorithms please consider the recommended bibliographic references for reading.

	population	trajectory	history	limit	round based	references
Genetic Algorithms	x					[Gol89, GD91, Whi89, Sys91, BT95a, BT95b]
Immune System	x				x	[MKM06]
Harmony Search	x					[ABKG08, Gee09]
Tabu Search		x	x		x	[GS01, KH05]
Simulated Annealing		x		x	x	[Hel04, vL87, FSAPMV08]
Great Deluge		x		x	x	[BBNP04]
Walk Down Jump Up		x			x	[Kil09, WK10]

Table 3 Used Algorithms and their characteristic properties

All algorithms were implemented using the EATTS Erlangen Advanced Time Tabling System framework and all runs were performed using computers as specified in table 4.

The following results are achived with sequential versions for Simulated annealing, Great Deluge, Walk Up Jump Down, while parallel versions were used for Genetic Algorithm, Harmony Search, Immune System and Tabu Search. All experiments were run on the same computer so results are comparable.

Two different setups for the experience where used. One setup was designed to achieve the fastest reduction of costs. The other setup was used to see the long term improved behaviour of the algorithms, e.g. does the algorithms profit form excessive computation. Both setups starts with randomly generated initial solutions.

3.1 Simulated Annealing

Basically Simulated Annealing maps the cooling down process of matter to optimization problems. The standard version uses an acceptance probability function which is fixed during cool down, in our implementation we allow modifications of this function over time, i.e. the probability of making the transition from the current state s to a candidate

	QuadCore	DuoCore
CPU:	Intel Core4Quad 2.8 GHz	Intel Core2Duo 3,0 GHz
RAM:	8 GB	4 GB
OS:	Debian Linux Kernel 2.6 x86-64	Suse Linux Kernel 2.6 i686
JVM:	Java 6 32 Bit Server JVM	Java 6 32 Bit Server JVM

Table 4 Specs of the computers used

new state s' is specified by the acceptance probability function P(e,e',T,t), that depends on the energies e = E(s) and e' = E(s') of the two states, on a global time-varying parameter T called the temperature, and the time t representing the time spend on the cooling process so far. In our simulations we used this parameter to adapt the temperature decrease on the maximum available time to cool down, i.e. the temperature always reaches it's minimum when the computation deadline is reached.

The performance of Simulated Annealing can be characterized as a slow starter but winner, see fig. 9. It's clearly visible that Simulated Annealing reduces the costs slower than all other algorithms except Great Deluge, which is designed to perform in an linear manner, see section 3.6.

Fig. 1 shows the details of the descent of the costs. The dots show the current solutions under review and the line indicates the current best solution. The gap between the dots and the line is the middle range of a distribution of solutions. The ranges is determined by an acceptance criteria (upper limit) and the current best solution (lower limit).

Remark: To make results more easily to interpret the x-axis (time) in the performance plots has been scaled to extend phases of fast decline and to condense phases of stagnation.

Experiments with parallel versions of the Simulated Annealing algorithms haven't shown sufficient results and will be subject to a closer investigation.

The starting temperature is chosen randomly as part of the random initial solution. Cooling rate was initialy 0.99 and was adapted linearly to reach 0.0 at the end of the given computation time.

3.2 Tabu Search

Tabu Search algorithm is trajectory based but can effectively be computed using multi threading because for the calculation of a new solution at first a set of neighbours is calculated and evaluated and this can be done in parallel. The best speed up is achieved when the number of threads matches the number of available cores respectively CPUs. On a Core2Quad CPU 19.4 iterations/second are computed when only one core is used, the multi-threading version achieves 45.3 iterations/second yielding a speed up factor of 2.33.

Fig. 2 shows the details when Tabu Search solves the MuT 2009 example. In contrast to the nearly evenly solution candidate distribution of Simulated Annealing here the solution candidates are concentrated near the currently best solution.

The size of the tabu list was 40, 200 neighbours were generated in each step and the champion was the initial solution of the next iteration.

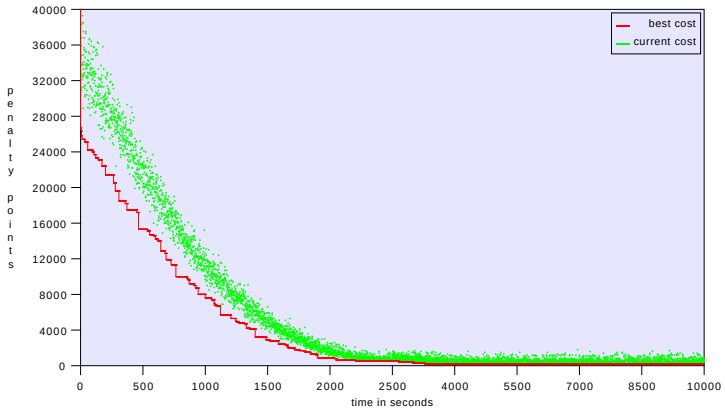


Fig. 1 Simulated Annealing solving the School 2009 problem

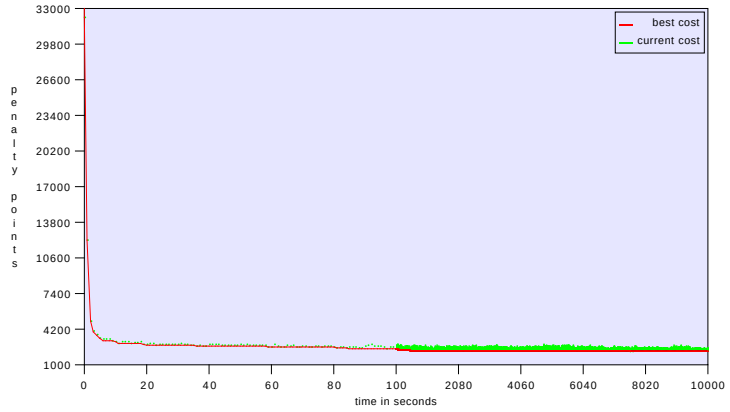


Fig. 2 Tabu Search solving the MuT 2009 problem

3.3 Genetic Algorithm

In our implementation all genetic operators have a complement operator, which reverses the effect of the operator. This speeds up the computation because it enables multiple operator applications. A crossover requires significant more computing time than a mutation, therefore reversal of a mutation leading to a lethal chromosome after a successful crossover avoids the waste of the time spend for the crossover. So our implementation reverses all genetic operations when they don't lead to a better solution.

Genetic Algorithms are population based and therefore a good candidate for parallel computation. We implemented an Island Ferry concept [CP95] which we optimized for use on multi-core CPUs. For each CPU core a genetic algorithm instance is started and

all instances exchange their best solutions periodically over time, so all populations reach the current global lowest cost value.

Fig. 3 shows the usual descent of a Genetic Algorithm with its typical plateaus.

Experiments were run with population sizes of 10 or 100, approx. half of the population is replaced in each generation. A two-point cross-over and a mutation rate of 0.005 was used.

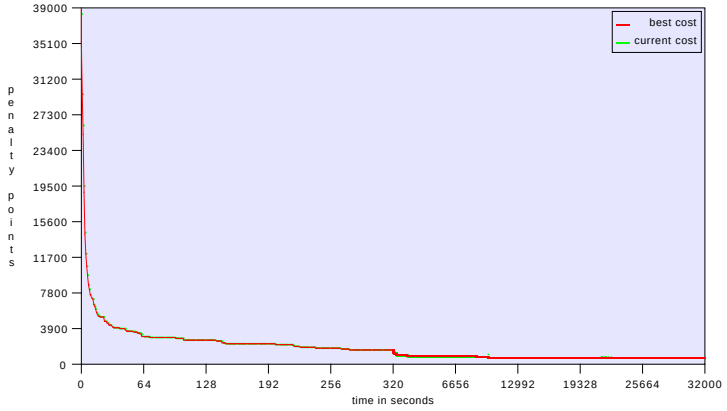


Fig. 3 Genetic Algorithm solving the School 2009 problem

3.4 Harmony Search

We first modified the original Harmony Search algorithm to allow variants with a wider bandwidth as suggested by [MFD07]. But the results were disappointing. Better results were achieved when stagnation was dissolved using randomly generated parameter values or chosen from an interval (shake).

In each iteration Harmony Search generates a new solution which can be post-processed by the same hybrid operators as used in Genetic Algorithms.

Harmony Search can be computed in threads, but the threads depend on the same Harmony Memory. The number of threads should match the number of cores/CPUs, otherwise threads sharing a core/CPU will fall behind the other threads and produce solution candidates which are based on outdated versions of the Harmony Memory. This means that precious computing time is dissipated.

Fig. 4 shows a rapid decline in the first seconds followed by a much smoother and slower descent, a quite different behaviour compared to the other algorithms. This nearly 90 degree turn is an extreme example but in all our Harmony Search experiments similar turns were observed. The parameters used were Harmony Memory Consideration Rate 0.99, Pitch Adjusting Rate PAR 0.01, Harmony Memory Size HMS 10 or 100, bandwidth = 0.1.

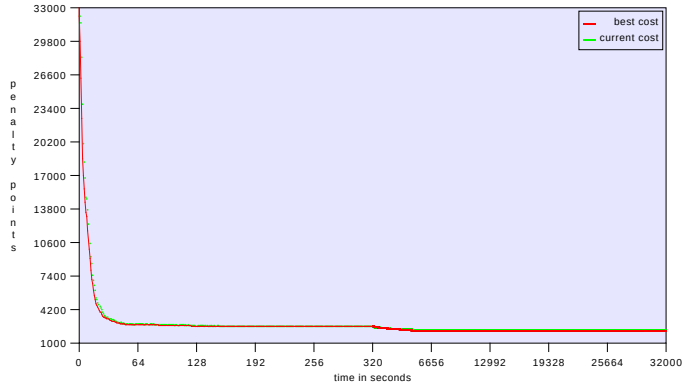


Fig. 4 Harmony Search solving the MuT 2009 problem

3.5 Immune System

The original Immune System suggests three variants: clonal selection, immune network and negative selection. Best results on our problems were achieved using the negative selection variant which basically eliminates candidates (so called detectors) with fitness under the mean fitness value of the population.

Because Immune System is population based it is a good candidate for multi threaded computation. The detectors can be generated and evaluated in parallel because they don't depend on each other. The main algorithm wait until all detectors are generated and evaluated, therefore it is recommended for efficiency reasons to generate generators in numbers which are multiples of the number of available cores/CPUs.

Fig. 5 shows a remarkable pattern in the distribution of solution candidates. Three clearly distinct bands are visible: the first close above the champion, the second a short gap above the first and a very thin third quite far away from the other two. The first two bands represent the candidates below/above the mean fitness.

Fig 6 shows even more bands. The gap between the bands corresponds to the given penalty cost. The line indicating the drops down immediately after start and is stable for the rest of the run.

For Immune System a population size of 10 or 100 was used and approx. half of the population was replaced in each iteration. Parameters were maxMultiMoves 10.0, Multimove tweak, allEqualFactor 1.1, all equal tweak.

3.6 Great Deluge

The original Great Deluge algorithm has a linear descending limit. Generated solutions with cost above this limit are rejected and below are accepted. The best solution is always saved. A small modification is the introduction of a best solution backup. When current solution cost can't get below the limit for a certain amount of time, the saved best solution can be restored if it has lower cost, so the algorithm is revived.

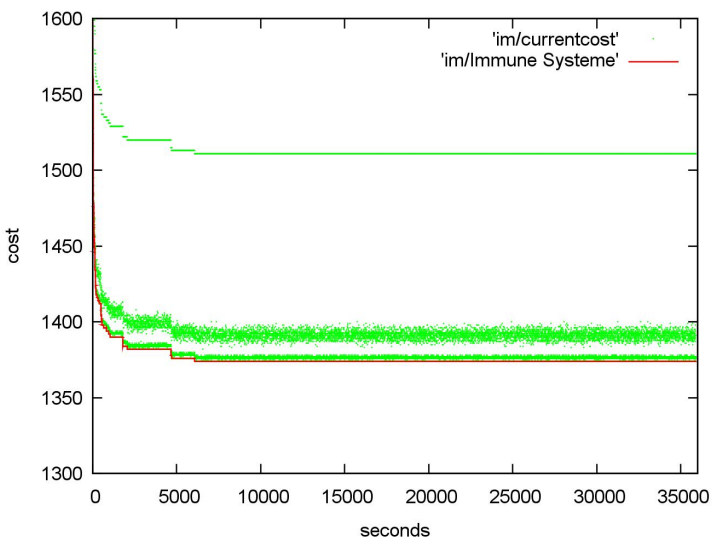


Fig. 5 Immune Systeme solving the MuT 2009 problem

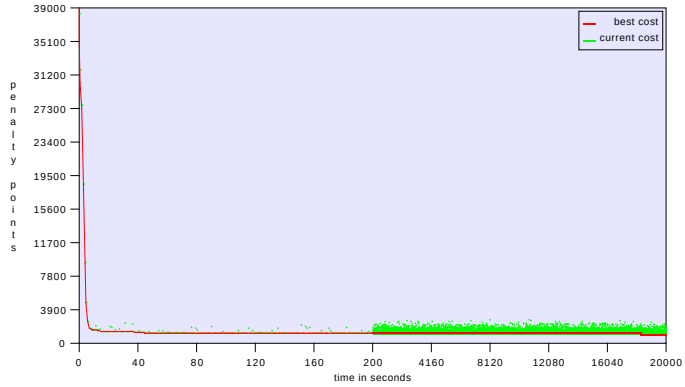


Fig. 6 Immune System solving the School 2009 problem

Fig. 7 shows the behaviour of Great Deluge in detail. The line indicates the best solution while the dots represent solution candidates. The sharp edge on top of the solution cloud is due to the limit function.

The linear decreasing rate was adopted to the expected runtime, i.e. initial cost divided by runtime in seconds giving the value to be subtracted in each step.

3.7 Walk Down Jump Up Algorithm

Walk Down Jump Up [WK10, Kil09] combines hill climbing, jump operator and Great Deluge. It begins with an initial random solution and starts it's descent until a local

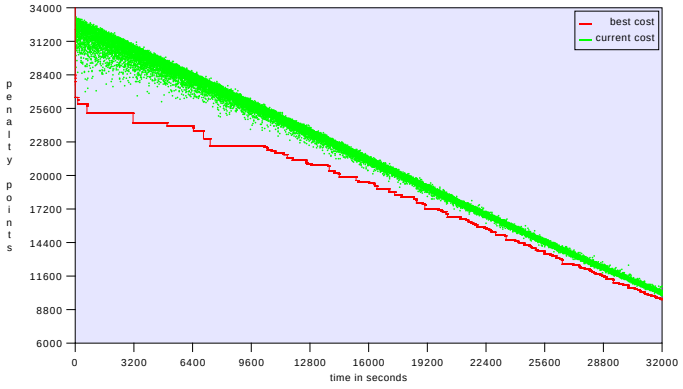


Fig. 7 Great Deluge solving the MuT 2009 problem

minimum is reached. Then the jump operator is used to set an higher acceptance limit, i.e. all newly generated solutions with costs below this limit are accepted and each following iteration will decrease this limit until a new local minimum is reached. If the new local minimum is not better than the old one the height of the jump is increased, otherwise the search continues from the newly found local minimum and jump height is reset. A small modification is best solution backup, when current solution reached a local minimum which is above best solution, then best solution is restored before next jump.

Walk Down Jump Up is trajectory based and again therefore multi threading is not really straight forward. Because the Jump Up operator can jump pretty far even if the current solution is quite good it males sense to start the Walk Down Jump Up algorithm on all available cores/CPUs and use the Island Ferry mechanism to exchange the best solutions.

Fig. 8 shows details of the performance. Walk Down Jump Up has a very steep descent phase of approximately 200 seconds at the very beginning followed by a long stagnation phase for the rest of the 20.000 seconds run.

4 Summary

The performance of all 7 algorithms applied to the 2 real world problems is shown in fig. 9. The diverse nature of the problems has different impacts on the algorithms. While the Immune System algorithms is leading head on head with Simulated Annealing and Walk Down Jump Up when solving the MuT 2009 example, is comes in second to last when solving the School 2009 example. Great Deluge shows excellent performance on the School 2009 example, but is extremely bad on the MuT 2009 example.

Table 5 resp. 6 show the results achieved in 10 hours.

While all algorithms except the Great Deluge algorithm show nearly equal performance, the situation for the School 2009 example is quite heterogeneous. Here the field is lead by Walk Up Jump Down, Great Deluge and Simulated Annealing, a mid field

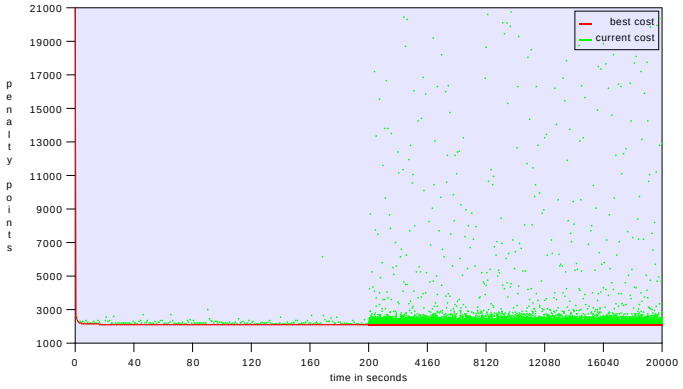


Fig. 8 Walk Down Jump Up solving the MuT 2009 problem

consisting of Genetic Algorithm and Harmony Search, and trailed by Immune System and Tabu Search.

MuT 2009 for 10h	solved	final cost	stagnation after
Genetic algorithm	yes	2216	6h:6m
Great Deluge	no	7300	no
Harmony Search	yes	2196	9h:42m
Immune System	yes	2142	1h:48m
Simulated Annealing	yes	2130	2h:56m
Tabu Search	yes	2273	2h:5m
WalkDownJumpUp	yes	2138	4h:6m

Table 5 Comparison of all 10h runs for the MuT 2009 problem

School 2009 for 10h	solved	final cost	stagnation after
Genetic algorithm	no	628	9h:56m
Great Deluge	yes	256	4h:57m
Harmony Search	yes	319	9h:18m
Immune System	no	1055	5h:5m
Simulated Annealing	yes	206	1h:10m
Tabu Search	no	1182	0h:0m:5s
WalkDownJumpUp	yes	278	0h:12m

Table 6 Comparison of all 10h runs for the School 2009 problem

A closer look on the time available for the computation shows that two algorithms, namely Genetic Algorithm and Harmony Search, can profit from additional computing time, while the others don't improve their costs significantly relative to the solutions

found after only 10 seconds, but the hard constraints become fulfilled, so the solutions become "more and more" valid. This observation is true for both examples.

Overall winner is Simulated Annealing, best results and least expensive runtime costs make this algorithm the best choice for the given problems.

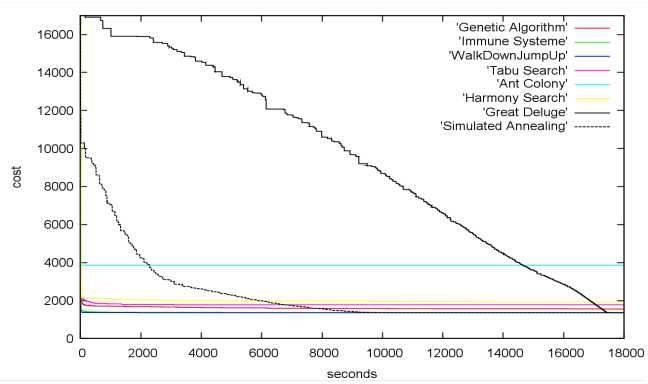


Fig. 9 Performance of all algorithms solving the MuT-Problem over 5 hours

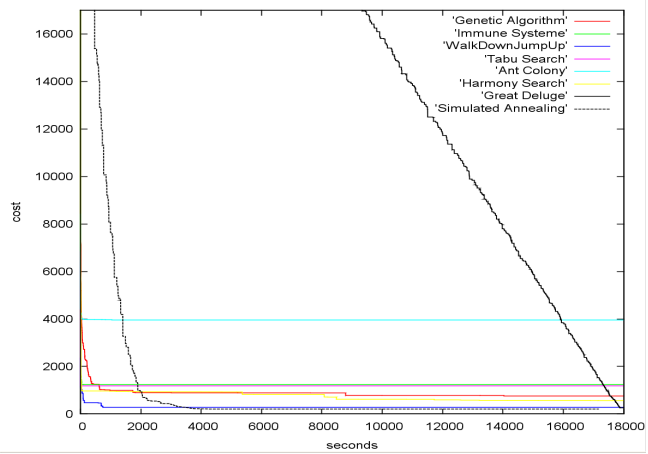


Fig. 10 Performance of all algorithms solving the School-Problem over 5 hours

5 Conclusion

The investigations have shown that it is a good idea to test several algorithms on their ability to solve a given problem, even if the problems look quite similar.

Future work will be the extension of our database of problem descriptions and implementation of additional algorithms.

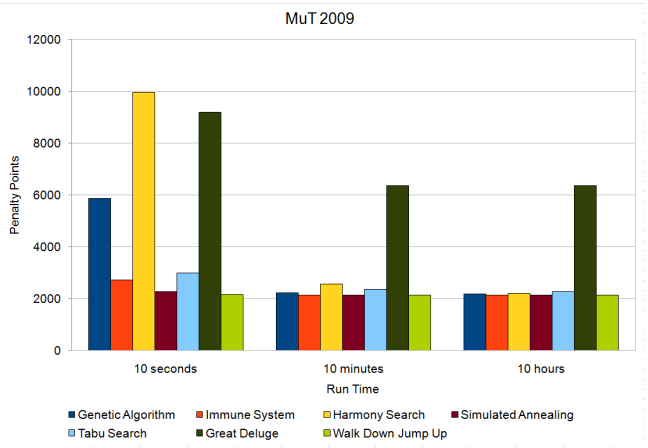


Fig. 11 All algorithms and their performance on the MuT 2009 example

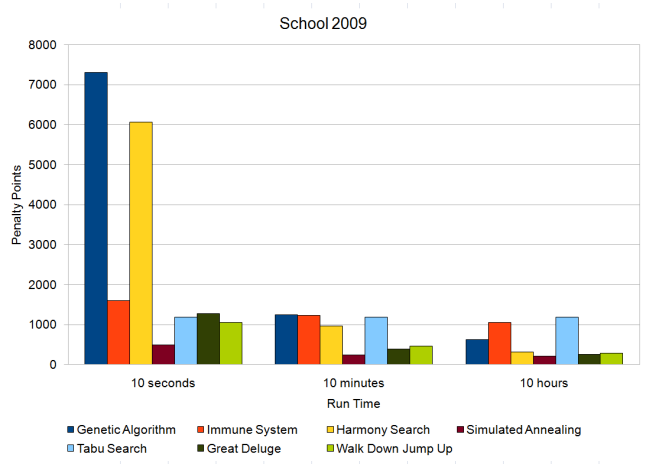


Fig. 12 All algorithms and their performance on the School 2009 example

Acknowledgements We would like to thank all our colleagues at EATTS for their support and special thanks to Dimo Korsch, Sabine Helwig, Johannes Ostler for their contributions to the implementation and test of the EATTS algorithms toolbox.

References

[ABKG08] Mohammed Azmi Al-Betar, Ahamad Tajudin Khader, and Taufiq Abdul Gani. A harmony search algorithm for university course timetabling. 2008.

[BBNP04] Edmund Burke, Yuri Bykov, James Newall, and Sanja Petrovic. A time-predefined local search approach to exam timetabling problems. 2004.

[BM10] E. Burke and Barry McCollum, editors. *Springer Lecture Notes in Computer Science*. Springer-Verlag, 2010.

[BT95a] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. *TIK-Report*, 11, 1995.

[BT95b] Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. *Genetic Algorithms: Proceedings of the 6th International Conference*, 1995.

[CP95] E. Cantu-Paz. A summary of research on parallel genetic algorithms. Technical Report 95007, IlliGAL Report, 1995.

[FSAPMV08] Juan Frausto-Solis, Federico Alonso-Pecina, and Jaime Mora-Vargas. An efficient simulated annealing algorithm for feasible solutions of course timetabling. In *MICAI 2008: Advances in Artificial Intelligence*, volume 5317 of *Lecture Notes in Computer Science*, pages 675–685. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-88635-8.

[GD91] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, pages 69–93, 1991.

[Gee09] Zong Woo Geem, editor. *Music-Inspired Harmony Search Algorithm: Theory and Applications*. 2009.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[GS01] Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-42421-5.

[Hel04] Sabine Helwig. Erweiterung eines frameworks für zeitplanungsprobleme. Master's thesis, Universität Erlangen–Nürnberg, 2004.

[KH05] Graham Kendall and Naimah Mohd Hussin. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In *Multidisciplinary Scheduling: Theory and Applications*, pages 309–328. Springer US, 2005. ISBN 978-0-387-25266-7 (Print) 978-0-387-27744-8 (Online).

[Kil09] Helmut Killer. Entwurf und implementierung von algorithmen für zeitplanungsprobleme. Master's thesis, Universität Erlangen-Nürnberg, Germany, 2009.

[MFD07] M. Mahdavi, M. Fesanghary, and E. Damangir. An improved harmony search algorithm for solving optimization problems. 2007.

[MKM06] Muhammad Rozi Malim, Ahamad Tajudin Khader, and Adli Mustafa. Artificial immune algorithms for university timetabling. 2006.

[Sys91] G. Syswerda. A study of reproduction in generational and steady-state genetic algorithms. *Foundations of Genetic Algorithms*, pages 94–101, 1991.

[vL87] Peter J.M. van Laarhoven. *Simulated annealing*. D. Reidel Publishing Company, 1987.

[Whi89] Darrell Whitley. The genitor algorithm and selection pressure. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, 1989.

[Wil10] Peter Wilke. The Erlangen Advanced Time Tabling System (EATTS) Version 5. In Burke and McCollum [BM10], page submitted.

[WK10] Peter Wilke and Helmut Killer. Walk Up Jump Down - a new Hybrid Algorithm for Time Tabling Problems. In Burke and McCollum [BM10], page submitted.

Walk Down Jump Up - a new Hybrid Algorithm for Time Tabling Problems

Peter Wilke · Helmut Killer

Abstract A new trajectory time tabling algorithm is introduced. Because of it's nature the algorithm is called Walk Down Jump Up. The algorithm is described in a basic version and an advanced version. The performance of the algorithm when solving two real world problems is discussed.

Keywords timetabling · hybrid algorithm · time tabling algorithm · walk-down-jump-up

1 Introduction

Working with school or course time tabling problems leads to the usual "suspects" like Simulated Annealing, Tabu Search or Genetic Algorithms. If the achieved results are not acceptable then variants of these algorithms come into play trying to tailor the algorithm to fit the problem even better.

Here we would like to introduce a hybrid algorithm which is inspired by hill climbing, jumps and Simulated Annealing variants like Great Deluge [BBNP04].

Our algorithm is divided in two phases: first costs are reduced with a fast descending acceptance rate, second on stagnation the acceptance rate is increased at one go by a certain amount and phase one starts again and so on. That's why it is called Walk Down Jump Up.

Peter Wilke
Universitaet Erlangen-Nuernberg, Department Informatik, Martensstrasse 3, 91058 Erlangen, Germany
Ph.: +49 (9131) 85-27998
E-mail: Peter.Wilke@Informatik.Uni-Erlangen.de
Helmut Killer
E-mail: Helmut.Killer@gmx.de

2 Walk Down Jump Up Algorithm

Walk Down Jump Up is a trajectory based local search algorithm. Overall idea of the algorithm is based on changes of the acceptance rate. The idea for the algorithm comes on one hand from observing hillclimbing algorithm where only new solutions with better or equal costs than previous solution are accepted. Hillclimbing can reduce costs fast but then gets stuck in stagnation. On the other hand the idea is inspired by Great Deluge algorithm where new solutions are accepted when costs are below a falling cost limit. When the cost limit falls too fast then there will be stagnation, if it falls too slow no acceptable solution will be found in the estimated time. So our conclusion was that the acceptance rate should fall fast and on stagnation where all neighbour solutions to current solution have higher costs, it is necessary to set the acceptance limit to far above this surrounding cost values, so that it can jump up to a more far away solution and walk down to a new stagnation cost value that is hopefully lower than the stagnation cost before. The image in mind is that of the moon surface where many sinkholes exist and the task is to find one of the lowest. The astronaut walks down into the crater until lowest position is reached, then he jumps up above the border and tries to walk down into another crater which is hopefully deeper and so on.

2.1 Algorithm workflow

At the beginning an initial solution is generated randomly. At this stage the current best solution is equal to the initial solution and the jump distance is set to zero. Now a loop begins, if the solution isn't good enough. A new solution is generated by modifying the current solution using certain random move operators. If new solution has better or equal costs it is accepted and solutions with costs worse than the current solution are rejected. This leads typically to a local minimum and a stagnation phase. To get out of this trap the *jump operator* is enabled. When it is applied the threshold of accepted solution is increased from the costs for the current solution to a temporary limit. This limit is calculated by multiplying the current costs with an integer jump factor. After the Jump Operator is enabled the limit is decreased with every new iteration step by 1. If stagnation occurs again the current costs are compared with the costs of the last stagnation phase. If they are better, the solution is saved as new best solution, if they are worse but costs are in the same ball park as before, the search is continued according to the cost distance up to 10 times longer to finally yet get a better cost value. If the costs are still worse the jump factor is increased and the jump operator is applied again. Fig. 1 shows the control flow of the algorithm.

2.2 Random move operators

In each step the Walk Down Jump Up algorithm modifes current solution randomly to get a new solution with hopefully better costs. The initial solution is a random assignment of resources like students, teachers, rooms and timeslots to events. Modification of such solution means ressources are randomly assigned to or removed from some events, or directly exchanged. This leads to conflicts between the resources and

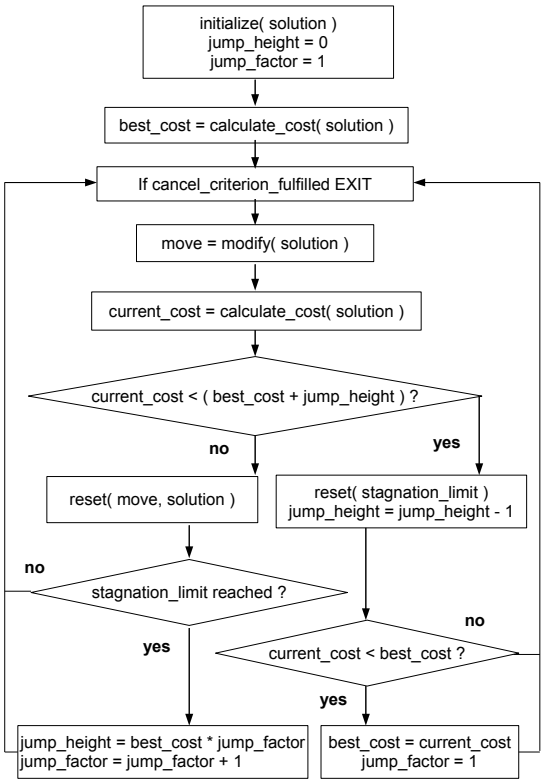


Fig. 1 Flow chart showing Walk Down Jump Up algorithm

violates constraints which increase solution costs or solves conflicts and satisfies constraints like "compact time table" which leads to lower costs. In one algorithm step only very few events are changed by a move operator and after cost evaluation the move is accepted or discarded.

2.3 Advanced Version - intelligent jump height

In the basic version of Walk Down Jump Up algorithm the jump height factor is increased by one after every walk down whose final cost is worse than before. This leads to more higher jumps after every algorithm walk down until new best cost is reached and jump height is reset to one. In the advanced version of the algorithm the jump height values which were successful are saved in a history list. When a new jump height has to be determined, the jump height factor is chosen randomly from the history list or a random value between 1 and the specified maximum jump height factor that is e.g. 10 times current cost value. The experiments show that the advanced version is able to reach better final costs than basic version.

3 Experiments and Results

Walk Down Jump Up was developed when we worked on two real world problems, namely school timetabling and univerity course time tabling. Walk Down Jump Up has been implemented using the current version of EATTS (Erlangen Advanced Time Tabling System [Wil10]). Because also many other algorithms like Simulated Annealing, Genetic Algorithm and Tabu Search are available in EATTS the algorithm performance can easily be compared.

3.1 The Problems

3.1.1 School 2009 Time Tabling

The data for our School 2009 time tabling problem represents an existing school with students from year 1 to 10. In this scenario the classes and their subjects are given, while class rooms and time slots have to be assigned to the events. Teachers can be assigned fixed to class/subject pairs, but don't have to. Here it is sufficient that one student represents the entire class, details given in table 1. Beside calculating a feasible solution the main constraint is to also generate a timetable as compact as possible.

Events:	178
Resources of type TimeSlot:	81
Resources of type Class:	14
Resources of type Teacher:	28
Resources of type Room:	37
Resources of type Subject:	78
Resources of type LessonProperties:	178

Table 1 The main characteristics of the School 2009 example

3.1.2 MuT 2009 courses at a university

Our university organises a girl-and-technology (in german: "Maedchen und Technik", abbreviated MuT) week each year to attract more female students to technical subjects. In this scenario the tutors and time slots for the events are fixed while students (not classes) have to be assigned to the project of their choice, details given in table 2. Each student has a list of 4 preferred courses and can declare 4 friends with whom she wants to share the same courses.

Events:	229
Resources of type TimeSlot:	57
Resources of type Subject:	52
Resources of type Girl:	170

Table 2 The main characteristics of the MuT 2009 example

3.2 Results

Figure fig. 2 shows the performance when solving the MuT 2009 problem. The red line indicates the current best solution, while the green dots show current solution, which can be worse then the best solution because of the jump operator and/or the random changes made to find new solutions. When the runtime is limited to 1 second Walk Down Jump Up shows a fast descent of costs followed by a long phase of slow descent. When applied to the MuT 2009 problem Walk Down Jump Up performed equally good to Simulated Annealing [vL87, FSAPMV08] and slightly better than Tabu Search [GS01, KH05] and Immune Systems [MKM06].

On the School 2009 problem Walk Down Jump Up achieved the best results together with Harmony Search [ABKG08, Gee09], Simulated Annealing, Great Deluge [BBNP04] when long runs, i.e. several hours, are evaluated. On short runs, i.e. several seconds, Walk Down Jump Up was head on with Tabu Search and Great Deluge, but all three were outperformed by Simulated Annealing. For a more detailed comparison of Walk Down Jump Up with six other algorithms see [WK10]

4 Conclusion

Walk Down Jump Up has proven that it is able to solve real world problems efficient and effective. But the results have also shown that there is still room for improvements.

References

[ABKG08] Mohammed Azmi Al-Betar, Ahamad Tajudin Khader, and Taufiq Abdul Gani. A harmony search algorithm for university course timetabling. 2008.

[BBNP04] Edmund Burke, Yuri Bykov, James Newall, and Sanja Petrovic. A time-predefined local search approach to exam timetabling problems. 2004.

[BM10] E. Burke and Barry McCollum, editors. *Springer Lecture Notes in Computer Science*. Springer-Verlag, 2010.

[FSAPMV08] Juan Frausto-Solis, Federico Alonso-Pecina, and Jaime Mora-Vargas. An efficient simulated annealing algorithm for feasible solutions of course timetabling. In *MICAI 2008: Advances in Artificial Intelligence*, volume 5317 of *Lecture Notes in Computer Science*, pages 675–685. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-88635-8.

[Gee09] Zong Woo Geem, editor. *Music-Inspired Harmony Search Algorithm: Theory and Applications*. 2009.

[GS01] Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-42421-5.

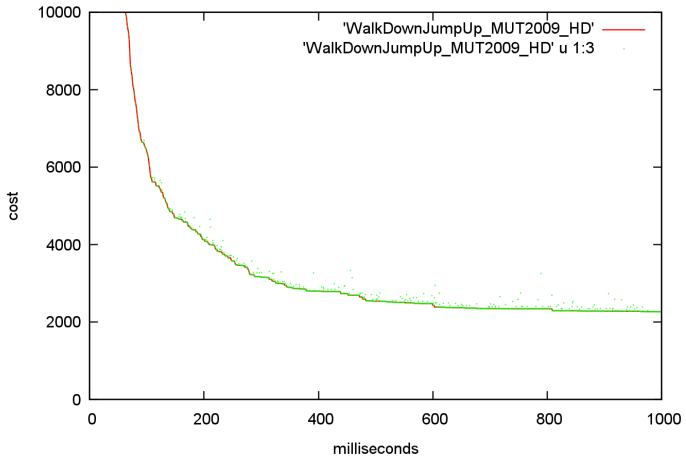
[KH05] Graham Kendall and Naimah Mohd Hussin. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In *Multidisciplinary Scheduling: Theory and Applications*, pages 309–328. Springer US, 2005. ISBN 978-0-387-25266-7 (Print) 978-0-387-27744-8 (Online).

[MKM06] Muhammad Rozi Malim, Ahamad Tajudin Khader, and Adli Mustafa. Artificial immune algorithms for university timetabling. 2006.

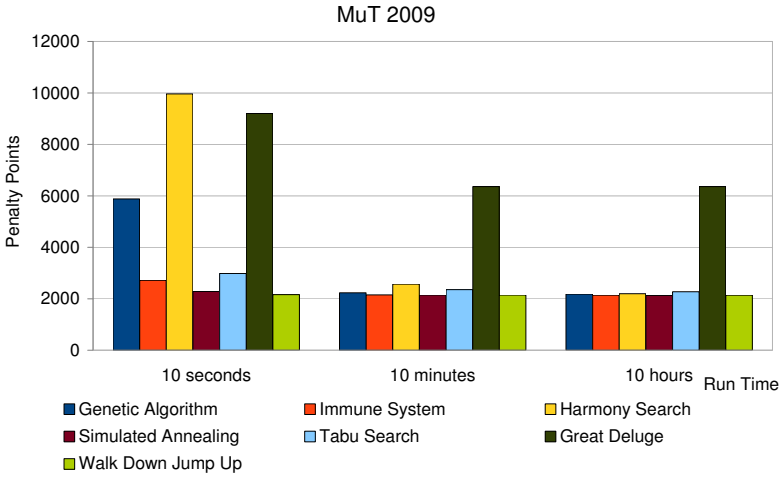
[vL87] Peter J.M. van Laarhoven. *Simulated annealing*. D. Reidel Publishing Company, 1987.

[Wil10] Peter Wilke. The Erlangen Advanced Time Tabling System (EATTS) Version 5. In Burke and McCollum [BM10], page submitted.

[WK10] Peter Wilke and Helmut Killer. Comparison of Algorithms solving School and Course Time Tabling Problems using the Erlangen Advanced Time Tabling System (EATTS). In Burke and McCollum [BM10], page submitted.

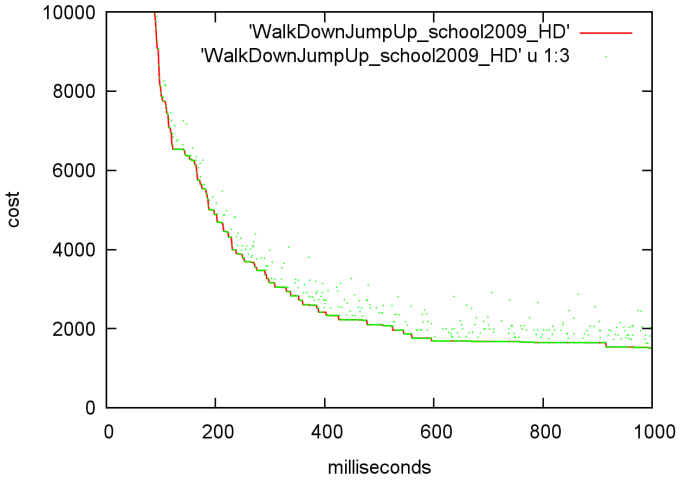


(a) Walk Down Jump Up on MuT 2009 with runtime 1 second

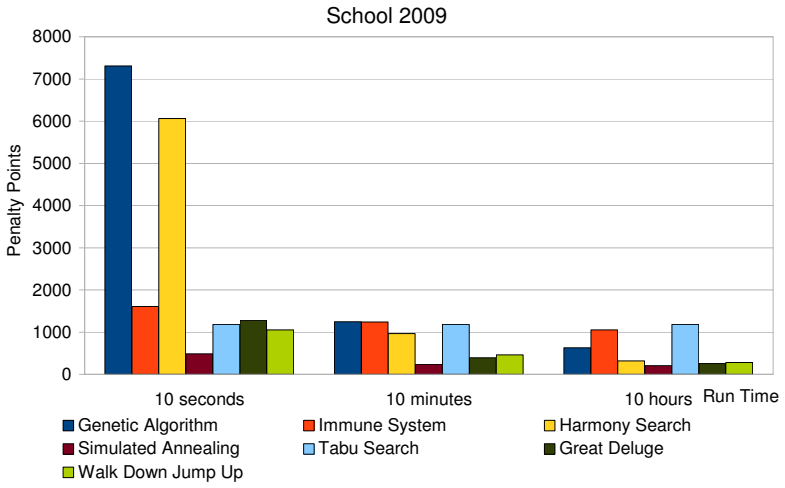


(b) Walk Down Jump Up on MuT 2009 with runtimes 10 second, minutes, hours

Fig. 2 Walk Down Jump Up solving the MuT 2009 example



(a) Walk Down Jump Up on School 2009 with runtime 1 second



(b) Walk Down Jump Up on School 2009 with runtimes 10 second, minutes, hours

Fig. 3 Walk Down Jump Up solving the School 2009 example

The Erlangen Advanced Timetabling System (EATTS) Unified XML File Format for the Specification of Timetabling Systems

Johannes Ostler · Peter Wilke

Abstract It would be nice if all timetabling problems could be described in a canonical and universal description language with standardized syntax. If such a description would be available algorithms and/or frameworks could be used to interpret and/or compile these descriptions, and solve the problem at stake. And systems, algorithms, problems, etc. could be compared in a simple way.

Here we would like to suggest to step towards a standard, a XML format to define a great number of different problems like Course Timetabling, High School Timetabling, Employee Timetabling et cetera. In addition tools to edit these descriptions and to apply algorithms to solve the described problem.

Keywords EATTS · Erlangen Advanced Timetabling System · XML · Benchmarks · File Format · Standard · Timetabling Systems · Timetabling Problems

1 Introduction

A common and widely accepted description format for timetabling problems and its solution would be a great help when algorithms and/or frameworks are compared. When looking at the literature there are description formats but they are too specific and can't be used for general problems. The narrow focused system evolved due to historic reasons, specific domain properties or efficiency reasons, e.g. very often these description are incomplete because part of the problem is hard coded in the implementation. In the year 2003 our research group published a approach for a unified timetabling framework and format [GWB03]. In the following years we continued this work and would like to suggest the next version of our description format.

Johannes Ostler
Universitaet Erlangen-Nuernberg, Informatik 5, Martensstrasse 3, 91058 Erlangen, Germany
Tel.: +49 (9131) 85-27998
E-mail: Johannes.Ostler@Informatik.Uni-Erlangen.DE

Peter Wilke
Universitaet Erlangen-Nuernberg, Informatik 5, Martensstrasse 3, 91058 Erlangen, Germany
Tel.: +49 (9131) 85-27998
E-mail: Peter.Wilke@Informatik.Uni-Erlangen.DE

2 Existing XML format for High School Timetabling Problems

There are some approaches on standardization of formats to define timetabling problems of a certain problem type. Especially the standardized format for benchmarks in High School Timetabling [PAD+08] is interesting. In that paper a way is shown to give a unified format for High School Timetabling problems of different countries. The purpose is to provide a format for benchmark data interchange between independent groups of researchers. "The students are organized in base groups, and follow lessons. The lessons of a (usually fixed) group of students in the same subject (like math, arts, ...), usually with the same teacher, we will call a course."[PAD+08, p. 3] Students, teachers and rooms are the three main groups of resources. Moreover there are assets, which are organized in "four categories: Time, Courses, Subjects, and Resources"[PAD+08, p. 7]. Every time slot has a sequence number, the day and optional the week is attached. "A course is a collection of events that involves a group of students and a subject. Events that refer to the same course, are called lessons of the course. ... A resource is an entity with time restrictions. The most common resources are the students, teachers, and rooms. ... Events are the basic scheduling object. An event can represent either a single lesson, or a set of lessons, that have to be taught at the same time."[PAD+08, p. 8f] There are a set of standardized constraints, which can be referenced in the XML. Their parameters can be set via XML tags.

The structure of the XML is:

```
<Problem>
  <Instance>
    <Assets> ... </Assets>
    <Events> ... </Events>
    <Constraints> ... </Constraints>
  </Instance>
  <Solutions> ... </Solutions>
</Problem>
```

Listing 1 Structure of High School Timetabling XML [PAD+08, p. 11f]

The `Solutions` tag includes an `Event` tag per event which stores information about the assignments of this event. Information about the costs and/or the constraint violations also can be included.

3 Benefits of a new XML format

The existing formats can only describe a special class of timetabling problems. But the need for flexibility requires to categorize and compare many different problems. One approach would be to adapt our solver so that it can read many input file formats. But if there is a unified format other research groups can use our benchmark files, too.

We decided to create a XML format for the following reasons. First of all XML is a standard of information interchange using internet and good support to handle easily XML data is available in most computer languages. XML is a structured format, the syntax can be defined by a Document Type Definitin (DTD) [wik10] or a XML Schema Definition (XSD) [W3C10a]. At least XML is easy to understand and many software engineers are familiar with it. More information about XML can be found on the websites of W3C® [W3C10b].

4 The EATTS XML File Format for Problem Definitions

4.1 Entities of a Timetabling Problem

Descriptions start with the specification of the entities of a typical Timetabling Problem. In many cases a set of time slots has be assigned to a set of events. Usually this assignment is subject to some constraints. In some cases other resources are not assigned fixed to the events. For example in Course Timetabling exactly one room has be chosen from a set of rooms.

Our approach to describe different Timetabling problems distinguishes between three main entity types: resources (including time slots), events and constraints. The solution of a Timetabling problem is the assignment of resources of different resource types to the events. Solution have assigned costs depending on the violated constraints.

In our approach the description consists of a model part and a data part. While the model takes care of the general structure of the problem at hand the data part takes care of the individual components of the problem. For a School Timetabling Problem the model would contain the resource types like teachers and rooms, while the data part would contain the individual teacher with names, hours per week, subjects, etc.

4.2 RECPlan

In the EATTS XML format a Timetabling Problem is declared by a `RECPlan` tag. REC stands for Resources, Events and Constraints. The typical structure of this tag is shown in the listing below.

```
<RECPlan name="nameOfThePlan" planVersion="0">
  <TimeFrame ... />
  <ResourceTypes> ... </ResourceTypes>
  <ResourceCollections> ... </ResourceCollections>
  <Resources> ... </Resources>
  <Events> ... </Events>
  <EventCollections> ... </EventCollections>
  <Constraints> ... </Constraints>
</RECPlan>
```

Listing 2 Structure of RECPlan

The `TimeFrame` tag defines the time frame of the planning period.

The defintion of resource types starts with the tag `ResourceTypes`. Resources of a resource type can be subdivided into collections which are defined in the tag `ResourceCollections`. The same mechanism is available to describe event types and event collections. To specify the resources the tag `Resources` is used, similar for events the tag `Events`.

The constraints have there own part in the description starting with the tag `Constraints`.

4.3 TimeFrame

The planning period of a Timetabling Problem starts at absolute date *start*. All other points in time and intervals of time are declared relatively to this date. Because of the variety of timetabling problems it is necessary to define a time grid. Typically schedules

are represented as a table. Often the columns represents one day and the lines one hour or parts of one hour. The time which lies between the first point in time of column i and this of column $i+1$ is called the horizontal scaling hs. The vertical scaling vs is the minimal timespan between two vertical lines. The value of the time t is given relatively to $start$, so $t_{abs} = t + start$. A time slot (s, d) starts at the absolute point in time $t_{start} = s + start$ and ends at $t_{end} = t_{start} + d$.
The following condition must be true to ensure that all sub-units can be mapped correctly to the given time line.

$$\{hs, t, s, d\} \subset \{x = n \cdot vs \mid n \in \mathbb{N}\} \tag{1}$$

```
<TimeFrame absoluteStart="1253526334272" horizontalScaling="86400000"
           verticalScaling="900000" cycleLength="7" repeatNCycles="1"/>
```

Listing 3 Structure of RECPlan

The attributes *horizontalScaling* and *verticalScaling* determine the horizontal and vertical scaling. The unit of time values is milliseconds. The absolute start *absoluteStart* is given as a Unix timestamp value. Unix time stamps are widely supported, even on windows systems, e.g. Java VM. The period to be planned is typically a multiple of hs, typical periods are a week respectively seven days (see above). In that case the attribute *cycleLength* has the value seven. For School Timetabling the schedule of a standard week will be valid for the following x weeks. So the attribute *repeatNCycles* will have the value x.

4.4 Resource Types

Every family of Timetabling Problems has the same types of resources. For example in Course Timetabling there are teachers, rooms, classes, students, and subjects, while for rosters the resource types are employees, machines et cetera. The description starts with the definition of resource types and after that the resources of each resource type are defined. Every resource type has a denomination like *Teacher* or *Employee* and a set of attributes. The definition of an attribute includes the name and the type of the attribute's value.

```
<ResourceTypes>
  <Type name="TimeSlot">
    <Attribute name="Slot" typeKind="S" typeDenom="TimeSlot" isList="false"/>
  </Type>
  <Type name="Room">
    <Attribute name="Name" typeKind="S" typeDenom="String" isList="false"/>
    <Attribute name="Size" typeKind="S" typeDenom="Integer" isList="false"/>
  </Type>
  ...
</ResourceTypes>
```

Listing 4 Example of a ResourceTypes tag

The `Type` tag defines a resource type. The attribute *name* specifies its name. This tag includes a set of `Attribute` tags, which represents the attribute definitions of the resource type. The attribute *typeKind* typecasts the type of the attribute value. Possible types are "S" for simple, "G" for group, "R" for resource and "E" for event. The type simple includes elementary values like integers, floats, strings or time slots.

The attribute *typeDenom* specifies this type more precisely. The attribute values can be single values or a list of values. This property will be defined by the attribute *isList*.

4.5 ResourceCollections

Resources of a certain type can be grouped in collections. There are four different kinds of collections.

1. Simple groups are called resource group. A resource group includes a set of resources of a certain resource type. The relationship between a resource and a resource group is n : m. Every resource can be added to any number of collections and vice versa.
2. The union of the elements of a set of collections can be defined by the `Union` tag.
3. The `Intersection` specifies the intersection of the elements of a set of collections.
4. If the set R_{rt} is the set of all resources of the resource type rt and G_{rt} is any resource collection of the resource type rt, then $INV(G_{rt}) = \{r \in R_{rt} | r \notin G_{rt}\}$ is the absolute complement of G_{rt}. The tag `Inverter` defines such a complement of a resource collection.

```
<ResourceCollections>
  <ResourceCollectionsByType typeName="TimeSlot">
    <Group name="MorningSlots"/>
    <Inverter name="AfternoonSlots">
      <CollectionRef name="MorningSlots"/>
    </Inverter>
    <Union name="AllSlots">
      <CollectionRef name="MorningSlots"/>
      <CollectionRef name="AfternoonSlots"/>
    </Union>
  ...
  </ResourceCollectionsByType>
  ...
<ResourceCollections>
```

Listing 5 Example of a ResourceCollections tag

All collections of a plan are enclosed in `ResourceCollections` tag. The `ResourceCollectionsByType` tag is a container for all definitions of collections of a certain resource type. The attribute *typeName* specifies the corresponding resource type by its name.

A simple collection of resources is defined by a `Group` tag. The *name* attribute determines the denomination of the collection. This name must be unique for all collections of the same type. This tag does not include any information regarding which resources are members of the group. A `CollectionRef` tag references a collection by its name. A union or intersection can include any number of other collections while an inverter can include only one.

4.6 Resources

The resource definitions are encapsulated in `Resources` and `ResourcesByType` tags following the collections. The `Resource` tag defines one resource. Every resource has its unique id which is declared in the *id* attribute. Enclosed in a `AttributeValues` tag the `AttributeValue` determines the value of the attribute which is referenced by the value of the *name* attribute. The resource defines to which resource groups it is assigned to.

```
<Resources>
  <ResourcesByType typeName="Class">
    <Resource id="Class0">
      <AttributeValues>
        <AttributeValue name="classRoom">
          <ResourceRef id="Room16" resourceType="Room"/>
        </AttributeValue>
        <AttributeValue name="name">
          <String>1A</String>
        </AttributeValue>
        <AttributeValue name="grade">
          <String>1</String>
        </AttribteValue>
        <AttributeValue name="teacher">
          <ResourceRef id="Teacher7" resourceType="Teacher"/>
        </AttributeValue>
        <AttributeValue name="group">
          <ResourceCollectionRef name="students 1A" resourceType="Student"/>
        </AttributeValue>
      </AttributeValues>
      <ContainingGroups>
        <CollectionRef name="sports male of grade 1"/>
        <CollectionRef name="sports female of grade 1"/>
      </ContainingGroups>
    </Resource>
  </ResourcesByType>
</Resources>
```

Listing 6 Example of `Resources` tag

The values of the attributes are enclosed in tags which depend on the type of the value. Possible Tags and their meaning are listed in the following table.

Value Tag	Type of Value
String	string literals
Integer	integer numbers
Float	floating-point numbers
Boolean	boolean values
Time	relative time values
TimeSlot	relative time spans
ResourceRef	references to resources
ResourceCollectionRef	references to a resource collection
EventCollectionRef	references to a event collection

Table 1 Value types of attributes

4.7 Events

The planning process is all about events. The time table - as a result of the planning process - determines which resource is assigned to which event. Below rt a resource type and e an event of the plan is shown. For every event it can be declared how much resources of a certain resource type have to be assigned as minimal $min_{e,rt}$ or maximal $max_{e,rt}$ value so that the resulting time table is a valid solution. Two additional subsets of resources of this type can be defined: one which is assigned fixed to this event and/or one subset of resources which could be assigned, at least one of them must exist.

Let $F_{e,rt}$ and $O_{e,rt}$ be a subset of R_{rt}. $F_{e,rt}$ is the set of resources of the resource type rt which is assigned fixed to the event e and $O_{e,rt}$ is the set of resources from which the planning component can select. Let s be a result of the timetabling problem, so $s(e, rt)$ is the set of resources of resource type rt which are assigned to the event e and $s(r)$ is the set of events assigned to the resource r.
The following must be true:

$$F_{e,rt} \subset R_{e,rt} \subset F_{e,rt} \cup O_{e,rt} \tag{2}$$

The following condition should be true to avoid constraint violations.

$$min_{e,rt} \leq |R_{e,rt}| \leq max_{e,rt} \tag{3}$$

```
<Events>
  <Event name="5A_Maths">
    <ResourceLists>
      <ResourceList resourceTypeName="TimeSlot" max="4" min="4" fixed="true">
        <OptionalGroup name="MorningSlots"/>
      </ResourceList>
      <ResourceList resourceTypeName="Class" max="1" min="1" fixed="true">
        <FixedGroup name="class 5A"/>
      </ResourceList>
    ...
      <ResourceList resourceTypeName="Teacher" max="1" min="1" fixed="true">
        <OptionalGroup name="teachers of 5A_Maths"/>
      </ResourceList>
      ...
   </ResourceLists>
   <ContainingGroups>
    <EventCollectionRef name="Maths Lessons">
    <EventCollectionRef name="Lessons of 5A">
   </ContainigGroups>
  </Event>
</Events>
```

Listing 7 Example of `Events` tag

The `Event` tag defines an event. Its name is set by the *name* attribute. For every resource type this tag includes a `ResourceList` tag, which defines $min_{e,rt}$ by the attribute *min* or $max_{e,rt}$ by *max*. $F_{e,rt}$ is declared by the tag `FixedGroup`, $O_{e,rt}$ by the `OptionalGroup` tag. If one of these group tags is missing the corresponding group is believed to be an empty set, but at least one of them must be not empty. The sets are referenced by the name of the group, which is stored in the *resourceTypeName* attribute. Just like the resources events can also be classified into collections. The `EventCollectionRef` tags in the `ContainingGroups` tag determines to which event group this event is assigned to.

The assignments of resources of a certain type rt to a set of events $\{e_1, ...e_n\}$ can be linked. That means that $s(e_1, rt) = s(e_2, rt) = ... = s(e_n, rt)$. For this purpose the attribute *baseEvent* must be added to the `ResourceList` tags of $\{e_2, ...e_n\}$. The value of this attribute is the name of e_1.

4.8 Constraints in general

In the preceding sections a unified model for defining different Timetabling Problems was described. Now it's time to develop a uniform way to declare constraints. First of

all lets have a look at different solutions s_i of the same problem. A constraint measures the quality of a solution from a certain point of view. It defines a cost function $q_c(s_i)$. The codomains of these functions are subsets of $\mathbb{R}_0^+$. Let C be the set of all constraints of a problem. The quality of a solution s_i is defined as $Q(s_i)$.

$$Q(s_i) = \sum_{c \in C} q_c(s_i). \tag{4}$$

The only aspect in which the solutions s_k and s_l can differ are the different assignments of resources to the events. If the costs of a constraint does not depend on these assignments this means that they are fixed costs for all solutions and can't be reduced by any planning algorithm. For simplicity they should not be part of the description, i.e. only variable costs should be defined in the model.

A constraint is defined by its scope $scope_c$ and a function $q_c(s, y)$, whereby s is a solution and y is an element of the scope. It is typical that the constraints differ in weight. So a weight function $w_c : \mathbb{N} \to \mathbb{R}$ must be defined for every constraint c. That way there must be a function $viol_c(s, y) : S \times scope_c \to \mathbb{N}$ which counts the number of violations against constraint c in the solution s on the view of y.

A view of a resource or event can be understood as looking at the solution showing only the parts where this resource or event is involved. E.g. a student wants to see his personal time table and not the whole plan of the entire school.

$$q_c(s, y) = w_c(viol_c(s, y)) \tag{5}$$

$$q_c(s) = \sum_{y \in scope_c} q_c(s, y) \tag{6}$$

```
<Constraints>
  <OperationLists> ... </OperationLists>
  <Selectors> ... </Selectors>
  <Variables> ... </Variables>
  <TimeClashConstraint>
    <ConstraintProperties name="TimeClash for Teacher" type="hard">
      <PenaltyFunction weight="50.0" coefA="1.0" coefB="0.0" exponent="2.0"/>
      <Description>
        No teacher can give more than one lesson at the same point in time.
      </Description>
    </ConstraintProperties>
    <ResourceTuple>
      <ResourceCollectionRef name="Teacher_All" resourceType="Teacher"/>
    </ResourceTuple>
  </TimeClashConstraint>
</Constraints>
```

Listing 8 `Constraints` tag

The definitions of the constraints are enclosed in the `Constraints` tag. To define standardized constraints we use lists of operations, selectors and variables. These entities will be explained in the sections 4.9, 4.10, and 4.12 below. Now the typical structure of a constraint definition is shown using the example of a `TimeClashConstraint` tag. The constraint of the example is violated if one teacher gives more than one lesson at the same time. The `ConstraintProperties` tag declares all properties common to all types of constraints. Every constraint has a denomination given by the *name* attribute and a *type*. In our description language constraints come in three different flavours: the familiar hard and soft constraints and in addition soft hard constraints. In normal mode the latter are considered as hard constraints but in exception mode - when no

feasible solution could be found in normal mode - they are considered as soft constraints. This eases the burden on the planning algorithm. Every constraint c must have a weight function w_c. We use a standardized weight function which depends on four float parameters: weight, coefA, coefB and exponent. These parameters are set by the corresponding attributes of the `PenaltyFunction` tag.

$$w_c(x) = \begin{cases} weight \cdot (coefA \cdot x^{exponent} + coefB) & if \quad x \in \mathbb{N}^+ \\ 0 & \quad\quad\quad x = 0 \end{cases} \tag{7}$$

The `Description` offers the opportunity to document additional information.

The `ResourceTuple` tag declares that this constraint must be evaluated for every resource of the resource type *Teacher*. The view of a constraint definition can be one of the following: the view of an event, the view of a relation between events or the view of one resource. An example for the latter is the view of a teacher. So there are three different main types of constraints in the model: event, event relation and resource constraints.

4.9 Event Constraints

An event constraint ec measures a solution s from the view of a single event e. The costs of the event e of ec for the solution s are defined by the function $viol_{ec}(s, e)$. Every event constraint has a scope $Scope_{ec} \subset E$.

Let's have a look at the declaration of $viol(s, e)$. Let $A(s, e)$ be the assigned resources of event e in the solution s. The violation count function $viol(s, e)$ indicates if the constraint is violated in regards to event e in the context of solution s. Typically $viol(s, e)$ is composed of three parts: two operations $evar_i(A(e))$ and $evar_j(A(e))$ and a relational operator rop. An event variable $evar$ is composed of three parts too: a selection function $sel(A(e)) : A(s, e) \rightarrow s(e, rt_i) \subset R_{rt_i}$, an attribute value selection function $attr : R_{rt_i} \rightarrow V$, and an operation $op : V \rightarrow V'$. Hereby V and V' are sets of lists of attribute values. The function rop maps the cross product $V'_l \times V'_r$ into the set of natural numbers.

$$evar(A(s, e)) = op(attr(sel(A(s, e))) \tag{8}$$

$$viol_{ec}(s, e) = rop(evar_i(A(s, e)), evar_j(A(s, e))) \tag{9}$$

An example of an event constraint if the Room Size Constraint which is violated if the capacity of one of the assigned rooms is less than the number sc of assigned students.

$$viol^*_{ec}(e, s, r) = \begin{cases} 1 & if \quad capacity(r) < sc \\ 0 & else \end{cases} \tag{10}$$

$$viol_{ec}(e, s) = \sum_{r \in s(e, Room)} viol^*_{ec}(e, s, r) \tag{11}$$

```
<Selectors>
  <EventSelector name="Room Capacity Selector" lookAtEveryResourceSingle="
      true">
    <TupleValue type="Room" attribute="Capacity"/>
  </EventSelector>
```

```
    <EventSelector name="Student Count Selector" lookAtEveryResourceSingle="
        false">
      <TupleValue type="Student"/>
    </EventSelector>
  </Selectors>
```

Listing 9 Selectors of Room Size Constraint

The *Room Capacity Selector* returns the capacity values of the assigned rooms. Because *lookAtEveryResourceSingle* has the value *true* the capacity of every assigned room must be compared with the right operand. The *Student Count Selector* collects all assigned students into a list of resources. Because only the count of students matters no attribute value is selected.

```
<OperationLists>
  <OperationList name="List Length">
    <Operation>list_size</Operation>
  </OperationList>
</OperationLists>
  ...
<Variables>
  <EventVariable name="Student Count" typeDenom="Integer">
    <EventSelectorRef name="Student Count Selector"/>
  <OperationListRef name ="List Length"/>
  </EventVariable>
  <EventVariable name="Room Capacity" typeDenom="Integer">
    <EventSelectorRef name="RoomCapacitySelector"/>
  </EventVariable>
</Variables>
```

Listing 10 Variables of Room Size Constraint

The values returned by the selectors will be handled by the variables. *Student Count* returns the size of the students list of the *Student Count Selector*, hereby it uses the operation list *List Length*.

```
  <EventConstraint relOpDenom="geq">
     <ConstraintProperties name="Room size" type="soft" slopeType="split">
       <PenaltyFunction weight="50" coefA="1" coefB="2" exponent="1"/>
       <Description>The room size must be >= student count</Description>
     </ConstraintProperties>
     <EventVariableRef name="Room Capacity"/>
     <EventOperand>
       <EventVariableRef name="Class Capacity"/>
     </EventOperand>
     <CollectionRef name="Events_All"/>
   </EventConstraint>
```

Listing 11 Room Size Constraint

The relational operator $\geq$ is declared by the attribute *relOpDenom*. There is a list of different relational operators which can be referenced by name. The `CollectionsRef` tag defines the scope of the constraints, in this case all events. The event operand can also be a constant.

A special type of event constraints is the `MinMaxConstraint`. The scope of this constraint is a set of events and the constraint is defined for a certain resource type rt. The resource list, which is associated with event e and resource type rt, defines a minimal $min(e, rt)$ and maximal $max(e, rt)$ count of resources giving the interval in which the number of resources assigned to event e should lie.

$$viol_{minmax\_rt}(e) = \begin{cases} min_{e,rt} - |s(e,rt)| & if \quad |s(e,rt)| < min_{e,rt} \\ |s(e,rt)| - max_{e,rt} & if \quad |s(e,rt)| > max_{e,rt} \\ 0 & else \end{cases} \tag{12}$$

```
<MinMaxConstraint type="minmax">
  <ConstraintProperties name="TimeSlot MaxConstraint" type="hard">
    <PenaltyFunction weight="100.0" coefA="1.0" coefB="2.0" exponent="0.0"/>
    <Description/>
  </ConstraintProperties>
  <ResourceTypeRef name="TimeSlot"/>
  <CollectionRef name="All"/>
</MinMaxConstraint>
```

Listing 12 MinMaxConstraint tag

The attribute *type* defines which limit must be checked. Possible values are '*min*' (only the minimal limit), '*max*' (only the maximal limit), and '*minmax*' (check both limits). The tags `ResourceTypeRef` references the resource type rt, and `CollectionRef` defines the scope of the constraint.

4.10 Resource Constraints

The view of a resource constraint is the view of a single resource or a tuple of resources. A special example of the first case is shown above with the *TimeClashConstraint*. If a student *stud* can choose between courses of the same subject *sub*, there will be a constraint necessary to ensure that the student will participate only at one of these courses. One possibility to express this constraint is to define a maximal number of events to which both (*stud* and *sub*) are assigned to.

In this case the view of this constraint are tuples $(stud, sub) \quad whereby \quad stud \in R_{Student} \quad and \quad sub \in R_{Subject}$. Below the definition of a resource constraint by the example of a Workload Constraint is given. Every teacher has an attribute *workload* and this limit should not be exceeded by the generated solution.

A resource tuple selector *rts* maps the set of all resources to a set of resource tuples. The codomain $Codom(rts)$ of *rts* is a subset of $T_{rt_1,...,rt_n}$ the set of all tuples of resources of the resource types $rt_1, ..., rt_n$.

$$\begin{gathered} Codom(rts) \subseteq T_{rt_1,...,rt_n} \\ T_{rt_1,...,rt_n} = \{tuple \in R_{rt_1} \times ... \times R_{rt_n} : rt_i \neq rt_j \quad \forall i,j \in [1,n] \wedge i \neq j\} \end{gathered} \tag{13}$$

Let tuple $t = (r_1, ..., r_n)$ be an element of $T_{rt_1,...,rt_n}$. There are two ways to define the associated events, on the one hand the intersection way $IS(t)$, on the other hand the union way $U(t)$.

$$IS(t) = \{e \in E \mid e \in s(r_i) \quad \forall i \in [1,n]\} \tag{14}$$

$$U(t) = \{e \in E \mid (\exists i \in [1,n] : e \in s(r_i))\} \tag{15}$$

A resource selector *rs* includes a resource tuple selector rts_{rs} and defines whether US or IS will be applied. The resource selector rs_{rc} of a resource constraint rc defines the scope $scope_{rc}$.

$$scope_{rc} = Codom(rts_{rc}) \tag{16}$$

Let $t \in scope_{rc}$ be a tuple.

$$rs(t) = \begin{cases} IS(t) \\ U(t) \end{cases} \tag{17}$$

```
<Selectors>
 <ResourceSelector name="Events of a Teacher" unionType="Union">
  <ResourceTuple>
   <ResourceCollectionRef name="Teacher_All"
                          resourceType="Teacher" type="elem"/>
  </ResourceTuple>
 </ResourceSelector>
 <ResourceSelector name="Student x Subject" unionType="Intersection">
  <ResourceTuple>
   <ResourceCollectionRef name="Student_All" resourceType="Student"
                          type="elem"/>
   <ResourceCollectionRef name="mainSubjects"
                          resourceType="Subject" type="group"/>
  </ResourceTuple>
 </ResourceSelector>
</Selectors>
```

Listing 13 Resources Selectors

Events of a teacher represents the view of one teacher and returns whose assigned Events. The second resource selector $Student \times Subject$ is a little bit more complex. Its view is a tuple of a student and a set of subjects *subjects*. If the attribute *type* is '*group*' the tuples will be generated with a set of all resources of the group. An example is the group of all main subjects *mainSubjects*. So tuples between all students of the collection *Student_ALL* ('*elem*') and the collection *mainSubjects* ('*group*') can be generated. The return values are the events to those the student and at least one subject of *subjects* is assigned to. IS is selected because the *unionType* is set to *Intersection*.

A resource variable rv is based on a resource selector rs_{rv}. It can be expanded by an event variable ev_{rv} and an operation list ol_{rv}. Every resource variable rv_{rc} maps a tuple $t \in scope_{rc}$ to a value $rv_{rc}(t)$. This value is the left operand of $viol_{rc}(t)$.

$$EV_{rv}(x) = \begin{cases} ev_{rv}(x) & \text{if } ev_{rv} \text{ is set} \\ x & \text{else} \end{cases} \tag{18}$$

$$OL_{rv}(x) = \begin{cases} ol_{rv}(x) & \text{if } ol_{rv} \text{ is set} \\ x & \text{else} \end{cases} \tag{19}$$

$$rv_{rc}(t) = OL_{rv}(EV_{rv}(rs_{rv}(t))) \tag{20}$$

```
<Constraints>
  <OperationLists>
    <OperationList name="Duration"> ... </OperationList>
  </OperationLists>
  <Selectors>
    <EventSelector name="timeSlotsSelector">
      <TupleValue type="TimeSlot" attribute="slot"/>
    </EventSelector>
    ...
  </Selectors>
  <Variables>
    <EventVariable name="timeSlots">
      <EventSelectorRef name="timeSlotsSelector"/>
```

```
    </EventVariable>
    <ResourceVariable name="Workload of Teacher">
      <ResourceSelectorRef name="Events of a Teacher"/>
      <OperationListRef name="Duration"/>
      <EventVariableRef name="timeSlots"/>
    </ResourceVariable>
  </Variables>
```

Listing 14 Variables of Work Load of a Teacher

The event variable *timeSlots* maps the list of events given by the resource selector *Events of a Teacher* to a list of time slots. The operation list *Duration* computes the sum of durations of these time slots. This is the result of the evaluation of the resource variable *Workload of Teacher*.

A resource constraint rc consists of a relational operator rop, a resource variable rv as left operand and a right operand $right$. The scope of the constraint is defined by rs_{rv}. The violation count function $viol_{rc}(s, tuple)$ is defined by the following equation:

$$viol_{rc}(tuple) = rop(rv(tuple), right(tuple)); \tag{21}$$

$$q_{rc}(s) = \sum_{tuple \in scope_{rc}} w_{rc}(viol_{rc}(s, tuple)) \tag{22}$$

The operand on the right hand side of a resource constraint can be a constant value. This value has one of the types which are available for the definition of attributes. In some cases it is necessary to refer to an attribute value $attr$ of a resource r of the tuple $tuple$.

```
<ResourceConstraint relOpDenom="DISTTIME">
  <ConstraintProperties name="Work load of a teacher" type="soft">
    <PenaltyFunction weight="50.0" coefA="1.0" coefB="2.0" exponent="1.0"/>
    <Description>A teacher should work the given work load </Description>
  </ConstraintProperties>
  <ResourceVariableRef name="workloadOfTeacher"/>
  <ResourceOperand>
    <TupleValue type="Teacher" attribute="workload"/>
  </ResourceOperand>
</ResourceConstraint>
```

Listing 15 Work Load of a Teacher

4.11 Event Relation Constraints

In some cases the view of one event or one resource tuple is not sufficient to express all constraints involved. An example is the Same Resource Constraint which is violated if for a certain resource type rt $s(e_1, rt) \neq s(e_2, rt)$. This relation is postulated for a the resource type rt and a set of events E_{rel} with $e_1 \in E_{rel} \wedge e_2 \in E_{rel}$. An event relation constraint erc sets two collections of events E_{left} and E_{right} in a relation. The scope of this constraint is $scope_{erc} = E_{left} \times E_{right}$. So the owner of a constraint violation is a tuple of events $s(e_l, e_r)\, with\, e_l \in E_{left} \wedge e_r \in E_{right}$. The parameters of the violation count function $viol_{erc}$ are this tuple and the solution s. As said before the tuple consists of two parts e_l and e_r. For each of this parts a reference to an event variable ev_{left} resp. ev_{right} is defined. And a relational operator rop_{erc} must be defined.

For each event of this tuple must be referenced event variables ev_{left} and ev_{right} and a relational operator rop_{erc} must be set.

$$viol_{erc}((e_l, e_r)) = rop_{erc}(ev_{left}(e_l), ev_{right}(e_r)) \tag{23}$$

```
<EventRelationsConstraint relOpDenom="SET_EQUAL">
  <ConstraintProperties name="Sports 1A at Same Time" type="soft">
    <PenaltyFunction weight="10.0" coefA="1.0" coefB="2.0" exponent="1.0"/>
    <Description>The sports lessons of the class 1A should be at the same
       time</Description>
  </ConstraintProperties>
  <EventRelationOperand>
   <EventVariableRef name="timeSlots"/>
   <CollectionRef name="Sports Male 1A"/>
  <EventRelationOperand>
  <EventRelationOperand>
   <EventVariableRef name="timeSlots"/>
   <CollectionRef name="Sports Female 1A"/>
  <EventRelationOperand>
</EventRelationsConstraint>
```

Listing 16 Event Relation Constraint: Same Time

In the example above the left operand returns the assigned time slots of the male sports lessons, the right operand the time slots of the female sports lessons. The relational operator *SET_EQUAL* compares this two lists and returns 0 if there is no difference and a value $x > 0$ otherwise. Another possibility is to declare two events at the same time to link the resource lists of these events.

4.12 Operation Lists

In the sections 11 and 14 we used operation lists without defining this entity. An operation list *ol* is a sequence of operations $(op_1, ..., op_n)$.

$$\begin{gathered} ol(x) = op_n(...(op_2(op_1(x)) \\ domain(ol) = domain(op_1) \wedge codomain(ol) = codomain(op_n) \\ codomain(op_i) \subset domain(op_{i+1}) \quad \forall\, i \in [1; n-1] \end{gathered} \tag{24}$$

The operations are defined in a XML file. This file of definitions can be used by an editor to create a selection of available operations. The definition of operations must be expressed in a programming language as part of the code of the solver tool. At this point the designer of a timetabling system has to decide whether be would prefer to modify the solver program code for performance reasons or to interpret the specification and to preserve the flexibility. The same approach is used for defining relational operators.

5 EATTS XML File Format for Results

5.1 Requirements to a Result Format

In section 4 we have described the EATTS format to define different timetabling problems. The purpose of planning is to generate a time table of high quality, i.e. with low costs. After a couple of runs of the planning framework there is a list of solutions. Then

these results must be compared. In some cases this comparison could be based on the absolute costs of the solution, but in other cases it must be a bit more precise. So a list of constraint violations and their owner would be nice. Let ec be an event constraint and $viol(e, s) > 0$. So ec is violated on the view of the event e. So there is a constraint violation of ec with owner e. The owner of a resource constraint violation is a resource tuple. The owner of an event relation constraint violation is a tuple of events.

A result file should contain information about the plan it is linked to, certainly the assignments of the solution, the absolute costs and the constraint violations and its owners. But there is another problem. If a plan is accepted it will be used by many people. These users are interested in various views of this plan, but they are usually not interested in the problem definition. So the result file should include all data to generate all the required views on the plan.

5.2 The Structure of EATTS Result Format

The format is also XML based, because of the benefits shown in section 3.

```
<TTResult>
  <Plan/>
  <Events>...</Events>
  <TimeModel>...</TimeModel>
  <Resources>...</Resources>
  <ConstraintViolations>...</ConstraintViolations>
</TTResult>
```

Listing 17 Structure of EATTS Result Format

The `Plan` stores a reference to the problem definition file. A list of all events with name and index is enclosed in the `Events` tag. For visualization it is necessary to have some informations regarding the time model. The common style of timetable visualization is a table. This table has *cycleLength* columns and r rows. Every timeslot must be assigned to one cell or a set of cells in this table. So every time slot resource gets a day-index which defines the column, and vertical start and end index, which set the rows. `Resources` includes all resources sorted by resource types. Every `Resource` has a *name* and an *index* attribute and contains a list of the events it is assigned to.

The total costs of the solution is the value of the attribute *totalCost* of the `ConstraintViolations`. This tag includes a list of the constraint violations sorted by constraints in their order of appearance in the XML file. The `ConstraintViolation` tag has information about its cost and its owner.

6 Conversion of High School Timetabling XML into EATTS XML Format

6.1 Conversion of Time, Resources, and Events

In this section we want to show how a problem given in the existing format for High School Timetabling can be described in the EATTS format. `TimeGroups`, including `Week` and `Day`, can be converted into resource groups of the type *TimeSlot*. Resources of this type must be generated automatically so that there is a 1:1 mapping between `Time` elements *telem* and the resources of the type *TimeSlot timeSlot*. As example *Mon_1* will be mapped to a time resource with start time 8 *hours* and duration 1

lesson, whereby the start of the planning period is $Monday\ 0:00$ and the duration of one lesson must be set. The membership to time groups of *telem* must be adopted to the corresponding resource groups of type *TimeSlot*.

The concept of resource types, resources and resource groups is very similar in both formats so that a conversion is possible without a hitch. It should be considered that the names of resource types and resource groups in EATTS format must be unique. The name of the resources will be stored in attribute *name* of type *String*.

The conversion of courses and events is a little bit more difficult. In the EATTS concept a course consists of one event or a set of events with their linked time slot resources. In some cases the accumulated working time may differ from the time which should be credited to the work load account, i.e. a premium, a *Properties* resource type with an attribute *workCredit* and a property resource must be assigned fixed to every event. If a class is split into two or more courses, like for sports, for each of these courses a group of the participating students must be declared. If these courses should be at the same time an event relation constraint can be added or the *TimeSlot* resource lists can be linked to ensure that they are assigned to the same time slots.

6.2 Conversion of Constraints

Now will be shown how the constraints can be defined with the concept of EATTS format. There is a list of constraints in the paper [PAD+08, p. 10f]. *AssignTimSlotConstraint* and *AssignResourceConstraint* can be converted to a `MinMaxConstraint`. The *LinkedEventsConstraint* and the *AvoidSplitAssigmnetsConstraint* can be matched to an event relation constraint, as shown in section 4.11, or the according resource lists can be coupled.

NoResourceClashConstraint is a resource constraint, for every tuple of resources must the count of events, which are assigned to all resources of the tuple, less or equals 1. For the constraints *WorkloadAssignment*, *IdleTimesConstraint* and *TimeSlotAmountConstraint* an operation must be applied to the list of assigned time slots of a certain resource. This operation must compute the duration of these time slots or the count or must measure the idle times.

The selector of a *SubjectSequenceConstraint* selects the assigned events to a combination of a class or student and a collection of subjects. The resource variable selects the time slots of these events and applies an operation on the time slots list which measures the sequences. The $viol(class, subjectCollection)$ function can compare the result of the operation with the allowed sequence length. The *ClusterTimeSlotConstraint* is similar, the count of events assigned to a resource and a collection of TimeSlots should lie between minimum and maximum. The *CourseSpreadingConstraint* can also be matched to a resource constraint by selecting the event groups over the fixed assigned property resources. In that case there must be a 1:1 relation between property resource and event.

6.3 Conversion of Solutions

The conversion of the solutions needs a framework which computes the constraint violations. In the first step the problem must be read. Then the assignments must be set according to the solution. Now the framework can compute the total cost and the

constraint violations. In the last step the data can be written to the solutions file in EATTS format.

6.4 Conversion of EATTS XML format into High School Timetabling XML Format

This way of conversion would in most cases be senseless, at least if the statement of the problem is not concerned with High School Timetabling. But if there is a well defined problem definition for High School Timetabling like the result of the conversion above, an inversion of this process is possible. Let 2^{HST} be the set of all timetabling problems which can be defined with the High School Timetabling format and 2^{EATTS} the set of all timetabling problems which can be described with the EATTS format. Above we have shown that there is a conversion $C : 2^{HST} \rightarrow 2^{EATTS}$. Let $Codom(2^{HST})$ be the codomain of 2^{HST} in 2^{EATTS}, so $Codom(2^{HST}) \subsetneq 2^{EATTS}$. The inverse function C' is defined by the following equation.

$$C' : Codom(2^{HST}) \rightarrow 2^{HST}; C'(C(hst)) = hst \quad \forall\, hst \in 2^{HST} \tag{25}$$

7 EATTS XML File Format Tools

The EATTS framework provides a set of tools to handle XML files defined in the format described above. First of all there is a web based editor. It can be used to create new timetabling problems, the input mask will be automatically adapted to the user defined resource types with different attributes. There is also a tool, called plan viewer, to visualize timetables from various views like the view of an event or of a resource. Often the generated plans should be changed manually after planning. So an application would be nice which gives support on changing the assignments, whereby the costs and constraint violations must be updated just in time. The work on this will be finished soon. At least the core of an automatic timetabling framework must be a solver for generating as good as possible timetables. Our solver is written in JAVA and provides classes to read and write the EATTS XML format and a set of optimization algorithms which can be applied to all timetabling problems defined by the XML. So the framework can be used to compare different algorithms on many various timetabling problems.

8 Conclusion

Our approach is to define different types of timetabling problems with one format. The existing formats are not flexible enough, because they can only describe timetabling problems of one class. So we declared the EATTS XML format with regard to the possibility of describing as much as possible problems.

Some very special constraints, like counting idle times, are not easy compatible with the EATTS general approach of constraint definition. In these cases must be implemented very special operations in the solver. Often these operations can be used for other problems, too. The XML definition of some constraints could be very inconvenient and need many tags, especially in case of event relation constraints. But this should not be a big problem if the problem definition is automatically generated.

The main future work is to describe many different real world problems with our format. On this way we will adapt the XML format and define a set of operations. Then we want to categorize these problems and test the solver. We think a group of types will need a special set of operators, like neighborhood or genetic operators, to get good time tables. We want to have a special look on reusability and standardization. We like to get problem definitions and suggestions at any time.

Acknowledgements We would like to thank Eugen Kremer for his participation in the EATTS project and especially the XML file format.

References

[GWB03] Matthias Gröbner, Peter Wilke, and Stefan Büttcher. A standard framework for timetabling problems. In *Practice and Theory of AutomatedTimetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 24–38. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-40699-0. ISSN 0302-9743 (Print) 1611-3349 (Online). URL `http://www.springerlink.com/content/2mj0rwlpbp5uvh1v/`.

[PAD+08] Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngas, Cimmo Nurmi, David Ranson, and Henri Ruizenaar. An xml format for benchmarks in high school timetabling. In *Proceeding of the 7th international conference on the Practice and Theory of Automated Timetabling*. Patat2008, 2008. URL `http://w1.cirrelt.ca/~patat2008/PATAT_7_PROCEEDINGS/Papers/Post-WD2a.p% df`.

[W3C10a] W3C®, January 2010. `http://www.w3.org/XML/Schema`.

[W3C10b] W3C®, January 2010. `http://www.w3.org/XML/`.

[wik10] wikipedia, January 2010. `http://en.wikipedia.org/wiki/Document_Type_Definition`.

Abstracts

Assigning referees to a Chilean football tournament by integer programming and patterns

Fernando Alarcón · Guillermo Durán · Mario Guajardo

Abstract The referee assignment in real-world sport competitions can lead to a hard combinatorial decision problem when a number of conditions must be taken into account, such as fairness and operational considerations. In practice, the assignment is often carried out manually by a group of experts, based on poorly defined criteria. Recently, a number of sports scheduling articles have focused on improving this task, by developing different models and solution approaches.

In this talk, we study a referee assignment problem in the Chilean football context. The main parameters are a tournament and a set of referees. The tournament is a set of games scheduled in a given number of rounds. Every game is played by two teams in a venue known beforehand. Every game is refereed by a "main referee" (or just "referee"), usually supported by two "assistant referees". The problem consists of assigning main referees to the games, fulfilling a number of conditions which have been defined together with the managers in charge of the referee assignment in the Chilean Football Association. These conditions intend to add transparency, objectivity and fairness to the assignment decision. They include: balancing the total amount of games refereed by every referee, balancing the number of games a referee is assigned to a same team, balancing the travel distances of the referees and taking into account their experience to referee special matches (e.g., the Association prefers to assign the most experienced referees to the matches played by *classic* rivals). We propose an integer linear programming model to tackle this problem.

The natural formulation of this referee assignment model can be solved by using standard computer solvers. However, large instances may lead to relatively long solution

F. Alarcón
Department of Industrial Engineering, FCFM, University of Chile, Chile.
E-mail: falarcon@ing.uchile.cl

G. Durán
Department of Industrial Engineering, FCFM, University of Chile, Chile, and Department of Mathematics, FCEN, University of Buenos Aires, Argentina.
E-mail: gduran@dii.uchile.cl

M. Guajardo
Department of Finance and Management Science, Norwegian School of Economics and Business Administration, Norway.
E-mail: mario.guajardo@nhh.no

times, an undesirable matter for practical use. The current tournament of the Chilean First Division League consists of 21 teams that play 420 games distributed over 42 rounds. There are 16 referees, who can be assigned to these games. The IP model for this tournament contains about 6,700 variables and 10,000 constraints. Normally, the managers of the Association would like to analyze different instances of the problems in relatively short times. Furthermore, though at the beginning of the season they require us to find a solution to assign the referees to all the games of the tournament, they can also ask us to modify the assignment round by round, based on unexpected facts such as a temporary non-availability of a given referee. They expect us to solve every instance in a matter of minutes.

We have developed a solution approach based on patterns, inspired in the well-known home and away pattern procedures that have been successfully utilized in a number of articles aimed at scheduling sport games. While in the scheduling of games the patterns indicate if a team plays at home or away (or if it is bye) in each round of the tournament, the patterns we implement for the referee assignment indicate the set of games to which a referee can be assigned in each round. Given the particular geography of Chile, a very long and narrow country, we define these sets of games based on the location of the venues where they are going to be played. Any other arbitrary criteria to define the sets may also be suitable. Our solution methodology consists of two stages. In the first stage, we generate the patterns for each referee by solving an IP model that considers some of the constraints of the original problem. In the second stage, we implement another IP model that incorporates the rest of the conditions and assigns the referees to the games of the tournament. To the best of our knowledge, this work is the first one developing an approach based on patterns to solve a referee scheduling problem.

We implement the model for real instances of the problem and report results that improve the traditional assignment significantly. For instance, while the traditional assignment in the last tournament (generated manually) exhibits relatively large differences in the number of times a referee is assigned to games where one or another given team played, our assignment balances these amounts. Furthermore, by using the patterns-based approach we obtain a solution in a couple of minutes, while by running the natural formulation of the model it takes up to an hour (in both cases, we use an up-to-date computer and the solver CPLEX).

Currently, we are working together with the managers of the Chilean Football Association, evaluating the suitability of this referee assignment methodology for its real use in their 2010 professional and young divisions tournaments. By using our approach, the managers intend to stop receiving complaints from the teams on the referee assignment, a matter they have faced in multiple occasions by using their traditional methodology.

We would expect a concrete application of this work to contribute to the state of the art in sports scheduling and the related practices in football and other sports. In fact, though the scheduling of games has shown a significant development in the last decades, the literature on applications of sports scheduling techniques in real-world referee assignment problems is still scarce.

Keywords Sports scheduling · Referee assignment · Chilean football · Integer linear programming · Patterns

Tabu assisted guided local search approaches for freight service network design

Ruibin Bai · **Graham Kendall**

1 Introduction

Service network design involves determination of the most cost-effective transportation network and service characteristics subject to various constraints. Good progress has been made in developing metaheuristic approaches that can compete or even outperform some commercial software packages (Ghamlouche et al 2004; Pedersen et al 2009). However, since most of these metaheuristic methods involve solving many capacitated multi-commodity minimum cost flow problems, computational time tends to be a bottleneck. In this research, we intend to build on the success of a guided local search metaheuristic (Bai et al 2010) in reducing computational time, without compromising solution quality, and carry out a set of experiments and analyses in an attempt to discover elements and mechanisms that could improve the algorithmic performance further.

2 Freight Service Network Design

We focus on a specific service network design formulation that has been studied recently in (Pedersen et al 2009). For the purpose of completeness, we also present it here. Let $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ denote a directed graph with nodes $\mathscr{N}$ and arcs $\mathscr{A}$. Denote (i, j) be the arc from node i to node j. Let $\mathscr{K}$ be the set of commodities. For each commodity $k \in \mathscr{K}$, let $o(k)$ and $s(k)$ stand for its origin and destination respectively. Let y_{ij} be boolean design variables and y_{ij} equals 1 if arc (i, j) is used in the final design and 0 otherwise. Denote x_{ij}^k be flow of commodity k on arc (i, j). Let u_{ij} and f_{ij} be the capacity and fixed cost of arc (i, j). Denote c_{ij}^k be the variable cost of moving one unit of commodity k along arc (i, j). The service network design problem can be formulated as follows:

Ruibin Bai
Division of Computer Science, University of Nottingham Ningbo China
Tel.: +86-574-88180278
Fax: +86-574-88180125
E-mail: ruibin.bai@nottingham.edu.cn

Graham Kendall
School of Computer Science, University of Nottingham, Nottingham, UK
E-mail: gxk@cs.nott.ac.uk

$$\min \quad \sum_{(i,j)\in\mathscr{A}} f_{ij}y_{ij} + \sum_{k\in\mathscr{K}} \sum_{(i,j)\in\mathscr{A}} c_{ij}^k x_{ij}^k \tag{1}$$

subject to

$$\sum_{k\in\mathscr{K}} x_{ij}^k \leq u_{ij}y_{ij} \quad \forall (i,j) \in \mathscr{A} \tag{2}$$

$$\sum_{j\in\mathscr{N}^+(i)} x_{ij}^k - \sum_{j\in\mathscr{N}^-(i)} x_{ji}^k = b_i^k, \quad \forall i \in \mathscr{N}, \forall k \in \mathscr{K} \tag{3}$$

$$\sum_{j\in\mathscr{N}^-(i)} y_{ji} - \sum_{j\in\mathscr{N}^+(i)} y_{ij} = 0 \quad \forall i \in \mathscr{N} \tag{4}$$

where $x_{ij}^k \geq 0$ and $y_{ij} \in \{0,1\}$ are decision variables. $\mathscr{N}^+(i)$ (respectively $\mathscr{N}^-(i)$) stands for the set of outward (respectively inward) neighbours of node i. We set $b_i^k = d^k$ if $i = o(k)$, and $b_i^k = -d^k$ if $i = s(k)$, and 0 otherwise. Note that for a given set of design variables $\bar{y}_{ij}$, the problem becomes a capacitated multicommodity minimum cost flow problem (CMMCF).

In (Bai et al 2010), we have shown that a variant of the guided local search (GLS) approach is able to produce competitive results with much less computational time than a recently proposed tabu search method (Pedersen et al 2009). Based on this initial success, this research aims to investigate, in detail, components and mechanisms that may lead to further improvement either in terms of computational time or solution quality. In particular, we intend to investigate; a) how effectively the current GLS escapes from a local optimum. b) whether more efficient mechanisms can be found and integrated within GLS.

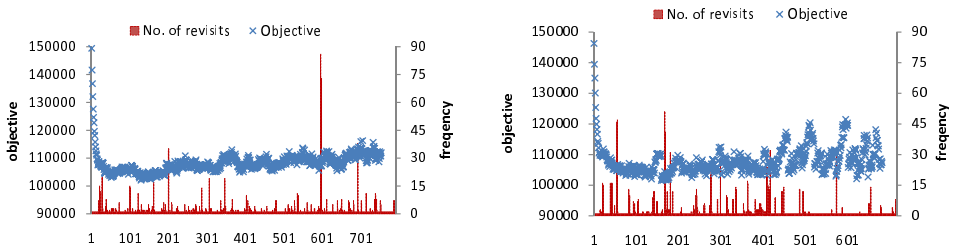
3 Guided Local Search

Guided local search (GLS) is a metaheuristic for constraint satisfaction and combinatorial optimization problems (Voudouris and Tsang 2003). Its main idea is augmenting the original objective function so that the search not only escapes from local optima but also obtains high quality solutions. In our implementation, the neighbourhood is defined by either closing or opening an arc. The flow variables (x_{ij}) are then determined by solving the corresponding CMMCF problem using a free LP solver, `LP_Solve`. More implementation details of this approach can be found in (Bai et al 2010).

3.1 Local optima trap

In order to analyse how efficiently GLS escapes from local optima, we carried out experiments based on some of the 24 C-set benchmark problems used in (Pedersen et al 2009). All local optima that have been identified during the search are recorded together with its visit frequency (i.e. the number of times a local optimum is visited). All algorithms are tested on a same machine with the same amount of computational time. Initially we tested a simple GLS approach and the multi-start GLS (denoted by M-GLS) proposed in (Bai et al 2010). Figure 1 (a) and (b) show the corresponding information for instance C37 (similar results were obtained for other instances). The horizontal axis represents the list of local optima found during the search. One can see that both approaches revisited the same local optima many times. For the simple GLS, a few local optima are revisited over 80 times. On average, it visits each local optimum 2.3 times. For M-GLS, to our surprise, the average number

of visits per local optimum is even higher (2.9). Nevertheless, high quality local optima by M-GLS tend to attract higher visit frequencies which may be one of the reasons that lead to better performance than the simple GLS. Overall, both versions of GLS wasted significant time when the same solution is evaluated many times. It is also interesting to note that GLS seems to converge to a good local optimum very quickly but is not so efficient when escaping from some local optima.



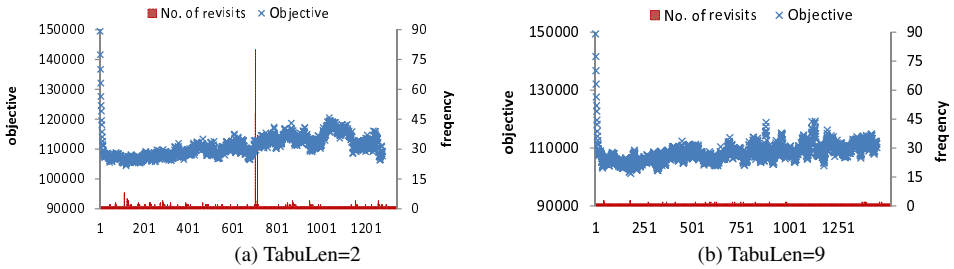
(a) Simple GLS (b) Multi-start GLS

Fig. 1 The number of visits to each local optima by the simple GLS and M-GLS (C37)

3.2 Tabu assisted GLS (T-GLS)

Since M-GLS cannot effectively prevent local optima revisits, we borrow the idea of the tabu search metaheuristic and introduced a tabu list into the simple GLS. The tabu list contains a list of arcs that have been modified recently in the current solution. The length of the tabu list is fixed to a predefined parameter *TabuLen*. The list is then maintained on a first-in-first-out basis. Figure 2 (a) and (b) plot the objective values and the revisit frequency by GLS with $TabuLen = 2$ and $TabuLen = 9$ respectively. It can be seen that even a tabu list of length 2 is effective in reducing GLS visiting local optima many times. When we increase *TabuLen* to 9, the majority of local optima are visited only once.

In order to measure the performance of these variants of GLS (namely simple GLS, M-GLS and T-GLS), we have tested them on 24 widely used benchmark problem instances. Details of the results are not included here but we will present them during the conference. The general observation is that T-GLS obtains better results than the simple GLS for all instances but is outperformed by M-GLS for the majority of instances. It seems, based on our observations, that T-GLS lacks necessary random elements to diversify the search.



(a) TabuLen=2 (b) TabuLen=9

Fig. 2 Revisit frequency of local optima by tabu assisted GLS (C37)

4 Discussions and future work

In this research, we carried out experiments to monitor the revisit frequency of local optima by a GLS metaheuristic. Results show that in both the simple GLS and its multi-start version, time is wasted due to revisiting. A simple tabu assisted GLS schema is implemented to prevent this problem. Although improvements have been obtained when compared against the simple GLS, this simple hybridisation fails to produce better results than the multi-start GLS. One of the possibilities that causes this problem may be that the current tabu assisted GLS does not contain random elements and it lacks efficient mechanisms to "jump" to a distant point in the search space. We are currently trying to implement various schemata to combine the multi-start GLS with a tabu list. Early experiments on a limited number of instances have shown very promising results. Comprehensive tests will be carried out and results will be reported during the conference.

In addition, our observations show that `LP_Solve` struggles on some problem instances, for which the majority of computational time is used when solving CMMCF problems (more than 85%). For some instances, LP_Solve even fails to solve one single CMMCF instance within 300 seconds. In future, we will look at other more efficient LP solvers, including `NAG` and `CPLEX`. It is hoped that a faster LP solver can improve the proposed algorithm further.

References

Bai R, Kendall G, Li J (2010) A guided local search approach for service network design problem with asset balancing. In: 2010 International Conference on Logistics Systems and Intelligent Management (ICLSIM 2010), January 9-10, Harbin, China, pp 110–115

Ghamlouche I, Crainic TG, Gendreau M (2004) Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. Annals of Operations Research 131:109–133

Pedersen MB, Crainic TG, Madsen OB (2009) Models and tabu search metaheuristics for service network design with asset-balance requirements. Transportation Science 43(4):432–454

Voudouris C, Tsang EP (2003) Guided local search. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics, Kluwer, pp 185–218

The Relaxed Traveling Tournament Problem
Extended Abstract

Renjun Bao · Michael A. Trick

Abstract The Traveling Tournament Problem (TTP) is a sports scheduling problem that encapsulates two major aspects of some sports leagues: restrictions on acceptable home/away patterns and limits on travel distances. One major assumption in the TTP is that the schedule is compact: every team plays in every time slot. Some sports leagues have both pattern restrictions and distance limits but are not compact. In such schedules, one or more teams can have a bye in any time slot. We examine a generalization of the TTP where byes are possible.

Keywords Sports Scheduling · Integer Programming · Constraint Programming

Over the last twenty years, there has been increased interest in computational methods for creating sports schedules. This interest has been driven both by advances in the combinatorial structure of sports schedules and in the practical need for schedules by real sports leagues. There have been a number of recent surveys on the subject [5, 11,3] along with a recent annotated bibliography ([10]).

One path of research has revolved around the Traveling Tournament Problem (TTP). In the TTP, there are $2n$ teams, each with a home venue. The teams wish to play a double round robin tournament, whereby each team will play every other team twice, once at each team's home venue. This means that every team needs to play $2n-2$ games. There are $2n-2$ time slots in which to play these games, so every team plays in every time slot. Associated with a TTP instance is a distance matrix D where D_{ij} is the distance between the venue of team i and team j. Teams are assumed to begin and end the tournament at their home venue. If team i plays consecutive games at the venues of j and k, then i travels from its home venue to that of j then on to k before returning home to i's venue (and similarly for longer trips). The objective is to minimize the total travel of the teams subject to some requirements on the number

R. Bao
Cleveland State University
Cleveland, OH USA E-mail: baorenjun@gmail.com

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

of consecutive home (or road) games by each team. Those requirements can vary, but the canonical TTP requires that each team play no more than three consecutive home games or three consecutive road games.

The TTP was developed to abstract out the key issues in scheduling Major League Baseball, the United States professional baseball league. For that league, there are dozens of restrictions and requirements but the key issue was the tradeoff between distance traveled and home/away requirements. Since its introduction, the TTP has been the subject of numerous papers (see, for instance, [6, 2, 12, 9, 4, 14]) and is supported by an active website. Despite this interest, the TTP has proven to be a computational difficult challenge. For many years, the six-team instance NL6 was the largest instance solved to provable optimality. In 2008, NL8 was solved; NL10 was solved in late 2009 along with other ten team instances. This leaves twelve teams as the smallest unsolved instances, which still seems a remarkably small league size for such a simple problem description.

The goal of the TTP is to find a compact schedule: the number of time slots is equal to the number of games each team plays. This forces every team to play in every time slot. There are a number of leagues which are concerned with both home/away patterns and distance traveled but do not require compact schedules. Two significant examples in the United States are the National Basketball Association and the National Hockey League. Both leagues are economically significant, with yearly revenues of US$3.6 billion and US$2.8 billion respectively. If we examine the schedule for a team in each league, as shown in Figure 1, we can see a number of scheduling similarities.

The timetable at the top of Figure 1 is the schedule for the NBA's Cleveland Cavaliers for December 2009. In that schedule, home games are represented by darker background dates; away games have a white background. The timetable at the bottom of Figure 1 is for the NHL's Pittsburgh Penguins, with away games marked with an "@" symbol. For both leagues, the dates on which games are played vary by team. In fact, there are both NBA and NHL games every day of the months given. Over the course of a season, there is approximately one off day for every game played by a team (season lengths are 82 games per team for each league over approximately 160 days).

For both of these leagues, it is generally the case that teams with consecutive road games travel between the road cities, rather than returning home in between. This makes travel an important component of the schedule. For instance, the Penguins schedule begins the month with road games at Anaheim, Los Angeles and San Jose (all teams on the US west coast) before returning to the east coast to play at Boston and then returning home. This is a much better trip than Anaheim, Boston, Los Angeles and San Jose for a team based in eastern part of the United States, as Pittsburgh is.

These schedules lead to a natural generalization of the Traveling Tournament Problem which we call the Relaxed Traveling Tournament Problem (RTTP): instead of fixing the schedule length to be $2n - 2$, let the schedule length be $2n - 2 + K$ for some integer $K \geq 0$. For a given K, we will denote the corresponding problem as the K-RTTP. For $K = 0$, the RTTP is just the TTP. For $K > 0$, each team has K slots in which it does not play. We call such a slot a bye for the team. There are many ways in which these byes could be counted. Initially, we will simply ignore byes when determining consecutive home or away games. So a home(H)/away(A)/bye(B) pattern of `HHBHAABBA` would be treated as having one three game home stand followed by a three game road trip. The advantage of this definition is that solutions for the TTP

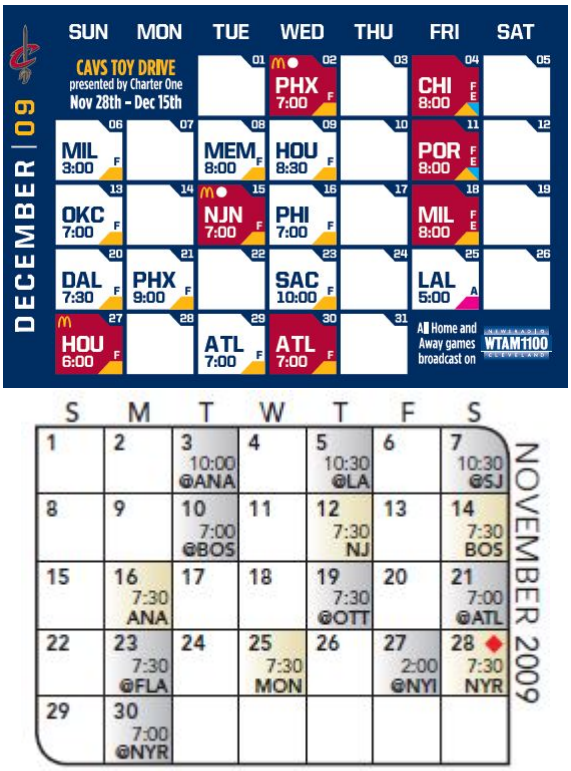


Fig. 1 NBA and NHL Schedules

are feasible for the K-RTTP for all $K \geq 0$ (in fact, k_1-RTTP are feasible for k_2-RTTP for $k_1 \leq k_2$).

For relatively small K this treatment of byes is reasonable. But for large K, simply ignoring the byes can lead to undesirable behavior whereby, for instances, a sequence like ABBBBBBBBBBA is treated like a two game road trip, when any real team would return home in the interim. For larger K (like $K = 2n - 2$, mimicking the NBA and NHL), we can put lower and upper bounds on the number of consecutive byes, or have other restrictions to have the patterns reflect playable schedules.

With this definition of K-RTTP, there are a number of interesting questions. Key to some of these is the idea of the Independent Lower Bound (ILB). For the TTP, the ILB is found by determining, for each team, the minimum distance that team must travel to visit all other teams, respecting limits on trip length. The ILB is then the sum of that value over all teams. Clearly the ILB is a lower bound for the TTP and for the K-TTP for all K. It is a reasonable conjecture that the ILB is tight for TTPs of at least a certain size. The work of Urrutia and Ribeiro [13] show this is not the case, even if there are no upper bounds on trip length and the distance between i and j is 1 for any $i \neq j$. Do byes help in this case? We conjecture that for sufficiently large K, the ILB is tight for the K-TTP, where K depends on n, but not on D. Even stronger, it may be that the ILB is tight for the 1-RTTP. While this may seem farfetched (can one bye per

team be enough?), the work of [7] shows that for avoiding "breaks" (consecutive homes or aways), one bye per team is sufficient to reduce the number of breaks in a round robin schedule on $2n$ teams from $2n-2$ to zero. Perhaps one break is also enough to allow every team minimum travel. Our computational methods confirm this for NL4, though that is extremely slender evidence.

Key to exploration of this and other issues is the need for a computational approach for solving K-RTTPs. Over the last decade, there have been a number of approaches proposed to exactly solve the TTP (not counting many more heuristic approaches which are beyond the scope of this work). In [1], a number of alternative approaches were proposed, including generalizations of the well-known three phase approach (finding pattern sets, schedules, and game assignments) and trip formulations for the TTP. We have implemented integer and constraint programming versions to determine optimal schedules and are in the process of developing a system based on logic-based Benders decomposition [8]. With our current implementations, we can state the following:

1. The RTTP appears to be even harder than the TTP to solve to optimality
2. Even small K leads to interesting, difficult instances
3. Current techniques for the TTP can be generalized to the RTTP, though the generalizations are not trivial or straightforward.

In the full version, we will outline the generalizations and describe how they work computationally. We will also address issues of the structure of small K problems, and the relationship to the ILB.

References

1. Bao, R.: Time Relaxed Round Robin Tournament and the The National Basketball Association Scheduling Problem. dissertation, Cleveland State University (2009)
2. Benoist, T., Laburthe, F., Rottembourg, B.: Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems. In: Proceedings CPAIOR'01, Wye College (Imperial College), Ashford, Kent UK (2001)
3. Briskorn, D.: Sports Leagues Scheduling: Models, Combinatorial Properties, and Optimization Algorithms. Springer (2008)
4. Cheung, K.: A benders approach for computing lower bounds for the mirrored traveling tournament problem. Discrete Optimization **6**, 189–196 (2009)
5. Drexl, A., Knust, S.: Sports league scheduling: Graph - and resource - based models. Omega (to appear) (2010)
6. Easton, K., Nemhauser, G., Trick, M.: The traveling tournament problem: Description and benchmarks. In: T. Walsh (ed.) Principles and Practice of Constraint Programming - CP 2001, *Lecture Notes in Computer Science*, vol. 2239, pp. 580–585. Springer Berlin / Heidelberg (2001)
7. Froncek, D., Meszka, M.: Round robin tournaments with one bye and no breaks in home-away patterns are unique. In: S.P. G. Kendall E. Burke, M. Gendreau (eds.) MISTA 2003, pp. 331–340 (2005)
8. Hooker, J., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96**, 33–60 (2003)
9. Irnich, S.: A new branch-and-price algorithm for the traveling tournament problem. Technical Report OR-01-2009, Chair for Operations Research and Supply Chain Management, RWTH Aachen University, Aachen (2009)
10. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography. Computers & Operations Research **37**(1), 1 – 19 (2010)
11. Rasmussen, R., Trick, M.: A Benders approach for constrained minimum break problem. European Journal of Operational Research **177**(1), 198–213 (2007)

12. Urrutia, S., Ribeiro, C., Melo, R.: A new lower bound to the traveling tournament problem. In: Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling (2007)
13. Urrutia, S., Ribeiro, C.C.: Maximizing breaks and bounding solutions to the mirrored traveling tournament problem. Discrete Appl. Math. **154**(13), 1932–1938 (2006)
14. Uthus, D.C., Riddle, P.J., Guesgen, H.W.: Dfs* and the traveling tournament problem. In: CPAIOR '09: Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 279–293 (2009)

Modelling Issues in Nurse Rostering

Burak Bilgin [1] · **Patrick De Causmaecker** [2] · **Greet Vanden Berghe** [1,3]

Abstract The real world nurse rostering problem requires numerous extensions to the nurse rostering models reported in the literature. In this work, we present some issues in nurse rostering model and illustrate them with specific examples. The extensions to our previous model [1] are inspired by the cases collected during collaboration with an industrial partner (GPS Time Security NV).

De Causmaecker and Vanden Berghe have categorised the nurse rostering problems regarding to the personnel environment, work characteristics, and optimisation objectives [2,3]. The skill type is an element of the personnel environment in this categorisation. Although there are several categories regarding the treatment of skill types, we report the most complicated one in this abstract. In some wards, employees can have multiple skill types and different levels of experience for each skill type. Consider a senior regular nurse who is a trainee as a head nurse. She has two skill types: regular and head nurse, but different levels of experience for each of them: senior and trainee. In this type of problems, the minimum level of experience is also given in the coverage constraints.

Domain. Domain is a new modelling element used in the definition of constraints. The composition of a domain object is as follows:

- Date set
  - Handling of the date set: individual or complete
- Shift type set

This research project is funded by IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders) within the project IWT 080356 - GPS Plan.

[1] Combinatorial Optimisation and Decision Support (CODeS)
Department of Information Technology, KaHo Sint Lieven
{Burak.Bilgin, Greet.Vandenberghe}@kahosl.be

[2] Combinatorial Optimisation and Decision Support (CODeS)
Department of Computer Science, KUL@K, K.U.Leuven
Patrick.DeCausmaecker@kuleuven-kortrijk.be

[3] Katholieke Universiteit Leuven,
Department of Computer Science and Information Technology

– Skill type set

Absence request as a block. In the following examples, different domain objects are used as parameters in the absence and assignment requests of the nurses. Consider a nurse who plans a ski vacation over a period of five days with a fixed starting day. This vacation request will be considered granted, if and only if all the days in the date set are granted. Hence, the date set is handled as a single block.

Absence request on individual days. Another example is a nurse who wants to help redecorating her house. This vacation request can be granted partially. The more days are granted, the better it is for the nurse. Hence, the dates in the set are handled individually.

Absence request on a specific shift types set. A common example encountered in Belgian hospitals is a nurse who wants to take care of her children on Wednesday afternoons, because Wednesday afternoons are school free in Belgium. In this case, the date set consists of the Wednesdays in the planning period, and the shift type set consists of the afternoon, evening, and probably also the night shifts.

Assignment request for a specific skill types set. There are also cases in which a nurse is a senior caregiver but a trainee as a regular nurse. She wants to work as a regular nurse as much as possible to gain experience in this skill type. In this case, the *skill type set* of the domain element consists of "regular nurse".

Individual bank holidays. The utilisation of domains is not limited to the employee requests. Some institutions allow their nurses to define their own bank holidays. There are two advantages of this practice. First, the shortage of the available nurses during the public holidays is reduced, because not all nurses take leave at the same time. Second, nurses can select their own holidays based on their religion and nationality. Instead of defining a global *holidays worked counter* that applies to the whole ward, counters with domains that represent the holidays of the individuals are defined to address this practice.

Constraint period exceeding the planning period. The *holidays worked counter* poses another challenge. The period of this constraint, a year, exceeds the planning period. A typical planning period of four weeks is shorter than a year. The holidays before and after the current planning period need to be taken into account as well. The threshold values of the constraint need to be adjusted using the following formula. Let h be the number of assignments on bank holidays before the current planning period. Let r be the number of remaining holidays after the current planning period.

$$min' = min - h - r \tag{1}$$

$$max' = max - h \tag{2}$$

In academic models, the planning period is usually considered to be isolated. The real world practice, however, requires the consideration of assignments in the previous planning periods, as well as the structure of the upcoming planning periods. The *holidays worked counter* is an example of this requirement. The continuity between the

planning periods is studied in greater detail by Glass and Knight [4]. They carried out their studies on the Nottingham benchmarks, a collection of nurse rostering problem instances from different countries [5].

Collaboration. The collaboration constraint restricts the composition of the nurses that work together. The parameters of the collaboration constraint are as follows:

- Employee set
- Domain
- Threshold
- Weight

The employee set needs to consist of at least two employees that have to work together or not. The collaboration can be defined regarding any domain element. The threshold value determines the nature of the collaboration. If the maximum threshold value is set to zero, this means that a collaboration among the nurses in the employee set is not desired. A positive minimum threshold is needed in order to express a requirement of nurses working together.

Training. The objective of the training constraint is to increase the level of experience of the trainees. Usually, seniors of a skill type are engaged to supervise the work of trainees of the same skill type. This supervision is considered as training. The training constraint can occur in different ways in real world practice. Here is a common example: a senior can train up to five trainees at the same time. There must be sufficient numbers of seniors assigned for the trainees that are assigned to the same domain. The parameters and the formula of the training constraint are as follows:

- Preceding level of experience (t)
- Succeeding level of experience (s)
- Ratio (r)
- Domain
- Weight

$$\lceil r \cdot t \rceil \leq s \tag{3}$$

In this work, we report extensions to the nurse rostering model in order to address the real world problems. The publication of an XSD of the extended model, real world data files with the extended features and experimentation on the data files constitute the future work. The nurse rostering problem is dynamic in its nature and all the time provides the industry professionals with new challenges. The models need to be in continuous redevelopment to address the upcoming challenges.

References

1. Bilgin, B., De Causmaecker, P., Rossie, B., Vanden Berghe, G.: Local search neighbourhoods for dealing with a novel nurse rostering model. Tech. rep., KaHo Sint-Lieven, Information Technology; K.U. Leuven Campus Kortrijk, Computer Science and Information Technology (2009). Submitted to Annals of Operations Research - Patat Special Issue.
2. De Causmaecker, P., Vanden Berghe, G.: Categorisation of personnel rostering problems. Working Paper K.U. Leuven (2009)

3. De Causmaecker, P., Vanden Berghe, G.: Towards a reference model for timetabling and rostering. Accepted for publication in Annals of Operations Research (2010)
4. Glass, C., Knight, R.: The nurse rostering problem: A critical appraisal of the problem structure. European Journal of Operational Research In Press (2009)
5. Burke, E.K., Curtois, T., Qu, R., Vanden Berghe, G.: A scatter search approach to the nurse rostering problem. Journal of the Operational Research Society Accepted for Publication (2009)

Semidefinite Programming Relaxations in Timetabling (Abstract)

Edmund K. Burke · Jakub Mareček · Andrew J. Parkes

Keywords timetabling · bounded colouring · vertex colouring · graph colouring · semidefinite programming

Semidefinite programming has recently gained much attention as a powerful method for deriving both strong lower bounds and approximation algorithms in combinatorial optimisation. There have not been, however, any applications to timetabling. We show one reason to believe that this could well change, ultimately.

Definitions In linear programming (LP), the task is to optimise a linear combination $c^T x$ subject to linear constraints $Ax = b$, together with the constraint that each in vector x of n variables is non-negative. The non-negativity of x, $x \in (R^+)^n$, can be seen be seen as a restriction of the variables to lie in the convex cone of the positive orthant. Using interior point methods, linear programming can be solved to any fixed precision in polynomial time. These methods also work for other symmetric convex cones.

Semidefinite programming (SDP, Bellman & Fan, 1963; Alizadeh, 1995; Wolkowicz, Saigal, & Vandenberghe, 2000) is a generalisation of linear programming, replacing the vector variable with a square symmetric matrix variable and the polyhedral symmetric convex cone of the positive orthant with the non-polyhedral symmetric convex cone of positive semidefinite matrices. Note that an $n \times n$ matrix, M, is positive semidefinite if and only if $y^T My \geq 0$ for all $y \in \mathbb{R}^n$. As all scalar multiplies of positive semidefinite matrices and convex combinations of pairs of positive semidefinite matrices are positive semidefinite, positive semidefinite matrices do form a convex cone in R^{n^2}. We denote $A \succeq B$ whenever $A - B$ is positive semidefinite, and use $\langle A, B \rangle$ for the inner product of matrices, which is $\sum_{i,j} A_{i,j} B_{j,i}$. Formally, semidefinite programming is the minimisation of $\langle C, X \rangle$ such that $\langle A_i, X \rangle = b_i \quad \forall i = 1 \ldots m$ and $X \succeq 0$, where X is a (primal) square symmetric matrix variable, C and A_i are compatible symmetric matrices, m is the number of constraints, and $b \in \mathbb{R}^m$.

Let us now consider a simple model of timetabling, underlying integer programming decompositions (Burke, Mareček, Parkes, & Rudová, 2010), for instance. The input consists of identifiers of events V, distinct enrolments U ("curricula"), rooms R, and periods P, plus mapping $F : \mathrm{U} \to 2^{\mathrm{V}} \setminus \emptyset$ from "curricula" to non-empty sets of events. Conflict graph $G = (\mathrm{V}, E)$ is given

Automated Scheduling, Optimisation and Planning Group, School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK. E-mail: {ekb,jxm,ajp}@cs.nott.ac.uk. For supplementary material, please see: `http://cs.nott.ac.uk/~jxm/`

by F, where events $F(u)$ is a clique in G for all $u \in \mathrm{U}$. The "core" decision variables are

$$Z_{p,v} = \begin{cases} 1 & \text{event } v \text{ is taught at period } p \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

which are subject to linear constraints

$$\forall v \in \mathrm{V} \qquad \sum_{p \in \mathrm{P}} Z_{p,v} = 1 \tag{2}$$

$$\forall p \in \mathrm{P}\ \forall u \in \mathrm{U} \qquad \sum_{v \in Fu} Z_{p,v} \leq 1 \tag{3}$$

$$\forall p \in \mathrm{P} \qquad \sum_{v \in \mathrm{V}} Z_{p,v} \leq |\mathrm{R}| \tag{4}$$

Notice that there is only a single mention (4) of rooms, which makes the colouring of the conflict graph $|\mathrm{R}|$-bounded. This means the cardinality of each colour class or the number of uses of each colour is at most $|\mathrm{R}|$. Depending on the tightness of the $|\mathrm{R}|$-bound, the chromatic number alone is not necessarily a good lower bound.

Related Work There are a number of ways to bound the chromatic number of a graph using SDP. Informally, the point is that a parameter of the graph, denoted theta, is at least as large as the clique number and no more than the chromatic number, yet is computable in polynomial time using SDP. The known theta-like bounds for unbounded colouring form a hierarchy (Szegedy, 1994):

$$\alpha(G) \leq \vartheta_{1/2}(G) \leq \vartheta(G) \leq \vartheta_2(G) \leq \chi(\overline{G}), \text{ or } \omega(G) \leq \vartheta_{1/2}(\overline{G}) \leq \vartheta(\overline{G}) \leq \vartheta_2(\overline{G}) \leq \chi(G),$$

where α is the size of the largest independent set, ω is the size of the largest clique, χ is the chromatic number, $\vartheta_{1/2}$ is the vector chromatic number (Karger, Motwani, & Sudan, 1998), ϑ is the strict vector chromatic number (Karger et al., 1998), ϑ_2 is the strong vector chromatic number (Kleinberg & Goemans, 1998), and bar indicates complementation. For the corresponding vector programming and semidefinite programming formulations, please consult the literature (Szegedy, 1994). In theory, all could be extended to bounded graph colouring, but none has been so far, up to the best of our knowledge.

In terms of applications, the celebrated SDP relaxation of the maximum cut problem (MAX-CUT, Goemans & Williamson, 1995) has been adapted to scheduling workload on two machines (Skutella, 2001; Yang, Ye, & Zhang, 2003) and home-away patterns in sports scheduling (Suzuka, Miyashiro, Yoshise, & Matsui, 2007). The techniques of "vector lifting" and "matrix lifting" have been applied in signal decoding in multi-antenna systems (Mobasher & Khandani, 2007; Mobasher, Taherzadeh, Sotirov, & Khandani, 2007). All of the above can be though of as rank-minimisation matrix completion problems (Fazel, Hindi, & Boyd, 2004), whose applications range from signal processing to statistics and system theory. We are now aware, however, of any applications to timetabling.

Bounding the Bounded Chromatic Number by SDP A clear application of semidefinite programming is in the detection of infeasibility in timetabling (2–4). The infeasibility test is given by lower bounding the $|\mathrm{R}|$-bounded chromatic number of the conflict graph and comparing it against $|\mathrm{P}|$, the number of periods available. Here we follow the method and notation of (Dukanovic & Rendl, 2007), briefly reported also in PATAT 2004 (Dukanovic & Rendl, 2004). The underlying matrix variable M is:

$$M_{u,v} = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are of the same colour} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

If we define $Y = tM$, we obtain legal colouring for integral t and $Y \in \{0, t\}$. In computing theta, these integrality constraints are dropped, resulting in an instance of SDP.

Table 1: An illustration of the effects of bounding the $|\mathrm{R}|$-bounded chromatic number of the instance sta-f-83: Column $\chi^{|\mathrm{R}|}$ lists the $|\mathrm{R}|$-bounded chromatic number obtained using integer linear programming, within time listed under "$\chi^{|\mathrm{R}|}$ Runtime" in seconds. Column $\vartheta^{|\mathrm{R}|}$ lists the bounds obtained using semidefinite programming and rounding up, within time listed under "$\vartheta^{|\mathrm{R}|}$ Runtime" in seconds. Column $|V|/|\mathrm{R}|$ lists the lower bound on the colours obtained by simple counting arguments and rounding up. Dash denotes the the omission of the $|\mathrm{R}|$-bounding constraint, giving the standard theta function instead of $\vartheta^{|\mathrm{R}|}$.

$\lvert\mathrm{R}\rvert$	$\chi^{\lvert\mathrm{R}\rvert}$	$\chi^{\lvert\mathrm{R}\rvert}$ Runtime	$\vartheta^{\lvert\mathrm{R}\rvert}$	$\vartheta^{\lvert\mathrm{R}\rvert}$ Runtime	$\lvert V\rvert/\lvert\mathrm{R}\rvert$
1	47	0.09	47	3.46	47
2	26	2.88	26	2.92	24
3	20	2.67	20	3.34	16
4	16	7.22	16	3.70	12
5	14	11.10	14	3.24	10
6	13	2.67	13	3.12	8
7	12	8.77	12	3.26	7
8	11	2.89	11	3.40	6
9	11	3.39	11	3.14	6
47	11	0.35	11	3.92	1
—	11	0.34	11	3.45	—

The theta relaxation can be modified to provide a bound on the bounded chromatic number by the addition of linear inequalities. In bounded colouring, we expect $\sum_u M_{uv} \leq |\mathrm{R}| \quad \forall v \in V$. This gives us the following SDP:

$$\vartheta^{|\mathrm{R}|}(\overline{G}) = \min t \tag{6}$$

$$\text{s.t.}: \quad \forall v \in V \qquad Y_{vv} = t \tag{7}$$

$$\forall \{u,v\} \in E \qquad Y_{uv} = 0 \tag{8}$$

$$\forall v \in V \qquad \sum_u Y_{uv} \leq t|\mathrm{R}| \tag{9}$$

$$Y - J \succeq 0 \tag{10}$$

where J is the all-ones matrix. A closely related bound can be derived using the matrix lifting operator $M_+(K)$ of Lovász and Schrijver (1991).

Numerical Experiments As a concrete example, we consider a small conflict graph from a standard benchmark problem. Specifically, we take the instance "sta-f-83" from the Toronto examination timetabling benchmarks [1]. There are 139 events, but the conflict graph has three connected components of 30, 47 and 62 vertices. Here, we use the 47-vertex component to study semidefinite programs produced by YALMIP (Löfberg, 2004) and solved using SeDuMi 1.21 (Sturm, 1999) and MathWorks Matlab R2009a on an Intel Core Duo P8600 at 2.4 GHz with 2 GB of RAM. For comparison, the bounded chromatic numbers are also provided. These were obtained using the most straightforward integer linear programming formulation solved using the defaults of ILOG CPLEX 12.10 on the same machine. Results are given in Table 1. Firstly, note that $|\mathrm{R}| = 1$ gives precisely the number of nodes, as would be expected. Secondly, note that $\vartheta^{|\mathrm{R}|}$ is generally much tighter than the lower bound $|V|/|\mathrm{R}|$ obtained by simple counting arguments. Accidentally, $\vartheta^{|\mathrm{R}|}$ lower bounds actually happen to match the optima in this particular instance. For example, at $|\mathrm{R}| = 5$, counting cannot rule out a 10-colouring, but the SDP bound shows that at least 14 colours are required. A 14-colouring together with a certificate of its optimality can be obtained using CPLEX, but not in polynomial time. As far as we know, SDP relaxations are the only way to get such information in polynomial time.

[1] See `ftp://ftp.mie.utoronto.ca/pub/carter/testprob/` and `http://www.cs.nott.ac.uk/~rxq/data.htm`

We have also tried $|\mathrm{R}|$-bounded modifications of the extensions to theta as given in (Dukanovic & Rendl, 2007): $\vartheta^{|\mathrm{R}|+}$ by keeping the $Y \geq 0$ constraints, and $\vartheta^{|\mathrm{R}|+\triangle}$ keeping the $Y \geq 0$ constraints and also adding triangle inequality constraints. These, however, slow down the solver and do not improve the bound on the tested instances. Also, theta can also be formulated on the complement graph, and this might be useful when the edge density is high, but we have not yet explored $|\mathrm{R}|$-bounded versions.

Future Work In the SDP relaxation of bounded graph colouring above, the colour assignment was not represented directly, but only in terms of the "same-time" classes of equivalence of nodes assigned the same colour. This makes it naturally invariant under permutation of the colours. This is sufficient for bounded colouring, but many objectives in timetabling refer to time-based patterns of activities, e.g. whether events should be on the same day or not. These are not invariant under "colour permutations" and so the "same-time" representation is no longer sufficient. For example, in lower bounding the **Surface** component of integer programming decompositions (Burke et al., 2010), i.e. the assignment of events to periods, including all the respective terms of the objective function, we presumably need to re-introduce some matrix variable mapping events to timeslots as in **Surface** (1). The matrix variable will need to be constrained so that there is only a single event in each roomslot. This gives a constraint on the rank of the matrix variable, and this can then be expressed in SDP. This can also be though of as an application of matrix-lifting operator $M_+(K)$ of Lovász and Schrijver (Lovász & Schrijver, 1991). Work in this direction is in progress.

Conclusions The aim of this abstract was not to present a practical method for bounding the optima in timetabling problems, yet. Indeed, SDP solvers are less well-developed than LP solvers, in general. Current interior point methods for semidefinite programming are rather slow, albeit running in time polynomial in the dimensions of the instance for any fixed precision. Our hope, however, is that SDP solvers will improve significantly in the future. There is some evidence that this could happen (Monteiro, 2003). The nascent bundle (Helmberg & Rendl, 2000; Helmberg, 2003) and augmented Lagrangian methods (Burer & Vandenbussche, 2006) are particularly promising, as they seem to be able to cope with thousands of vertices in the conflict graph.

Notwithstanding the caveat above, SDP provides some of the strongest known relaxations in timetabling. An extension of theta to bounded graph colouring gives a useful lower bound on the number of periods required in the timetable, considering the conflict graph and the number of rooms. More complex relaxations seem to allow for the optimisation over the assignments of events to periods and rooms as well.

References

Alizadeh, F. (1995). Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, *5*(1), 13–51.

Bellman, R., & Fan, K. (1963). On systems of linear inequalities in Hermitian matrix variables. In *Proc. Sympos. Pure Math., Vol. VII* (pp. 1–11). Providence, R.I.: Amer. Math. Soc.

Burer, S., & Vandenbussche, D. (2006). Solving lift-and-project relaxations of binary integer programs. *SIAM J. Optim.*, *16*(3), 726–750 (electronic).

Burke, E. K., Mareček, J., Parkes, A. J., & Rudová, H. (2010). Decomposition, reformulation, and diving in university course timetabling. *Comput. Oper. Res.*, *37*(1), 582–597.

Dukanovic, I., & Rendl, F. (2004). Combinatorial tricks and Lovasz theta function applied to graph coloring. In *Proc. PATAT 2004* (p. 479).

Dukanovic, I., & Rendl, F. (2007). Semidefinite programming relaxations for graph coloring and maximal clique problems. *Math. Program.*, *109*(2-3, Ser. B), 345–365.

Fazel, M., Hindi, H., & Boyd, S. (2004). Rank minimization and applications in system theory. In *American control conference* (pp. 3273–3278). AACC.

Goemans, M. X., & Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, *42*(6), 1115–1145.

Helmberg, C. (2003). Numerical evaluation of SBmethod. *Math. Program.*, *95*(2, Ser. B), 381–406. (Computational semidefinite and second order cone programming: the state of the art)

Helmberg, C., & Rendl, F. (2000). A spectral bundle method for semidefinite programming. *SIAM J. Optim.*, *10*(3), 673–696.

Karger, D., Motwani, R., & Sudan, M. (1998). Approximate graph coloring by semidefinite programming. *J. ACM*, *45*(2), 246–265.

Kleinberg, J., & Goemans, M. X. (1998). The Lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discrete Math.*, *11*(2), 196–204.

Löfberg, J. (2004). Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proc. of CACSD*. Taipei, Taiwan.

Lovász, L., & Schrijver, A. (1991). Cones of matrices and set-functions and 0-1 optimization. *SIAM J. Optim.*, *1*(2), 166–190.

Mobasher, A., & Khandani, A. K. (2007). Matrix-lifting semi-definite programming for decoding in multiple antenna systems. *CoRR*, *abs/0709.1674*.

Mobasher, A., Taherzadeh, M., Sotirov, R., & Khandani, A. K. (2007). A near-maximum-likelihood decoding algorithm for MIMO systems based on semi-definite programming. *IEEE Trans. Inform. Theory*, *53*(11), 3869–3886.

Monteiro, R. D. C. (2003). First- and second-order methods for semidefinite programming. *Math. Program.*, *97*(1-2, Ser. B), 209–244.

Skutella, M. (2001). Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM*, *48*(2), 206–242.

Sturm, J. F. (1999). Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Softw.*, *11/12*(1-4), 625–653.

Suzuka, A., Miyashiro, R., Yoshise, A., & Matsui, T. (2007). The home-away assignment problems and break minimization/maximization problems in sports scheduling. *Pac. J. Optim.*, *3*(1), 113–133.

Szegedy, M. (1994). A note on the theta number of Lovász and the generalized Delsarte bound. In *Sfcs '94: Proceedings of the 35th annual symposium on foundations of computer science* (pp. 36–39). Washington, DC, USA: IEEE Computer Society.

Wolkowicz, H., Saigal, R., & Vandenberghe, L. (Eds.). (2000). *Handbook of semidefinite programming*. Boston, MA: Kluwer Academic Publishers. (Theory, algorithms, and applications)

Yang, H., Ye, Y., & Zhang, J. (2003). An approximation algorithm for scheduling two parallel machines with capacity constraints. *Discrete Appl. Math.*, *130*(3), 449–467.

A general approach for exam timetabling: a real-world and a benchmark case

Peter Demeester · Patrick De Causmaecker · Greet Vanden Berghe

1 Introduction

We discuss, model and tackle two examination timetabling problems. The first is a real-world case while the latter is a well-known benchmark problem. Both are solved with the same hyper-heuristics approach. Unlike meta-heuristics, in which the search is executed on the space of solutions, hyper-heuristics operate on a search space of heuristics [Burke et al., 2003]. Hyper-heuristics were originally introduced for automating the low-level heuristics' selection, for example by applying machine learning techniques [Burke et al., 2008]. The low-level heuristics employed in both examination timetabling cases are built so that each of them can individually solve one specific part of the problem. By combining the low-level heuristics, the particular properties of each of them can be exploited to solve the problem.

Leaving the cost function aside, both approaches only differ in the low-level heuristics.

Peter Demeester
KaHo Sint-Lieven, Departement Industrieel Ingenieur, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
Tel.: +32-92658610
Fax: +32-92256269
E-mail: Peter.Demeester@kahosl.be

Patrick De Causmaecker
K.U. Leuven, campus Kortrijk, Computer Science and Information Technology, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
Tel.: +32-56246002
Fax: +32-56246052
E-mail: Patrick.DeCausmaecker@kuleuven-kortrijk.be

Greet Vanden Berghe
KaHo Sint-Lieven, Departement Industrieel Ingenieur, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
Tel.: +32-92658610
Fax: +32-92256269
E-mail: Greet.VandenBerghe@kahosl.be

2 Problem Description

First, the hyper-heuristics framework is applied to a real-world examination timetabling problem at the School of Engineering of KaHo Sint-Lieven in Gent (Belgium). The duration of a typical examination schedule is 4 weeks, which corresponds to 40 time slots of four hours each. In Belgium, there is a distinction between oral and written exams. All written exams of the same subject should be organized in the same time slot, while the organization of oral exams is a bit more complex. The maximum number of examinees per time slot for oral exams is 20. This means that if, for example, 200 students take the course, at least 10 oral exams at different time slots should be organized.

The hard constraints of the KaHo examination problem are:

- a student cannot take more than one exam per time slot;
- the number of students assigned to a room cannot exceed its capacity;
- all exams should be organized within the planning horizon of four weeks.

The corresponding soft constraints are:

- All written exams of the same subject should be scheduled in the same time slot.
- Oral and written exams should not be merged into the same room.
- All oral exams should be scheduled such that the maximum number of examinees per timeslot is 20. Lecturers who take oral exams cannot examine more than one group at the same time.
- Students should have sufficient study time between two consecutive exams. At KaHo Sint-Lieven, the minimum study time between two consecutive exams for a student should be at least 3 time slots.

This problem is of particular interest since the manual planner actually needed 48 time slots to organize all exams. He needed to incorporate time slots on Saturdays into the schedule in order to arrange the exams in a 4 weeks period.

In order to compare the hyper-heuristic's performance with the state of the art, we also have applied it to the data sets of the examination timetabling track of the 2007 International Timetabling Competition (ITC 2007) [McCollum et al., 2009]. The hard constraints of the ITC 2007 exam timetabling track are:

- a student cannot attend more than one exam per time slot;
- an exam cannot be split over several rooms;
- the room's capacity cannot be exceeded;
- some exams require rooms with special properties;
- since every exam has a duration, its duration should be less than or equal to the duration of the selected time slot where it is assigned to;
- some exams should be scheduled before, after, at the same time or not at the same time as other exams.

As can be deduced from the hard constraints, the time slots have different durations. Also, the order of the exams is important. These two constraints do not apply to the KaHo problem.

The following soft constraints should be taken into account:

- two exams taken by the same student should not be scheduled on the same day or in two consecutive time slots;
- exams should be spread as much as possible;

– exams with different durations should not be assigned to the same room;
– large exams should be scheduled early in the timetable;
– some of the time slots in the examination timetable should be avoided;
– some of the rooms should be avoided for examination.

The problems have some soft constraints in common, but there are also differences. The distinction between oral and written exams at KaHo is not present in the ITC 2007 examination timetabling track. On the other hand, the ITC 2007 examination timetabling track demands that large exams should be scheduled in the beginning of the examination period, and that some of the time slots and rooms should preferably be avoided.

3 Solution Approach

A typical hyper-heuristics framework consists of a *heuristic selection* mechanism and *move acceptance* criteria [Özcan et al., 2008]. The heuristic selection mechanism that is applied in both examination timetabling cases is *simple random*. This is actually the simplest selection mechanism, since it randomly selects a low-level heuristic from a list of low-level heuristics. Concerning the move acceptance criteria, we experiment with four meta-heuristics: *simulated annealing*, *great deluge*, *steepest descent*, and *late acceptance* [Burke and Bykov, 2008].

Both problems share the same solution representation: a two dimensional matrix, of which the rows represents the rooms, and the columns the time slots. A room-time slot combination can hold several exams.

Regarding the examination timetabling problem at KaHo, the following low-level heuristics are employed:

– move a randomly chosen exam to a random room-time slot combination;
– move a randomly chosen exam to a random room within the same original time slot;
– move a randomly chosen exam to a random time slot while maintaining the original room.

The hyper-heuristic approach finds feasible solutions satisfying all the soft constraints within only 40 time slots. The best performing move acceptance criteria appear to be simulated annealing and late acceptance.

Due to the extra constraints of the ITC 2007 case, additional low-level heuristics tackling them in particular had to be introduced. On top of the low-level heuristics that were already present in the KaHo approach, the following constraints are also applied to the ITC 2007 case:

– a randomly chosen exam is moved to the same room but to a time slot that introduces no extra period penalty;
– a randomly chosen exam is moved to the same time slot but to a room that introduces no extra room penalty;
– the size of a randomly chosen exam is analyzed. If it is recognized as a large exam, it is moved to a time slot in the beginning of the examination period.

In fact, the ITC 2007 problem could also be solved with only the KaHo low-level heuristics, but preliminary experiments showed that the quality of the solutions was improved by introducing the extra low-level heuristics. Besides the extra low-level heuristics, both approaches only differ in their respective cost functions, since both problems

consider other constraints. Actually, both cost functions consist of a linear combination of the violations of the soft constraints and those hard constraints that cannot be expressed in the model. The remaining parts of both applications are the same. For more details we refer to [Demeester, 2010].

4 Conclusion and Future Work

With the general approach that was originally developed for tackling a real-world problem, we obtain results that are competitive with those generated during the competition. In future research we plan to replace the simple random heuristic selection mechanism by a more intelligent one, based on for example a learning automaton [Misir et al., 2009].

Acknowledgement

This work was partially supported by the IWT/SBO 060837 (Dicomas) project.

References

E.K. Burke and Y. Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In E.K. Burke and M. Gendreau, editors, *Proceedings of the The 7th International Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada, August 2008.

E.K. Burke, E. Hart E., G. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Handbook of Meta-Heuristics*, chapter Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer Academic Publishers, 2003.

E.K. Burke, M. Misir, G. Ochoa, and E. Özcan. Learning heuristic selection in hyperheuristics for examination timetabling. In *Proceedings of 7th International Conference of Practice and Theory of Automated Timetabling (PATAT08)*, Montreal, Canada, 2008.

Peter Demeester. *Heuristic approaches for real world timetabling problems in education and health care*. PhD thesis, K.U. Leuven, 2010.

B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A.J. Parkes, L. Di Gaspero, R. Qu, and E.K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, Articles in Advance:1–11, 2009.

M. Misir, T. Wauters, K. Verbeeck, and G. Vanden Berghe. A new learning hyperheuristic for the traveling tournament problem. In *Proceedings of the 8th Metaheuristic International Conference (MIC09)*, Hamburg, Germany, July 2009.

E. Özcan, B. Bilgin, and E.E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, January 2008.

A Hybrid LS-CP Solver for the Shifts and Breaks Design Problem

Luca Di Gaspero · Johannes Gärtner · Nysret Musliu · Andrea Schaerf · Werner Schafhauser · Wolfgang Slany

1 Introduction

The problem of designing workforce shifts and breaks patterns is a relevant employee scheduling problem that arises in many contexts, especially in service industries. The issue is to find a minimum number of shifts, the number of workers assigned to them, and a suitable number of breaks so that the deviation from predetermined workforce requirements is minimized.

We tackle this problem by means of a hybrid strategy in the spirit of Large Neighborhood Search, which blends a Local Search based procedure for determining the shifts, with a Constraint Programming model for assigning breaks. This is a preliminary work and experimentation is currently underway on a set of benchmark instances employed in the literature.

2 Problem definition

Formally, we are given a set D of days, which are subdivided into a set of equally long timeslots. The *planning horizon* is therefore a set $\{\tau_1, \tau_2, \tau_3, \ldots, \tau_h\}$ of consecutive *timeslots* at a given time granularity, each belonging to a single day d. Moreover, for each timeslot τ, we are given a *staffing requirement* r_τ, which indicates the number of employees that should be working during timeslot τ.

L. Di Gaspero, A. Schaerf
DIEGM, University of Udine
E-mail: l.digaspero@uniud.it, schaerf@uniud.it

J. Gärtner
Ximes Inc., Austria
E-mail: gaertner@ximes.com

N. Musliu, W. Schafhauser
DBAI, Technische Universität Wien, Austria
E-mail: musliu@dbai.tuwien.ac.at, schafha@dbai.tuwien.ac.at

W. Slany
IST, Technische Universität Graz, Austria
E-mail: wolfgang.slany@tugraz.at

The problem consists in designing the shifts and break patterns, i.e., determining the starting time σ_i and the length λ_i of each shift s_i and, for each day d, the number of employees assigned w_{id} and the start α_{idw} and length β_{ide} of breaks for each employee e. An employee is considered to be working during the timeslots comprised in the shift but not in any of his/her breaks in that shift. More formally, an employee e works on timeslot $\tau \in d$ if $\tau \in [\sigma_i, \sigma_i + \lambda_i]$ and $\tau \notin [\alpha_{idw}, \alpha_{idw} + \beta_{idw}]$.

3 Local Search and Constraint Programming models

In the context of this problem it is useful to define an `Interval` as a structure of two variables `start` and `length`, which entirely determines a shift, a working period or a break.

Local Search deals with a search space composed of a set of shifts $\mathcal{S}$, each of them is characterized by the following decision variables:

- the interval spanned by the shift;
- the number of employee assigned on each day of the planning horizon;
- for each each day the number of breaks each employee has to take.

Notice that Local Search works on a partial representation of the solution, since the breaks are only specified in their number and not in the intervals they span. To complete this representation to a full solution we resort to a Constraint Programming model (described below) whose purpose is to determine the interval variables of each break.

The Neighborhood relations considered are similar to those employed in [1], slightly modified to deal with the addition of the number of breaks, plus some new move dealing directly with the break component. In detail we make use of the following moves:

- *Change Staff*: the staff of a shift in a given day is increased or decreased by one employee.
- *Resize Shift*: the length of a shift is increased or decreased by one timeslot, either on the left-hand side or on the right-hand side.
- *Insert Shift*: insert a new shift in the solution belonging to a given shift type.
- *Merge Shift*: two shifts are merged together and the employees assigned to them are added; the interval of the outcoming shift as well as the number of breaks for each day are the average of those of the two shifts merged.
- *Change Breaks*: the number of breaks of a shift in a given day is increased or decreased by one.

The cost function is the weighted sum of the deviation (excess and shortage, with different weights) from the working requirements at each timeslot plus another weighted component that accounts for the number of shifts employed in the solution. Notice that for an accurate computation of the deviation from the requirements a full solution is needed, therefore the cost function has to be computed only after a full solution has been determined by the CP model.

4 Conclusions and Future work

The proposed idea is still at an early development stage and the solver experimentation is currently underway on a set of benchmark instances available from the literature.

Preliminary results show that this approach can be feasible to find good quality solutions employing a reasonable number of shifts. However, at present we do not have a full understanding about the contribution of each neighborhood to solution quality. Moreover also the CP model could benefit of some improvement, for example by adding implied constraints which allow for a more accurate constraint propagation and performing a principled evaluation of different heuristics for variable and value selection.

References

1. Luca Di Gaspero, Johannes Gärtner, Guy Kortsarz, Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. The minimum shift design problem. *Annals of Operations Research*, 155(1):79–105, 2007.

DIAMANT

RUBEN GONZALEZ-RUBIO
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
UNIVERSITÉ DE SHERBROOKE

1. Introduction

Diamant is a software system used to produce timetables at the Université de Sherbrooke, where it has been in use since 2001. Diamant allows the user to produce course and exams timetables. The timetable production can be done manually or automatically. This paper will detail the evolution of the system and its main features.

2. Motivation

Producing timetables at the Université de Sherbrooke is done at the Faculty level. Each Faculty has its courses, instructors, students and rooms. The courses are offered for the whole year, there are three terms within the year: Winter (January to April), Summer (May to August) and Fall (September to December). A course timetable is prepared for each term in each Faculty. Each Faculty has a different way to build their timetables, some use curriculum-based and some use post enrolment based, another big difference is the slots in the timetable some are 3 hours some are 30 minutes. A exam timetable is prepared for the end of each term, in some cases a mid-term exam timetable is also prepared. The report [GR07] is a detailed account of how timetables are produced at Université de Sherbrooke.

It can be said that the main motivation to develop Diamant was to satisfy the needs of all Faculties.

Initialy, Diamant was developed to replace another system called Saphir, because this system was to old to be updated. Saphir works in MS-DOS, and was programmed in a variant of Pascal. We chose a very conservative approach to creating this new program. It was first required to replace the existing system and then add new features. The program was set up so that Diamant would work like Saphir; using the same files and producing the same files. This process took about 18 months. Diamant has since been through several stages of evolution and is still in continuous progress towards further and better stages of development. Currently, the work is being done in a version adapted to produce timetables for the full term.

Taking in account all differences mentioned above, we believed that to build a system for each Faculty would be far too complex, especially for the maintenance of the program. Therefore, it was decided that only one system would be built, one that can be customized by changing the parameters for each Faculty. The development of the system was done using object oriented programming with Java and using the design patterns [GHJV95] along with

various practices coming from eXtreme Programming [Bec00] or [Mar03] to facilitate the evolution of Diamant.

3. Evolution

When Saphir was operational, there was a central system where data concerning courses, instructors and students were stored. The information for room data was in a separate file. All data was transmitted in files. Therefore, Diamant was a stand alone program which read files coming from the central system, and then produced the timetables when conflicts within the scheduling were eliminated. The timetable was then transmitted to the central system in order to display the produced timetable information in a Web site.

There was no verification of the data when it is entered within the system, this means that the produced files can contain errors that could drive Saphir to crash. In Diamant it has been ensured that all of the data is verified and valid. Once this process has been completed, a Web system called DiamantWeb, that was created for the process of validating data, is used as a new interface where the data entry can be done. A timetable structure can be defined in DiamantWeb, and each Faculty can have a customized timetable, which, could include exams on Saturdays and Sundays. For example: A Faculty can decide that the exams must be scheduled in 5 or 6 day according to their needs.

We are currently working to produce timetables for a whole term. The process begins with a timetable structure where holidays are indicated in such a way so that no courses can be placed on those days. Some courses are not given the same day each week, or the same amount of time. DiamantWeb is easily accessible for many users, this is especially important for instructors so that they can indicate their availability and preferences concerning each course. In the past, this operation was completed using paper, whereby the instructor and the data were entered manually into the system by another person in charge of that particular operation.

The main aspect concerning this program is that DiamantWeb serves as a data entry point where data validation and preparation take place easily. Diamant is then used to produce timetables, and the user can eliminate conflicts manually or through an automatic build, as well as a combination of both methods.

4. Diamant main features

Diamant can produce timetables manually or automatically for the three timetables types:

- exams [MMK$^+$07].
- post enrolment [LPM07].
- curriculum based [DMS07].

This report was written to address the problems determined by the International Timetable Competition ITC 2007. The International Timetable Competition ITC 2007 reported that the current method of creating timetables for the various types were a problem and numerated the ways that their solutions were evaluated, giving points or penalties in order to compare different solutions. The tracks were prepared to offer problems close to the real ones.

In this report the problems that are characterized by the ITC 2007 are used in order to indicate the possibilities of Diamant. Let us present how courses are organized at Université de Sherbrooke in order to justify why some details are different. In order to get a diploma, for example a bachelor in computer engineering a student needs to accumulate 120 credits of a well defined set of courses. Each course represent 3 credits. Normally the 3 credits are given to the student if he or she succeeds in the exam, and reaches the other requirements. Therefore, in order to succeed all exams and requirements, a student must perform different required activities during a term. A course is decomposed in differing events and these events must be scheduled in the weekly timetable. For example: The course Programming101 has 200 enrolled students and the classrooms that are available have room for 100 students, 50 students, and the computer laboratories have space for 25 respectively. This means that the events scheduled could be 2 lectures for groups of a 100 students, 4 groups of 50 students to make problem analysis and finally 8 groups in the computer laboratories to write and validate programs. In some cases the lectures are 3 hours but in others there are 2 hours + 1 hour lectures that are not on the same day. For the course Programming 101 there are 16 events to place in the timetable[1]. Enrolled students in Programming 101 must take one final examination. Only one exam is scheduled for all Programming 101 students. The exam can take place in different rooms but all exams start at the same hour. Exam rooms can be shared by student sets taking two different exams.

The instructors are assigned during timetable process build. No instructor is able to be in two different places at the same time. Normally there is one instructor who is responsible to for the grading of the exams, but he or she can be associated only to certain events, other instructors or assistants are assigned to the remaining events.

In the case of exams, the presence of the instructor responsible for the course is not required. Survey exams are done by assistants.

4.1. **Exam timetable.** In Diamant we can define a timetable composed by n days, each day can have a number of periods with a defined length. The length of the period could between 5 minutes and 12 hours. University exams are all three hours in length. For this problem the user needs a set of exams, a set of students and a set of rooms. The set of students contains for each student all exams that he or she will take[2].

Hard Constraints:

- No student can sit more than one examination at the same time. This is taken into account.
- The capacity of individual rooms cannot be exceeded. The rooms will be filled as much as possible.
- Period lengths are not violated. In our example the exams and periods have the same length. Changing this can be easy because the only thing to add is the length of each exam.

[1]There are 2 lectures groups, each has 2 lectures, 4 analysis groups and 8 laboratory groups.

[2]All courses where the student is enrolled

- Satisfaction of period related constraints e.g. ExamA after ExamB. This is not taken in account this kind of constraint but users can do so manually.
- Satisfaction of room related constraints e.g. ExamA must be in Room1. This could be done at any time. If a user sets two exams in the same room a conflict will be indicated.

The constraints that are taken into account are exams that are in a row and exams set in a day.

4.2. **Post enrolment timetable.** As in exam timetable, in Diamant we can define a timetable where events can be placed. The rooms are classified in categories, this means that the user can specify a room as "classroom" or "laboratory" then the system can take care of the capacity and features of the room. Normally, all slots are equivalent, but there some priorities to need to be decided upon if events should be placed in one particular slot. There are no precedence requirements.

Hard Constraints:

- No student should be required to attend more than one event at the same time. This is respected.
- In each case the room should be big enough for all the attending students and should satisfy all the features required by the event. The first part is respected and partially the second.
- Only one room is put into each room in a time slot. This is taken into account.
- Events should only be assigned to time slots that are pre-defined available for those events. There is no list like that.
- Where specified, events should be scheduled to occur in correct order during the week. There is no list like that.
- An extra requirement that we implement is that no instructor should be required to attend more than one event at the same time. The instructors have an availability that is respected.

We do not take care of specified soft constraints.

4.3. **Curriculum based timetable.** In Diamant we have an auxiliary program to take care of Curriculum based timetables. The auxiliary program generates virtual student sets, respecting the predictions. Furthermore, with these virtual student sets it is possible to take in account sets of events that must be programmed in different slots, because some virtual students follow the same set of courses.

Diamant takes care of all hard constraints for curriculum based timetables. We do not take care of specified soft constraints

5. Conclusion

The system Diamant has been presented and as detailed the system can take care of different types of timetable production. When the user is preparing to build a timetable he or she prepares the data for a specific type of timetable as outlined above, with the associated parameter the system works for this specific type of timetable. Users may enjoy

the facility of the system because they have already learned only one type of system with few commands. We do not fullfil all hard and soft constraints for all types of timetables because they are not requested by Faculties at the Université de Sherbrooke

References

[Bec00] K. Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.

[DMS07] L. Di Gaspero, B. McCollum, and A. Schaef. The second international timetabling competition (itc–2007): Curriculum-based course timetabling(track 3). Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT, Queen's University, Belfast, 2007.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GR07] R. Gonzalez-Rubio. La production et la consultation d'horaires dans une université. Technical report, Université de Sherbrooke, 2007.

[LPM07] R. Lewis, B. Paechter, and B. McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Technical Report ISSN: 1750-6658, Cardiff University, Wales, 2007.

[Mar03] R. C. Martin. *Agile Software Development*. Prentice-Hall, 2003.

[MMK+07] B. McCollum, P. McMullan, E. K..Burke, A. J. Parkes, and R. Qu. The second international timetabling competition: Examination timetable track. Technical Report QUB/IEEE/Tech/ITC2007/Exam, Queen's University, Belfast, 2007.

First International Nurse Rostering Competition 2010

Stefaan Haspeslagh · Patrick De Causmaecker · Martin Stølevik · Andrea Schaerf

Abstract Insert your abstract here. Include keywords, PACS and mathematical subject classification numbers as needed.

Keywords First keyword · Second keyword · More

Introduction

In hospitals much effort is spent producing rosters which are workable and of a high quality for their nurses. Though the Nurse Rostering Problem is known to be a difficult combinatorial optimisation problem of practical relevance, it has received ample attention mainly in recent years.

Building on the success of the two timetabling competitions, ITC2002 and ITC2007 [1], the First International Nurse Rostering Competition (INRC2010) aims to further develop interest in the general area of rostering and timetabling while providing researchers with models of the problems faced which incorporate an increased number of real world constraints.

Stefaan Haspeslagh
CODeS, Department of Computer Science, KULeuven Campus Kortrijk
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: stefaan.haspeslagh@kuleuven-kortrijk.be

Patrick De Causmaecker
CODeS, Department of Computer Science, KULeuven Campus Kortrijk
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: patrick.decausmaecker@kuleuven-kortrijk.be

Martin Stølevik
SINTEF ICT, Department of Applied Mathematics
P.O. Box 124, Blindern, NO-0314 Oslo, Norway
E-mail: martin.stolevik@sintef.no

Andrea Schaerf
DIEGM, University of Udine
via delle Scienze 206, 33100, Udine, Italy
E-mail: schaerf@uniud.it

A first important goal of INRC2010 is to generate new approaches to the associated problems by attracting users from all areas of research. As with many cases in the past, significant advancements have been made in research areas by attracting multi-disciplinary approaches and comparing them on a common ground.

The second important goal is to close the gap which currently exists between research and practice within this important area of operational research. Although for the sake of the competitive element, we do not include all aspects of the 'real world' problem, we do build on the recent developments to introduce significantly more depth and complexity.

A third goal of INRC2010 is to further stimulate debate within the widening rostering and timetabling research community.

The competition is composed of three tracks; called, after the Olympic disciplines, *1. Sprint*, *2. Middle Distance*, and *3. Long Distance*. The tracks differ from each other based on the maximum running times and on the size of the proposed instances, whereas the problem formulation considered is the same throughout the competition. These tracks represent distinct solution settings in practice. Track 1 (Sprint) requires a solution in a few seconds, and it is meant for interactive use. Track 2 (Middle Distance) requires the solution in a few minutes and simulates the practical situation in which the problem has to be solved a few times in a solving session. Track 3 (Long Distance) grants the solver a few hours of running time and is related to overnight solving. The algorithm features are often tuned to the available running time so that the three tracks represent different challenges to the participants. For each track there are three sets of instances. The Early instances are released immediately after the competition launch. The Late instances will be made available two weeks prior to the end of the Competition on 15 April 2010. A number of instances will be kept aside in order to test the best performing algorithms. These are the Hidden datasets and will be released to the community at a later stage once the competition ends.

Below, we provide information about the Nurse Rostering Problem considered in the competition. The competition rules, a precise problem description, and more info about the data formats can be found at the site of the competition[2] and in a technical reports describing the competition[3].

1 The Nurse Rostering Problem

The nurse rostering problem involves assigning shifts to nurses taking several constraints into account. As usual, we consider two levels of constraints:

- **hard constraints**: constraints that must be satisfied
- **soft constraints**: the sets of constraints that should be to satisfied but for which we expect that it will not be possible to satisfy them all

For example, the demand, i.e. the number of shifts to be covered per day, is a hard constraint. Personal preferences of nurses, work regulations, legal issues, ... provide the soft constraints.

A feasible solution is one in which all hard constraints are satisfied. The quality of the solution is measured in terms of soft constraint violations.

First a more detailed description of the problem is given. Second, we elaborate on the hard and soft constraints and the evaluation of the solution.

1.1 Problem Description

The problem consist of the following:

- a roster is made for a number of days for one ward in a hospital
- shift types: a shift type represents a time frame for which a nurse with a certain skill is required. E.g. between 08h30 and 16h30 a head nurse needs to be present.
- for each day and each shift type, the number of required nurses is provided
- the set of contracts representing the work regulations of the nurses. Each nurse works according to exactly one contract. A contract provides the following information:
  - maximum number of assignments:
    the maximum number of shifts that can be assigned to the nurse
  - minimum number of assignments:
    the minimum number of shift that must be assigned to the nurse
  - maximum number of consecutive working days:
    the maximum number of consecutive days on which a shift can be assigned to a nurse
  - minimum number of consecutive working days:
    the minimum number of consecutive days on which a shift must be assigned to a nurse
  - maximum number of consecutive free days:
    the maximum number of consecutive days on which a nurse has no shift assigned
  - minimum number of consecutive free days:
    the minimum number of consecutive days on which a nurse has no shift assigned
  - maximum number of consecutive working weekends
  - maximum number of working weekends in four weeks
  - the number of days off after a series of night shifts
  - unwanted shift patterns:
    e.g. a nurse does not want to work the following shifts in a row: L-E-L (late-evening-late)
- the nurses of the ward
- the nurses' requests:
  - day on/off requests: a nurse can request (not) to work on a certain day
  - shift on/off requests: a nurse can request (not) to work a particular shift type on a certain day

1.2 Constrains and Evaluation Function

We identify both soft and hard constraints. Note that our decision on which constraints are hard and which are soft is rather arbitrary. In practice many different combinations will be found. Furthermore, wards may assign different weights to certain soft constraints in an attempt to produce solutions that are more appropriate for their particular needs.

There are two hard constraints:

- all demanded shift types must be assigned to a nurse;
- a nurse can only work one shift type per day, i.e. no two shift types can be assigned to the same nurse on a day.

A feasible solution is a solution that does not violate any of those two constraints. All other constrains are soft. A formal description of the constraints can be found in the technical report[3]. We provide a solution evaluator at the site[2].

References

1. B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, R. Qu, and E. K. Burke. Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS Journal on Computing, Vol. 22, Issue 1. In press.
2. INRC2010 website: `http://www.kuleuven-kortrijk.be/nrpcompetition`
3. CODES/Technical Report/2010/1 `https://www.kuleuven-kortrijk.be/CODeS/`

A Weighted-Goal-Score Approach to Measure Match Importance in the Malaysian Super League

Nor Hayati Abdul Hamid
e-mail: nhayati@tmsk.uitm.edu.my

Norazan Mohamed Ramli
e-mail: norazan@tmsk.uitm.edu.my

Universiti Teknologi MARA,
40450 Shah Alam, Selangor, MALAYSIA

Graham Kendall
e-mail: gxk@cs.nott.ac.uk

Automated Scheduling, Optimisation and Planning (ASAP) Group,
School of Computer Science, University of Nottingham,
Nottingham NG8 1BB, UK

Naimah Mohd Hussin
Universiti Teknologi MARA Perlis Kampus Arau,
02600 Arau, Perlis, MALAYSIA
e-mail: naimahmh@perlis.uitm.edu.my

1. Introduction

This work is a continuation of our previous work which aims to define the level of importance of each fixture in the Malaysian Super League. Malaysian football is witnessing a decrease in stadium attendance and scheduling fixtures into timeslots so as to maximise the number of supporters is critical to the league administrators. In our previous work (Abdul-Hamid et. al, (2010), we have applied AHP (Analytic Hierarchy Process) in order to calculate the priorities of each fixture.

The aim of this study is to investigate a different method to measure match importance. We propose to measure match importance using a statistical model.

The result will be used as an input to schedule the Super league fixtures with the aim of maximising stadium attendance.

2. Related Work

Wright (2009) recently reviewed 50 years of sports research, with forecasting being one of the categorized activities. Forecasting is usually associated with gambling but in Malaysia betting on matches is prohibited by law. Nonetheless, we consider forecasting as a tool to predict match importance, and thus enable us to schedule more effectively.

There are a number of papers that make use of statistical methods for forecasting. Maher (1982) and Dixon and Coles (1997) use independent Poisson distributions for the number of goals scored by the home and away teams. Dixon and Robinson (1998) developed a model to predict the results of football matches and updated their predictions during the course of a match. Koning (2000) also used a statistical model to assess the balance of a competition. In the work of Koning et al. (2003), they develop a simulation/probability model that identifies the team that is most likely to win a tournament based on a scoring intensity measure.

Min et al. (2008) developed a framework for sports prediction using Bayesian inference and rule-based reasoning, together with an in-game time-series approach. Based on the framework, they developed a football results predictor called FRES (Football Result Expert System).

Other than football, there has been work on forecasting for other sports. Boulier and Stekler (2003) evaluates power scores as predictors of the outcomes of the NFL (National Football League) (1994–2000 seasons). There has also been research on predicting success at the Olympic games. Condon et al. (1999) used linear regression and neural network models to predict a country's success during the Summer Olympic Games, and Heazelwood (2006) used mathematical models to predict elite performance in swimming and athletics at the Olympic games.

For additional information on sports research, the interested reader is referred to Kendall et al. (2010) which references about 160 papers going back over 40 years.

3. Proposed methodology

Based on previous research (Maher 1982, Dixon and Cole 1997, Dixon and Robinson 1998), we can model soccer matches using Poisson distributions, for example, the number of goals scored over a season. There are several factors to be considered when modeling data with a Poisson distribution. For example, competing teams should have comparable ability which in our case is reasonable as we are considering the Malaysian Super League. Other factors include home and away performance whereby (typically) teams perform better on their home ground. The composition of teams is another factor as players change based on their contracts.

We were initially going to adopt the independent Poisson model initially used by Maher (1982) and also applied by Dixon and Coles (1997) and Dixon and Robinson (1998). In this model, the home team's and away team's scores in any one match are independent Poisson variables. In a match between team i and j, let $X_{i,j}$ and $Y_{i,j}$ be the number of goals scored by the home and away sides respectively. We will assume that $X_{i,j}$ is Poisson with mean $\alpha_i \beta_j$, that $Y_{i,j}$ is also Poisson with mean $\gamma_i \delta_j$, and that $X_{i,j}$ and $Y_{i,j}$ are independent. We can then assume that α_i represents the strength of team i's attack when playing at home, β_j the weakness of team j's defence when playing away, γ_i the weakness of team i's defence at home and δ_j the strength of team j's attack away.

However, the Chi-Square Goodness of Fit test on our data showed that $X_{i,j}$ and $Y_{i,j}$ do not follow Poisson distribution. Thus we propose another statistical approach which we call weighted-goal-score, inspired by the work of Norazan et al. (2009) who proposed a weighted approach to downweight suspected outliers. We adapt the approach in the context of goal scores by finding the level of importance of each match using weighted scores. The motivation for using this method is to investigate if replacing the priorities obtained through AHP (the focus of our previous work) with a statistical model can provide comparable, or even superior, results.

The weighted-goal-score is modeled as follows :

Let :

X_i = goals scored for team i in all their home matches.

Y_j = goals scored for team j in all their away matches.

$$H = \sum_{i=1}^{n} X_i$$

$$A = \sum_{j=1}^{n} Y_j$$

where n is the number of teams.

$$w_i = \frac{X_i}{H}$$

$$w_j = \frac{Y_j}{A}$$

Therefore, the level of importance l, of a match between team i and team j is given by

$$l = w_i \,.\, w_j$$

We will use l, as a measure of match importance, in the mathematical model of our previous work (Abdul-Hamid et al., 2010) in order to generate schedules which are attractive to supporters.

We will report our results at the conference.

4. References

Abdul-Hamid N., S. M., Pais T., Kendall G., Subramaniam E.R , Mohd-Hussin N. (2010). A Combined Mathematical Modelling and AHP Approach to Solve Sport Scheduling Problem. Under review.

Boulier, B. L., & Stekler, H. O. (2003). Predicting the outcomes of National Football League games. *International Journal of Forecasting, 19*(2), 257-270.

Condon, E. M., Golden, B. L., & Wasil, E. A. (1999). Predicting the success of nations at the Summer Olympics using neural networks. *Computers & Operations Research, 26*(13), 1243-1265.

Dixon, M. J., & Coles, S. G. (1997). Modelling Association Football Scores and Inefficiencies in the Football Betting Market. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 46*(2), 265-280.

Dixon, M. J., Robinson, M.E. (1998). A birth process model for association football matches. *Journal of the Royal Statistical Society Series D-the Statistician, 47*, 523-538.

Heazlewood, T. (2006). Prediction versus reality: The use of mathematical models to predict elite performance in swimming and athletics at the Olympic Games. *Journal of Sports Science and Medicine, 5*(4), 541-547.

Kendall, G., Knust, S., Ribeiro, C. C., & Urrutia, S. (2010). Scheduling in sports: An annotated bibliography. *Computers & Operations Research, 37*(1), 1-19.

Koning, R. H. (2000). Balance in competition in Dutch soccer. *Statistician, 49*, 419-431.

Koning, R. H., Koolhaas, M., Renes, G., & Ridder, G. (2003). A simulation model for football championships. *European Journal of Operational Research, 148*(2), 268-276.

Maher, M. J. (1982). Modelling association football scores. *Statistica Neerlandica, 36*(3), 109-118.

Min, B., Kim, J., Choe, C., Eom, H., & McKay, R. I. (2008). A compound framework for sports results prediction: A football case study. *Knowledge-Based Systems, 21*(7), 551-562.

Norazan, M. R., Habshah, M., & Imon, A. (2009). Weighted Bootstrap with Probability in Regression. In S. Y. Chen & Q. Li (Eds.), *Proceedings of the 8th Wseas International Conference on Applied Computer and Applied Computational Science - Applied Computer and Applied Computational Science* (pp. 135-141).

Wright, M. B. (2009). 50 years of OR in sport. *Journal of the Operational Research Society, 60*, S161-S168.

Swiss National Ice Hockey Tournament (NLA)

Abstract: The National Swiss Association of Ice Hockey plans every year a fourfold round-robin tournament. In this tournament every of the 12 teams plays 4-times against each team. In every of the 44 rounds each team plays against another team. Furthermore, the 12 teams are partitioned into 3 groups of 4 teams each, and within the groups each team has to play two-times against each other. That means that another 6 rounds have to be scheduled – the so-called "derby rounds". Hence, in total 300 games have to been scheduled within 50 rounds in a season.

Several hard conditions have to be considered:

1. In the first 25 rounds each team must play 2 (respectively 3) games against each other in the other groups (respectively in the same group).
2. Home and away games must alternate as much as possible. (3) Each teams should have the same number of home-game on a Saturday and Sunday if possible.
3. Some "high risk-games" must be fixed in particulary rounds (at a fixed date).
4. At various dates certain stadiums are occupied by other events and game cannot be fixed at this locations.

The goal is to find a schedule that fulfills these condition as much as possible. We shows a mixed integer approach to formulate the problem and solve it with standard MIP-solvers.

An Approximation Algorithm for the Unconstrained Traveling Tournament Problem*

Shinji Imahori[1], Tomomi Matsui[2], and Ryuhei Miyashiro[3]

[1] Graduate School of Engineering, Nagoya University,
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan.
imahori@na.cse.nagoya-u.ac.jp
[2] Faculty of Science and Engineering, Chuo University,
Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan.
matsui@ise.chuo-u.ac.jp
[3] Institute of Engineering, Tokyo University of Agriculture and Technology,
Naka-cho, Koganei, Tokyo 184-8588, Japan.
r-miya@cc.tuat.ac.jp

1 The Unconstrained Traveling Tournament Problem

A deterministic 3-approximation algorithm is proposed for the unconstrained traveling tournament problem, which is a variant of the traveling tournament problem. For the unconstrained traveling tournament problem, this is the first proposal of an approximation algorithm with a constant approximation ratio. In addition, the proposed algorithm yields a solution that meets both the no-repeater and mirrored constraints.

In the field of tournament timetabling, the traveling tournament problem (TTP) is a well-known benchmark problem established by Easton, Nemhauser, and Trick [2]. The present paper considers the unconstrained traveling tournament problem (UTTP), which is a variant of the TTP. In the following, some terminology and the TTP are introduced. The UTTP is then defined at the end of this section.

Given a set $T = \{0, 1, \ldots, n-1\}$ of n teams, where $n \geq 4$ and is even, a game is specified by an ordered pair of teams. Each team in T has its home venue. A double round-robin tournament is a set of games in which every team plays every other team once at its home venue and once in an away game (i.e., at the venue of the opponent). Consequently, $2(n-1)$ slots are necessary to complete a double round-robin tournament.

Each team stays at its home venue before a tournament and then travels to play its games at the chosen venues. After a tournament, each team returns to its home venue if the last game is played as an away game. When a team plays two consecutive away games, the team goes directly from the venue of the first opponent to the venue of another opponent without returning to its home venue.

* The present study was supported in part by Grants-in-Aid for Scientific Research, by the Ministry of Education, Culture, Sports, Science and Technology of Japan.

For any pair of teams $i, j \in T$, $d_{ij} \geq 0$ denotes the distance between the home venues of i and j. Throughout the present paper, we assume that triangle inequality ($d_{ij} + d_{jk} \geq d_{ik}$), symmetry ($d_{ij} = d_{ji}$), and $d_{ii} = 0$ hold for any $i, j, k \in T$.

Denote the distance matrix (d_{ij}) by D. Given a constant (positive integer) $u \geq 3$, the traveling tournament problem [2] is defined as follows.

Traveling Tournament Problem (TTP(u))
Input: A set of teams T and a distance matrix $D = (d_{ij})$.
Output: A double round-robin schedule of n teams such that
C1. No team plays more than u consecutive away games,
C2. No team plays more than u consecutive home games,
C3. Game i at j immediately followed by game j at i is prohibited,
C4. The total distance traveled by the teams is minimized.

Constraints C1 and C2 are referred to as the *atmost* constraints, and Constraint C3 is referred to as the *no-repeater* constraint.

Various studies on the TTP have been conducted in recent years (see [4] for detail), and most of these studies considered TTP(3) [5]. Most of the best upper bounds of TTP instances are obtained using metaheuristic algorithms. On the other hand, little research on approximation algorithms has been conducted for the TTP. Recently, Miyashiro, Matsui, and Imahori [3] proposed a $(2 + O(1/n))$-approximation algorithm for TTP(3). In addition, Yamaguchi, Imahori, Miyashiro, and Matsui [6] proposed an approximation algorithm for TTP(u), where $u \ll n$. For TTP(3), the approximation ratio of [6] is better than that of [3].

The unconstrained traveling tournament problem (UTTP) is a variant of the TTP, in which Constraints C1 through C3 are ignored. In other words, the UTTP is equivalent to TTP($n - 1$) without the no-repeater constraint. Although the UTTP is simpler than the TTP, no approximation algorithm for the UTTP has yet been proposed. The method proposed in [6] cannot be applied to the UTTP because the condition $u \ll n$ is necessary for the method. The method in [3], proposed for TTP(3), can be applied to the UTTP with a few modifications. However, this leads to a $((2/3)n + O(1))$-approximation algorithm, which is not a constant approximation ratio with regard to n.

In the present paper, we propose a deterministic 3-approximation algorithm for the UTTP. In addition, the solution obtained by the algorithm meets both the no-repeater and mirrored constraints, which are sometimes required in practice.

2 Approximation Algorithm

In this section, we describe the proposed approximation algorithm for the UTTP. A key concept of the algorithm is the use of the circle method and a shortest Hamilton cycle. The classical schedule obtained by the circle method satisfies the property such that the orders of opponents in almost all teams are very similar to a mutual cyclic order of teams. Roughly speaking, the proposed algorithm

constructs a short Hamilton cycle passing all venues, and finds a permutation of teams such that the above cyclic order corresponds to the obtained Hamilton cycle.

For a vertex set $V = \{0, 1, \ldots, n-1\}$, let $G = (V, E)$ be a graph such that the distance of edge (i, j) is given by d_{ij} for any $i, j \in V$. First, we assign aliases $t_0, t_1, \ldots, t_{n-1}$ to teams $0, 1, \ldots, n-1$ as follows.

1. For each $v \in V$, compute $\sum_{v' \in V \setminus \{v\}} d_{vv'}$.
2. Let v^* be a vertex that attains $\min_{v \in V} \sum_{v' \in V \setminus \{v\}} d_{vv'}$, and designate the team corresponding to v^* as t_{n-1}.
3. Using the Christofides' 3/2-approximation algorithm for the traveling salesman problem with the triangle inequality [1], construct a Hamilton cycle on the complete graph induced by $V \setminus \{v^*\}$. For the obtained cycle $(v_0, v_1, \ldots, v_{n-2})$, denote the corresponding teams by $(t_0, t_1, \ldots, t_{n-2})$.

Next, we construct a single round-robin schedule. In the following, "schedule without HA-assignment" refers to a "round-robin schedule without the concepts of home game, away game, and venue." Denote the set of $n-1$ slots by $S = \{0, 1, \ldots, n-2\}$. A single round-robin schedule without HA-assignment is a matrix K of which (t, s) element, say $K(t, s)$, denotes the opponent of team t in slot s. Let K^* be a matrix defined by

$$K^*(t,s) = \begin{cases} t_{s-t \bmod n-1} & (t \neq n-1 \text{ and } s - t \neq t \pmod{n-1}), \\ t_{n-1} & (t \neq n-1 \text{ and } s - t = t \pmod{n-1}), \\ t_{s/2} & (t = n-1 \text{ and } s \text{ is even}), \\ t_{(s+n-1)/2} & (t = n-1 \text{ and } s \text{ is odd}). \end{cases}$$

Lemma 1. [6] *The matrix K^* is a single round-robin schedule without HA-assignment. In addition, K^* is essentially equivalent to the classical schedule obtained by the circle method.*

Then, by the mirroring procedure, form K^* into a double round-robin schedule without HA-assignment. Finally, we assign home and away so as to complete a double round-robin schedule as follows:

- for team $t \in \{t_0, t_1, \ldots, t_{n/2-1}\}$, let the games in slots $n+2t-1, n+2t, \ldots, n+2t+n-3 \pmod{2(n-1)}$ be away games, and let the other games be home games.
- for team $t \in \{t_{n/2}, t_{n/2+1}, \ldots, t_{n-2}\}$, let the games in slots $2t-n+2, 2t-n+3, \ldots, 2t$ be away games, and let the other games be home games.
- for team t_{n-1}, let the games in slots $0, 1, \ldots, n-2$ be away games, and let the other games be home games.

The proposed double round-robin schedule, denoted by K^*_{DRR}, satisfies the no-repeater and mirrored constraints.

We now prove the above-mentioned algorithm is a 3-approximation algorithm for the UTTP. Designate the distance of a shortest Hamilton cycle on G as τ. In addition, let the distance of the cycle $(v_0, v_1, \ldots, v_{n-2})$ obtained above be τ'. Note that $\tau' \leq (3/2)\tau$.

Lemma 2. *The following propositions hold for G.*
(1) *For any path of two edges, its distance is bounded by τ.*
(2) *The distance of any Hamilton cycle is bounded by $n\tau/2$.*

In K^*_{DRR}, team t_{n-1} plays $n-1$ consecutive away games, and thus the distance by team t_{n-1} can be bounded by $n\tau/2$ from Lemma 2(2). In addition, analyzing the structure of the proposed schedule reveals the following lemma.

Lemma 3. *Let $l(i,j,k)$ be the distance of path (i,j,k) for $i,j,k \in V$. In K^*_{DRR}, the traveling distance of teams can be bounded by*

$$\begin{cases} \tau' + l(v_0, v^*, v_1) & (t = t_0), \\ \tau' + l(v_t, v^*, v_{t+1}) + l(v_{n-t-1}, v_t, v_{n-t-2}) & (t \in \{t_1, t_2, \ldots, t_{n/2-2}\}), \\ \tau' + l(v_{n/2-1}, v^*, v_{n/2-1}) & (t = t_{n/2-1}), \\ \tau' + l(v_{t-1}, v^*, v_t) & (t \in \{t_{n/2}, t_{n/2+1}, \ldots, t_{n-2}\}), \\ n\tau/2 & (t = t_{n-1}). \end{cases}$$

Although the following lemma is not obvious, we omit the proof due to space limitations.

Lemma 4. *Let v^* be a vertex that attains $\min_{v \in V} \sum_{v' \in V \setminus \{v\}} d_{vv'}$. Then, the following holds: $\sum_{v \in V \setminus \{v^*\}} d_{vv^*} \leq n\tau/4$.*

Theorem 1. *The proposed algorithm is a 3-approximation algorithm for the UTTP.*
Proof. Let the distance of K^*_{DRR} be $d(K^*_{\mathrm{DRR}})$. From Lemmas 2 through 4, we have:

$$\begin{aligned} d(K^*_{\mathrm{DRR}}) &\leq \tau'(n-1) + \sum_{t \in \{t_0, t_1, \ldots, t_{n-3}\}} l(v_t, v^*, v_{t+1}) + l(v_{n/2-1}, v^*, v_{n/2-1}) \\ &\quad + \sum_{t \in \{t_1, t_2, \ldots, t_{n/2-2}\}} l(v_{n-t-1}, v_t, v_{n-t-2}) + n\tau/2 \\ &\leq (3/2)\tau(n-1) + \sum_{v \in V \setminus \{v^*\}} 2d_{vv^*} + \tau + \tau(n/2-2) + n\tau/2 \\ &\leq (3/2)\tau(n-1) + 2(n\tau/4) + \tau + \tau(n/2-2) + n\tau/2 \\ &\leq 3n\tau. \end{aligned}$$

Since $n\tau$ is a lower bound of the distance of any double round-robin schedule, this concludes the proof. □

References

1. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976
2. Easton, K., Nemhauser, G., Trick, M.: The traveling tournament problem: description and benchmarks. *Lecture Notes in Computer Science* **2239** (2001) 580–585

3. Miyashiro, R., Matsui, T., Imahori, S.: An approximation algorithm for the traveling tournament problem. *Annals of Operations Research*, to appear
4. Rasmussen, R.V., Trick, M.A.: Round robin scheduling — a survey. *European Journal of Operational Research* **188** (2008) 617–636
5. Trick, M.: Challenge traveling tournament problems. Web page, as of 2010 http://mat.gsia.cmu.edu/TOURN/
6. Yamaguchi, D., Imahori, S., Miyashiro, R., Matsui, T.: An improved approximation algorithm for the traveling tournament problem. *Lecture Notes in Computer Science* **5878** (2009) 679–688

Data Formats for Exchange of Real-World Timetabling Problem Instances and Solutions

Discussion Session (Abstract)

Jeffrey H. Kingston

Keywords Timetabling · Data Formats · Benchmarks

1 Abstract

The problem of exchanging timetabling data has been a perennial topic at PATAT conferences. Indeed it was discussed at the very first (Cumming and Paechter 1995).

The principal difficulty is the large variety of kinds of requirements. To quote from a recent paper (Post et al. 2008): 'This complexity with the specification of the problem has been addressed in several ways. Some papers have tried to generalize and unify the constraints (Chand 2004; Kitagawa and Ikeda 1988). Others have adapted existing technologies in which constraints may be expressed, such as XML and the semantic web (Custers et al. 2005; de Causmaecker et al. 2000, 2002; Özcan 2003), or object-oriented modeling and frameworks (Grobner et al. 2003; Ranson and Ahmadi 2006). Others have expressed constraints as logic expressions within specifically designed specification languages (Burke et al. 1998; Kingston 2001; Mata et al. 1997). There has been at least one attempt to simply enumerate every possible constraint (Reis and Oliviera 2001).

'Another approach is to restrict the problem domain to one particular kind of timetabling, then use judicious simplification to further reduce the specification burden while maintaining the essence of the problem. The Carter data sets for examination timetabling (Carter et al. 1996) omit many details, notably all data related to rooms, and similar simplifications appear in the Traveling Tournament Problem (Easton et al. 2001) and the International Timetabling Competition (Paechter 2003). These are some of the most successful examples of timetabling data exchange. However, judicious simplification has been criticized for contributing to the gap between research and practice (Burke et al. 2006), at least in examination timetabling; and the data transfer has almost always been in one direction only.'

The paper from which this quotation is taken goes on to define an XML format for exchanging real high school timetabling problem instances and solutions. Since that paper

Jeffrey H. Kingston
School of Information Technologies
The University of Sydney, NSW 2006, Australia
http://www.it.usyd.edu.au/~jeff
E-mail: jeff@it.usyd.edu.au

was written, the format has been refined, and a set of instances and solutions from widely varying institutions around the world has been collected and stored in the format (Post 2009). An evaluator, which compares solutions against instances and produces badness values, has been made available as a web service (Kingston 2009).

The purpose of this discussion session is to explore the idea that the general approach taken by the high school timetabling project should be adopted for real-world data exchange in other sub-disciplines of timetabling.

Two key points define this approach. First, the format was developed in consultation with several groups of researchers, and is avowedly inclusive: if the accurate expression of real-world instances requires that certain features be present, there is a commitment that they will be added. Second, the format is not overly general: it is limited to high school timetabling, and it does not attempt to define constraints by logic expressions that researchers are unlikely to want to read and interpret. Instead, there is a fixed list of constraint types of predetermined meaning. At the time of writing there were 15 constraint types; more will be added as required.

It might seem that this proposal offers nothing new compared with the formats used by the Carter data set or the International Timetabling Competition cited above. The new point is the demonstration that it is feasible to specify *real-world instances* from *disparate sources* in *complete detail*, by limiting attention to a single sub-discipline and a fixed list of constraint types.

The high school project was influenced by an earlier project in nurse rostering. After the need for exchanging nurse rostering data had become clear (Burke et al. 2004; Cheang et al. 2003), an XML data format was defined and a web site (Curtois 2009) set up in the year 2005 (personal communucation from T. Curtois). This is the earliest example known to this author of real-world instances and solutions from disparate sources being brought together.

Another point for discussion is whether specific features of the high school format could be re-used in formats for other sub-disciplines. To support this discussion, the remainder of this abstract introduces the high school format.

The format allows any number of instances of the high school timetabling problem to be stored in one file, along with any number of sets of solutions to those instances, each set contributed by one researcher. The file may thus be a comprehensive archive, and the evaluator may produce tables comparing the solutions, of the kind frequently seen in research papers. The existing evaluator (Kingston 2009) does this.

Each instance consists of a `<Times>` section (the concrete format is XML) listing the times of the cycle in chronological order, a `<Resources>` section listing the resources (typically but not necessarily student groups, teachers, and rooms), an `<Events>` section listing the events, and a `<Constraints>` section listing the constraints. Named sets of times, resources, and events may be defined.

An event represents a meeting of arbitrary but fixed duration between any number of resources, beginning at a particular time. It contains one time variable and any number of resource variables, each of which may either be preassigned or left for solutions to assign, subject to constraints.

Each constraint contains its type (one of the 15 types mentioned above), whether it is a hard or soft constraint, its weight, the resources, events, or sets of events that it applies to, and possibly other parameters specific to the constraint type. For example, there is an `<AvoidResourceClashes>` constraint which would typically state that it applies to all resources, and whose meaning, defined implicitly, is the usual one that those resources should not have timetable clashes. If they do, the constraint indicates the penalty for each occurrence. Other constraints require sets of events to have the same starting time, impose domain

constraints on time and resource variables (e.g. requiring staff members to be suitably qualified), and so on. Everything affecting the evaluation of solutions appears explicitly as a constraint, allowing solutions to be evaluated unambiguously; this is the function performed by the web service (Kingston 2009).

A solution is a simple collection of assignments to the time and resource variables of the events of the corresponding instance.

The basic concepts of times, resources, events, and constraints seem applicable to other sub-disciplines, but the constraint types naturally vary. A recent comprehensive formulation of the curriculum-based university course timetabling problem (Cesco et al. 2008) is very similar to high school timetabling, the main addition being a limit on travel time between consecutive classes. Examination timetabling has times, resources (students and rooms), and events (examinations); its constraint types include constraints that limit the number of examinations that a student is required to attend over a short period of time. In the personnel rostering format (Curtois 2009) mentioned earlier, 'shifts' are times, 'shift types' are sets of times, 'employees' are resources, and their 'skills' are sets of resources; there are complex constraints on the timetable of each resource (at most one night shift per week, for example). Sports scheduling has times, resources (teams and venues), and events (matches); its constraints include the cost of travel between venues.

Despite some common structure, unifying all these problems into one is not advocated. It would be difficult in practice and would yield no practical benefit.

References

Burke EK, Kingston JH, Pepper PA (1998) A standard data format for timetabling instances. In: Practice and Theory of Automated Timetabling II, Burke EK and Carter M (Eds.), Springer Verlag Lecture Notes in Computer Science 1408:213–222

Burke EK, De Causmaecker P, Vanden Berghe G, Van Landeghem H (2004) The state of the art of nurse rostering. Journal of Scheduling 7:441–499

Burke EK, McCollum B, McMullan P, Qu R (2006) Examination timetabling: a new formulation. In: Proceedings of the Sixth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), Brno, 373–375

Carter M, Laporte G, Lee ST (1996) Examination timetabling: algorithmic strategies and applications. Journal of the Operational Research Society 47:373–383

Cesco F, Di Gaspero L, Schaerf A (2008) Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, and results. In: The 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008), Montreal

Chand A (2004) A constraint based generic model for representing complete university timetabling data. In: Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004), Pittsburgh, 125–148

Cheang B, Li H, Lim A, Rodrigues B (2003) Nurse rostering problems: a bibliographic survey. European Journal of Operational Research 151:447–460

Cumming A, Paechter B (1995) Standard formats for timetabling data, Unpublished discussion session at the First International Conference on the Practice and Theory of Automated Timetabling, Edinburgh

Curtois T (2009), Personnel scheduling data sets and benchmarks, `http://www.cs.nott.ac.uk/~tec/NRP/`. Accessed 2009

Custers N, De Causmaecker P, Demeester P, Vanden Berghe G (2005) Semantic components for timetabling. In: Practice and Theory of Automated Timetabling V, Burke EK and Trick M (Eds.), Springer Verlag Lecture Notes in Computer Science 3616:17–33

De Causmaecker P, Demeester P, De Pauw-Waterschoot P, Vanden Berghe G (2000) Ontology for timetabling. In: Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000), Konstanz, 481–482

De Causmaecker P, Demeester P, Lu Y, Vanden Berghe G (2002) Using web standards for timetabling. In: Proceedings of the Fourth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Gent, 238–257

Easton K, Nemhauser GL, Trick MA (2001) The travelling tournament problem: description and benchmarks. In: Principles and Practice of Constraint Programming (CP 2001), Springer Verlag Lecture Notes in Computer Science 2239:580–585

Gröbner M, Wilke P, Büttcher S (2003) A standard framework for timetabling problems. In: Practice and Theory of Automated Timetabling IV, Burke EK and De Causmaecker P (Eds.), Springer Verlag Lecture Notes in Computer Science 2740:24–38

Kingston JH (2001) Modelling timetabling problems with STTL. In: Practice and Theory of Automated Timetabling III, Burke EK and Erben W (Eds), Springer Verlag Lecture Notes in Computer Science 2079:309–321

Kingston JH (2009) The HSEval high school timetable evaluator, `http://www.it.usyd.edu/au/~jeff`.

Kitagawa F, Ikeda H (1988) An existential problem of a weight-controlled subset and its application to school timetable construction. Discrete Mathematics 72:195–211

Da Mata JM, de Senna AL, de Andrade MA (1997) Towards a language for the specification of timetabling problems. In: Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling (PATAT'97), Toronto, 330–333

Özcan E (2003) Towards an XML-based standard for timetabling problems: TTML. In: Multidisciplinary Scheduling: Theory and Applications (First International Conference, MISTA '03, Nottingham, Selected Papers), 163–185

Paechter B (2003) International timetabling competition. `http://www.idsia.ch/Files/ttcomp2002/`

Post G, Ahmadi S, Daskalaki S, Kingston J et al. (2008) An XML Format for benchmarks in high school timetabling. In: The 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008), Montreal

Post G (2009) Home Page, `http://wwwhome.math.utwente.nl/postgf/`

Ranson D, Ahmadi S (2006) An extensible modelling framework for the examination timetabling problem. In Practice and Theory of Automated Timetabling VI, Burke EK and Rudová H (Eds.), Springer Verlag Lecture Notes in Computer Science 3867:383–393

Reis LP, Oliviera E (2001) A language for specifying complete timetabling problems. In: Practice and Theory of Automated Timetabling III, Burke EK and Erben W (Eds), Springer Verlag Lecture Notes in Computer Science 2079:322–341

Solving the General High School Timetabling Problem
Abstract

Jeffrey H. Kingston

Keywords High School Timetabling · Benchmarks

1 Abstract

A set of instances of the high school timetabling problem, taken from a number of countries, has recently appeared (Post 2009). These instances are expressed in complete detail in a common XML format (Post et al. 2008, 2010). They make it possible, for the first time, to tackle the high school timetabling problem in its full generality, that is, as it really exists in high schools around the world.

This abstract describes work in progress on KHE, a general solver for high school timetabling problems. KHE is a software library, written in C, that will eventually be released under a GNU public licence. It brings together several themes from the author's previous work and elsewhere.

KHE follows the XML format closely. Its data types parallel the categories of the format (including instances, times, resources, events, and constraints), so that building an internal representation of an XML instance is straightforward. It offers several basic operations for modifying solutions, including assigning and deassigning times and resources, changing the domains of time and resource variables, and splitting and merging events. Tree searches, local searches, and other algorithms may be built on these basic operations. An efficient hand-coded constraint network monitors the solution state and reports its current badness in terms of violations of the constraints defined by the instance.

The spread of multi-processor computers has made it important to allow for coarse-grained parallelism when solving timetabling problems. KHE does this by ensuring that instances are immutable after creation (so that they may be shared) and that multiple solutions can be created and operated on independently in parallel.

KHE supports the features of the author's KTS timetabling system (Kingston 2007a) as well as the XML format, and will eventually replace the current KTS solver. Internally it is

Jeffrey H. Kingston
School of Information Technologies
The University of Sydney, NSW 2006, Australia
`http://www.it.usyd.edu.au/~jeff`
E-mail: jeff@it.usyd.edu.au

based on the author's layer tree data structure (Kingston 2007b) for hierarchical timetabling, enhanced with operations for assigning resources as well as times (Kingston 2008).

Compared to the author's earlier general solver (Kingston 2001), KHE is more efficient, being written in C and employing ordinary function calls instead of command objects to carry out its basic operations. And whereas the earlier solver handled assignment-type problems generally, KHE benefits from a specific focus on high school timetabling.

A major question raised by this work is whether the XML format can be supported in full generality without an unacceptable loss of efficiency. For example, layer trees use unweighted bipartite matching to monitor resource assignment, but the XML format allows each possible resource assignment to have its own integer cost, so that the equivalent monitoring requires edge-weighted bipartite matching, which is significantly more expensive. New kinds of constraints, such as limits on idle times, call for incremental algorithms which evaluate them efficiently as the solution changes.

References

Kingston JH, Lynn BYS (2001) A software architecture for timetable construction. Practice and Theory of Automated Timetabling III (Third International Conference, PATAT2000, Konstanz, Germany, August 2000, Selected Papers), Springer Lecture Notes in Computer Science 2079:342–350

Kingston JH (2007a) The KTS high school timetabling system. Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Brno, Czech Republic, August 2006, Selected Papers), Springer Lecture Notes in Computer Science 3867:308–323

Kingston JH (2007b) Hierarchical timetable construction. Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Brno, Czech Republic, August 2006, Selected Papers), Springer Lecture Notes in Computer Science 3867:294–307

Kingston JH (2008) Resource assignment in high school timetabling. 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008), Montreal

Post G (2009) Home Page, `http://wwwhome.math.utwente.nl/postgf/`. Accessed Dec 2009

Post G, Ahmadi S, Daskalaki S et al. (2008) An XML Format for Benchmarks in High School Timetabling. 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008), Montreal

Post G, Kingston JH, Ahmadi S et al. (2010) An XML Format for Benchmarks in High School Timetabling II. Submitted to 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT2010), Belfast

Towards an Integrated Workforce Management System

Dario Landa-Silva · Arturo Castillo · Leslie Bowie · Hazel Johnston

Abstract We describe progress towards a workforce management system in which personnel scheduling is integrated with other important processes such as payroll processing, attendance and absence recording, staffing forecast and planning, etc. Our focus is on customer-oriented sectors in which a considerable proportion of the available workforce is part-time. First, we discuss the importance of having an integrated workforce management strategy and then, we address the particular aspects of shift pattern design and staff allocation using a fast-food restaurant and a leisure centre as examples.

Keywords workforce management · personnel scheduling · shift design · task allocation

1 Introduction

In addition to making an efficient use of the available personnel, a workforce management strategy should align personnel skills to customer needs in order to help improving the overall customer satisfaction. Moreover, adequate integration of the scheduling process helps businesses to quickly respond to customer needs that could affect the overall staffing strategy. The integration of workforce scheduling within an overall workforce management strategy is crucial to achieve high level workforce utilisation. According to (Belkin 2009), many retailers do not have a workforce scheduling process in place and many do not integrate scheduling within the overall workforce management strategy.

Many businesses with wide working hours and varying requirements use a form of manual shift schedules to identify which staff will be working at any period. Simple rotation procedures are used to ensure fairness. But in many scenarios, it is difficult to make an accurate estimation of workforce demand and the shifts lead to over or under manning and a high reliance on temporary staffing from agencies. Our research has shown that unless

D. Landa-Silva
ASAP Research Group
School of Computer Science, University of Nottingham
E-mail: dario.landasilva@nottingham.ac.uk

A. Castillo, L. Bowie, H. Johnston
Midland HR
E-mail: (a.castillo,leslie.bowie,hazel.johnston)@midlandhr.co.uk

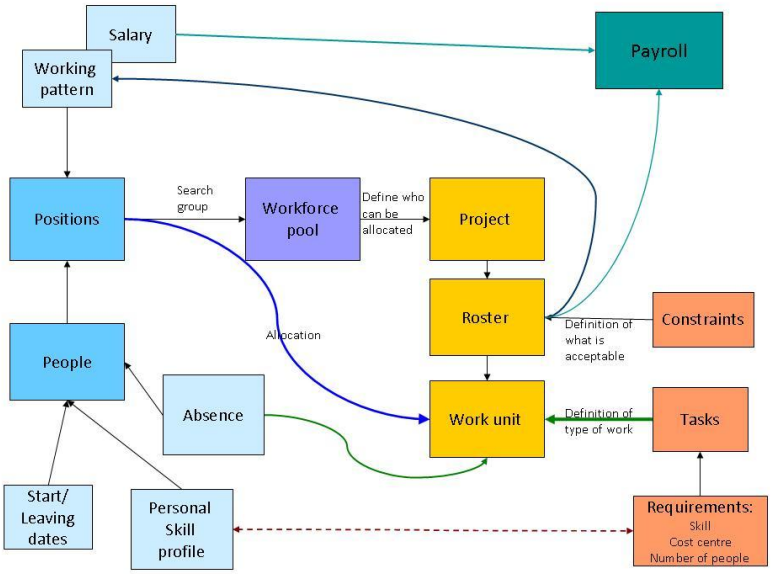


Fig. 1 Holistic approach to workforce management by Midland HR.

the workload pattern is highly regulated, non-standard shift patterns are required. But truly variable shift patterns can be unpopular with staff and a compromise is required between the staff's need for some sort of pattern and the business requirement to ensure minimum costs.

Midland HR is a UK company specialised in providing software solutions for HR management including payroll processing, attendance/absence management, staffing forecast and planning, HR administration, workforce analytics, etc. The company aims to offer a workforce management integral solution that focuses on putting the right employee in the right place at the right time. Fig. 1 depicts the main components used by our system, *iTrent*, when tackling rostering problems. Data is entered as part of concurrent non-rostering processes and by different parts of the business. In our approach every allocation is made to a position instead of to a person. This brings flexibility when replacing staff and allows future, already planned rosters, to remain the same.

In general, we take into account employee skills and availability. Every employee has a defined profile which could vary from time to time depending on career progression. This profile tries to encapsulate employee skills that are of significant use to the business. Similarly, every duty, task or assignment must have a record of the necessary skills needed to perform it. Positions are then grouped in workforce pools which are completely user defined, depending on the scenario. For management purposes, a set of rosters are organised in projects. A project is a meaningful way of assigning common task and workforce pools to several rosters. Finally, a work unit is a performed task in time, i.e. a pair duty-timeslot. Work-units can be allocated or not. Constraints are taken into account when assigning staff to rosters. Currently we have a reduced set of implemented yet flexible constraints but a long list of future ones depending on the industry sector. Our aim is to integrate this rostering system into a general HR and Payroll solution, and therefore, other aspects such as cost of staff including (salary, pension, etc), absences and skills play a significant part in the system.

2 Workforce Scheduling Approach

The process of workforce scheduling usually involves several stages such as: 1) demand modelling, 2) shift pattern design, 3) duty assignment and 4) staff assignment (Ernst et al 2004; Tien and Kamiyama 1982). Using a fast-food restaurant and a leisure centre as example scenarios, we give an insight into our work towards developing an integrated solution.

2.1 Fast-Food Restaurant Scenario

The rostering period is 1 week. The manager receives from the head office the estimated work demand in the form of a forecast with the number of employees needed in each timeslot of the week to ensure that the restaurant runs efficiently. Following this forecast the manager should assign shifts and duties to employees in order to construct the roster. The aim is to follow the head office forecast as closely as possible while constructing a practical roster that satisfies staff preferences and payroll budget. The forecast gives the estimated number of employees needed for each of the duty types such as: 'back-of-house' (BoH) staff (working in the kitchen), 'front-of-house' (FoH) staff (interacting with customers), 'management' (MG) staff, etc. Most employees are part-time working a number of hours per week according to their particular availability. A number of constraints exist in this scenario, but the ones that relate directly to the design of shift patterns are: (a) an employee works a maximum number of hours per week; (b) an employee works a maximum number of hours per day; (c) the maximum length of a shift is 8 hours; (d) the minimum length of a shift is 3 hours; (e) full-time employees must have 1 day-off per week; (f) shift splits are permitted with a 1 hour break minimum; (g) a 'closing' shift must not be followed by an 'opening' shift (nights not considered a break). The design of shift patterns (Musliu et al 2004; Di Gaspero et al 2007) is very important in this scenario given the variability in workforce demand and availability.

2.2 Leisure Centre Scenario

The rostering period is 1 week but currently the centre uses a combination of 5 week, 4 week and 3 week shift patterns with a manual rotation of staff. Since the leisure centre publishes opening hours for many of its facilities, the boundaries of duties are known. A class schedule is also agreed and published on a 10 week basis, approximating to school terms. There are specific tasks that happen at set times daily (e.g. opening and closing the centre which requires 2 staff and manning the reception) and weekly (Tuesday afternoon staff training session). However, within this seemly simple staffing requirement there are a number of variables: (a) the public can book private parties in the pool and/or children's play area (these require additional staff); (b) customer demand for public sessions can lead to more lifeguards being required; (c) special sport events in the pool or other sports facilities. Weekend and school holidays follow different routines. Staff fall into 3 categories: salaried staff (managers, lifeguards, dry-side staff and receptionists), contracted staff (class instructors) and ocassional staff (weekend staff and holiday staff). Most employees are part-time working an agreed number of hours per week. Adjustments to the roster are common because of staff absences and because staff can change their assigned duty under certain circumstances. While assigning staff and repairing changes to the roster are key in this scenario, designing shift patterns is not because shifts are static most of the times (althouhg there is also a desire to review and improve on the static rotating shift patterns being used).

Fig. 2 Structure to facilitate integration between scheduling and other processes such as demand modelling and attendance/absence management.

2.3 Shift Pattern Design and Work Unit Allocation

To facilitate the integration of the scheduling process within the workforce management strategy, we employ the structure shown in Figure 2. For each pair duty-timeslot, the corresponding node contains: the number of personnel required, a list of personnel that can be selected (with the required ability and availability) and a heuristic selection strategy that is used to make the assignment of staff to that duty-timeslot. This arrangement allows flexibility to sort the list of personnel within each node according to different criteria to suit the particular assignment heuristic strategy.

Shift Pattern Design and Work Unit Allocation Procedure. Steps to design shift patterns that aim to satisfy work demand while also assigning staff to work units (i.e. pairs duty-timeslot). Some variables are defined first:

R_{dt} : minimum number of employees required to work duty d on timeslot t.
X_{dt} : number of employees assigned to work duty d on timeslot t. It is assumed that only employees with the required ability and availability are assigned.
R^c_{dt} : number of employees currently required to work duty d on timeslot t. This value is given by $R_{dt} - X_{dt}$ and it changes during the procedure.
E_{dt} : initial number of employees with ability and availability to work duty d on timeslot t, this value is fixed.
E^c_{dt} : current number of employees with ability and availability to work duty d on timeslot t. This value is given by $E_{dt} - X_{dt}$ and changes during the procedure.
A_{dt} : number of employees with the ability and availability to work duty d on timeslot t in excess of the number currently available E^c_{dt}. This value indicates the current upper limit in the number of employees with the required ability and availability.

Step 1. From those duty-timeslot(d,t) pairs with $R^c_{dt} > 0$, select the critical one based on some criterion, e.g. A_{dt} where the smaller the value the more critical the pair (d,t). Break ties based on greater R^c_{dt}, duty priority or timeslot priority. Continue processing (d,t) pairs with the target of $R^c_{dt} = 0$ for all but avoiding A_{dt} to become zero or less.

Step 2. Select employees for assignment to the (d,t) critical pair. Assign consecutive (d,t) pairs in line with existing constraints. The criteria used to select an employee for (d,t) can be: least filled-in timeslot (those in the gap between two critical ones), least overstaffing, priority of the employee, hours needed to meet employee request, etc. Continue until all R^c_{dt} is zero or below.

Steps 1-2 are iterated to perform shift pattern design and work unit allocation simultaneously.

Step 3. Improvement phase to repair constraints and improve objective values. 3.1 Tackle constraint violations using specific moves and rules taking intra-employee or inter-employee schedules into account. 3.2 Improve on objective values using specific moves and rules.

Step 4. Perform swaps between assignments employee-duty-timeslot (e,d,t) to improve the schedule, for example to reduce the changes of duties (restaurant scenario) or to reduce the payroll cost (leisure centre scenario).

Step 5. Given a change in an assignment employee-duty-timeslot (e,d,t), repair the roster using ejection chain moves while satisfying relevant constraints.

The above procedure helps us to design a roster for purpose by generating tailored shift patterns when required (fast-food restaurant scenario), assigning staff to work units (pairs duty-timeslot) and repairing the roster when necessary (leisure centre scenario). The effort that a human planner normally puts in producing a roster, is reduced considerably by this automated procedure which takes into account the changing work demand forecast and variable workforce availability. Equally important is the integration of the above workforce scheduling procedure with other processes such as payroll, attendance/absence management, staffing forecast and planning, etc. In this short communication we discussed the importance of integrating workforce scheduling with other HR&Payroll processes and provided an insight into our progress towards developing Midland HR's workforce management integral solution.

Acknowledgements We thank the financial support from the Technology Strategy Board (TSB) in the UK through the Knowledge Transfer Partnership scheme (project KTP 07074).

References

Belkin G (2009) Effective workforce scheduling helps retailers establish in-store service differentiation. URL http://www.aberdeen.com/launch/report/sector_insights/6457-SI-workforce-scheduling-retail.asp

Di Gaspero L, Gärtner J, Kortsarz G, Musliu N, Schaerf A, Slany W (2007) The minimum shift design problem. Annals of operations research 155:79–105

Ernst A, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: a review of applications, methods and models. European journal of operational research 153:3–27

Musliu N, Schaerf A, Slany W (2004) Local search for shift design. European journal of operational research 153(1):51–64

Tien JM, Kamiyama A (1982) On manpower scheduling algorithms. Siam Review 24(3):275–287

The Home Care Crew Scheduling Problem

Jesper Larsen · Anders Dohn · Matias Sevel Rasmussen · Tor Justesen

Abstract The Home Care Crew Scheduling Problem (HCCSP) of this paper has its origin in the Danish healthcare system. The home care service was introduced in 1958 and since then, there has been a constant increase in the number of services offered. The primary purpose is to give senior and disabled citizens the opportunity to stay in their own home for as long as possible. The problem of scheduling the services for the citizens as handled by the municipalities in Denmark can basically be decomposed into two interesting optimization problems. The first long-term planning problem fixes visit on days and assigns the visit a time window in accordance with the quality standarts of the municipality. The second short daily problem assigns theses visits for a given day to a given home carer therebye building the daily routes for a group of home carers. So in this problem, denoted the HCCSP, home carers should be assigned tasks in a way that maximizes the service level, possibly even at a reduced cost.

Keywords crew scheduling · vehicle routing · column generation · set partitioning

The Home Care Crew Scheduling Problem (HCCSP) of this paper has its origin in the Danish healthcare system. The home care service was introduced in 1958 and since then, there has been a constant increase in the number of services offered. The primary purpose is to give senior and disabled citizens the opportunity to stay in their own home for as long as possible. The HCCSP is the problem of scheduling home carers in a way that maximizes the service level, possibly even at a reduced cost.

Jesper Larsen
Department of Engineering Management, Technical University of Denmark, Denmark
E-mail: jesla@man.dtu.dk

Anders Dohn
Department of Management Engineering, Technical University of Denmark, Denmark
E-mail: adohn@man.dtu.dk

Matias Sevel Rasmussen
Department of Management Engineering, Technical University of Denmark, Denmark
E-mail: mase@man.dtu.dk

Tor Justesen
Copenhagen Airports, Denmark
E-mail: tjustesen@mac.com

The methodology presented in this paper is built on the literature of the vehicle routing problem with time windows (VRPTW). Column generation has proven an invaluable tool in optimization of vehicle routing problems. The main differences to regular vehicle routing problems are a limited number of home carers with individual shifts, temporal dependencies between visits, and the presence of a number of soft constraints. These exceptions must naturally be dealt with explicitly in the model.

When a citizen applies for home care service, a preadmission assessment is initiated. The result of the assessment is a list of granted services. The services may include cleaning and laundry assistance and support for other everyday tasks. They may also include assistance with respect to more personal needs, e.g. getting out of bed, bathing, dressing, preparing food, and dosing medicine. As a consequence of the variety of services offered, people with many different competences are employed as home carers.

Given a list of services for each of the implicated citizens, a long term plan is prepared. In the long term plan, each service is assigned to specific time windows, which are repeated as frequently as the preadmission assessment prescribes. The citizens are informed of the long term plan, and hence they know approximately when they can expect visits from home carers. From the long term plan, a specific schedule is created on a daily basis. In the daily problem, home carers are assigned to visits. A route is built for each home carer, respecting the competence requirements and time window of each visit and working hours of the home carer. In the following, we restrict ourselves to look at the daily scheduling problem only. The problem is a crew scheduling problem with strong ties to vehicle routing with time windows. However, we have a number of complicating issues that differentiates the problem from a traditional vehicle routing problem. One complication is the multi-criteria nature of the objective function. It is, naturally, important to minimize the overall operation cost. However, the operation cost is not very flexible in the daily scheduling problem. It is more important to maximize the level of service that we are able to provide. The service level depends on a number of different factors. Usually, it is very hard to fit all visits into the schedule in their designated time windows. Hence, some visits may have to be rescheduled or cancelled. In our solutions, a visit is either scheduled within the given restrictions or marked as uncovered. The manual planner will deal with uncovered visits appropriately. The main priority is to leave as few visits uncovered as possible. Also, all visits are associated with a priority and it is important to only reschedule and cancel less significant visits. Also, it is important to service each citizen from a small subgroup of the whole workforce, as this establishes confidence with the citizen. Another complication compared to traditional vehicle routing, is that we have shared visits. These are visits requiring the presence of more than one home carer, and consequently each visit must be included in the route of several home carers, where the interconnected visits must be synchronized.

HCCSP as described above is decomposed and modelled as a set partitioning problem (SPP) with side constraints. An elementary shortest path problem with time windows (ESPPTW) is used for column generation. The SPP is denoted the master problem, and the ESPPTW correspondingly is the subproblem. This approach has presented superior results on VRPTW and the similarities to HCCSP are strong enough to suggest the same approach here.

The model is solved in a Branch-and-Price framework. As the number of feasible routes is exponential in the number of visits, it is impossible to include all routes a priori. Instead, the most promising routes (columns) are generated dynamically in an iterative process. The master problem is LP-relaxed and the columns are generated based on a dual solution to the LP-relaxation.

The method is tested on authentic test instances from two Danish municipalities. The results are compared to the current practice, which is based partly on an automated heuristic and partly on manual planning. We measure three quality parameters: Uncovered visits, constraint adjustments, and total travel time. The uncovered visits are visits, where a home carer has not been assigned in the schedule. In practice, this may imply that the visit is cancelled or that a substitute is called in and assigned to those uncovered visits. Another way of dealing with an uncovered visit is to adjust the original constraints, so that we are able to fit the visit into the schedule anyway. Possible options are to: reduce the duration of the visit, extend the time window of the visit or extend the work shift of one of the home carers. This is done a lot in practice. However, any of these adjustments will naturally decrease the overall quality of the schedule. In the presented solution method, we have chosen to keep all the original constraints intact, and let the constraint adjustment be a manual post-processing task. This decision is also supported by the fact that it is hard to put a quantitative penalty on all possible adjustments before solving. The number of constraint adjustments in our solution will hence always be equal to 0.

The results clearly indicate that we are able to enhance the service level. There is a significant decrease in the number of uncovered visits and a truly dramatic decrease in the number of necessary constraint adjustments.

University course scheduling problem with traffic impact considerations

LEE Loo Hay · CHEW Ek Peng · NG Kien Ming · HUNG Hui-Chih · WANG Jia · XIAO Hui

Abstract Motivated by a practical problem, we consider a course scheduling problem for a university which is expanding its current campus in our work. In the new part of the campus, various facilities will be built, such as lecture theaters, seminar rooms, educational sport center, residential colleges, etc. The new part is designed to facilitate multi-disciplinary teaching and learning. Therefore, students from different faculties/schools are expected to have equal opportunities to enjoy these facilities. The two parts of campus are connected by a vehicular and pedestrian bridge. Shuttle service will be used to transport students between these two parts of the campus. Some courses may be reallocated to the new part.

Our problem is to assign courses to the new part of campus and to evaluate the responding load for students' having class on transportation system subject to the new course schedule. The goal is to find the best solution which satisfies the following main criteria: 1) Fairness, e.g. provide equal opportunities for students from different faculties/schools to enjoy these new facilities, 2) Priority, e.g. give preference to freshman and sophomore students in the usage of these new facilities, 3) Utilization of resources, e.g. balance the usage of new facilities at certain threshold level. The minimum time for as student to go for his next class is the 15 minute break time.

We first solve this problem by analyzing the historical data on student enrollment in different courses. From this analysis, we cluster courses according to their correlations. We assume a student has equal probability of moving to his next class during the interval from his last class and next. Based on the existing course schedule, we predict the student movement using worst 15 minute movements across campus parts. We build two models with the difference in measuring the movement. The first one considers usual movements, including both lecture and tutorial. The second one only considers lecture. It is because some tutorial has multiple sessions selectable for a student. Through our analysis on historical course schedules that were in the setting of one campus, some students tend to choose the tutorial session mostly close to their last class, which will worsen the traffic in multi-campus-part settings. Thus, the first model reveals the historical data more complete but the second model

LEE Loo Hay
Department of Industrial & Systems Engineering, National University of Singapore, 1 Engineering Drive 2, Singapore 117576, Singapore
Tel.: +65-65162895
Fax: +65-67771434
E-mail: iseleelh@nus.edu.sg

focus on the part less affected by the input variance. For both models, we build a mixed integer programming (MIP) model to decide the courses to be mounted at the new part of campus, so as to minimize the traffic impact.

Numerical experiments in current stage are based on past year data, including nearly 800 courses across the whole universities. Given latest capacity of new campus part in prospect, nearly 200 are selected to the new campus part. It is also revealed that the second MIP model have decreased the objective value from up to 300 into 70, which is believed controllable by the shuttle service. Through the numerical experiment, we have observed that faculty fairness has a great influence on the speed of obtaining the initial feasible solution, and thus it affects the solving time most. On the other hand, the traffic level is highly affected by the percentage of the freshmen and sophomore we set. This can be explained by our course relationship analysis, which reveals that higher-level courses are less correlated among each other compared with that of lower level courses. More freshmen and sophomore lead to lower level courses, therefore, it will eventually induce more traffic.

The future work includes developing and testing the automated and integrated systems ready to select courses in next two years (although only half of the resources are available during the first year) and evaluating the effectiveness of the model through simulation. At the same time, more efficient algorithm is in pursuit to handle larger problem scale potential from future situation.

Keywords University course scheduling · Traffic · Mathematical programming

Ground Crew Rostering with Work Patterns at a Major European Airline

R. M. Lusby · A. Dohn · T. M. Range · J. Larsen

Received: June 21, 2010

Abstract Staff Rostering is a well known optimization problem in the Operations Research literature. People are one of the most important resources, and the construction of efficient rosters using tailored optimization algorithms can lead to significant potential savings for the employer. In this paper we address one such problem arising in the ground operations at a major European airline. This problem, termed the Ground Crew Rostering Problem with Work Patterns (GCRPWP), entails assigning a set of employees to a set of shifts, which are spaced over a given daily time horizon, in such a way that the required employee demand on each shift is satisfied as closely as possible. Having too few staff on a given shift results in undercoverage, which is undesirable. When assigning a sequence of shifts to a particular employee one must respect several practical constraints. In particular, unlike traditional rostering problems, one must satisfy a so called work pattern of length l days. A work pattern specifies both the number of consecutive days of work (on-stretch) as well as the number of consecutive days of rest (off-stretch) an employee must have and is repeated over the rostering horizon.

We present a cutting stock based formulation and propose a column generation solution approach to find an efficient set of roster lines, where a roster line is a legal sequence of on and off-stretches that one or more employees can work. While the use of a repeating work pattern limits the number of feasible roster lines, the work pattern can be staggered across the employees to ensure all employees are not off on the same day. This staggering results in l independent subproblems, each of which entails solving a resource constrained shortest path in an appropriate acyclic network. To solve the model, we decompose the six month time horizon into smaller, computationally tractable blocks. The column generation procedure is combined with a variable fixing routine to find a roster for each block. The blocks are solved sequentially and consistency between the rosters of successive blocks is enforced through shift fixing in a prespecified overlapping duration between two consecutive blocks.

R. M. Lusby, A. Dohn, J. Larsen
Department of Management Engineering, Technical University of Denmark
Produktionstorvet, Building 426, 2800 Kgs. Lyngby, Denmark
E-mail: {rmlu,adohn,jesla}@man.dtu.dk

T. M. Range
Department of Business and Economics, University of Southern Denmark
Campusvej 55, 5230 Odense, Denmark
E-mail: tra@sam.sdu.dk

In addition, we describe an alternative time-based model that constructs roster lines simply using the forecast workload. Like the first model, this assumes the shifts have been predetermined; however, it does not assume that the required employee demand for each shift is known. The number of employees working any shift is determined by the optimization as part of the solution. By doing this we are able to circumvent one step of the conventional roster planning process. This second model is very similar in structure to that of the first and can be solved using the same methodology. With the second model, one attempts to cover, as well as possible, the workload rather than the required employee demand on each shift. We demonstrate that this second model is more flexible from a modelling perspective in that one can more easily include robustness factors. Robustness factors of interest for the airline in question include being able to cover a higher workload than anticipated as well as a workload that has been delayed by a prespecified number of minutes.

Encouraging numerical results are reported using real-life data supplied by a major European Airline. We also stress test the approach on 10 artificially constructed instances. All instances have a time horizon of 189 days and contain as many as 139 employees. The proposed methodology is shown to produce high quality rosters that outperform what is currently done in practice. In particular, from a robustness perspective, we show that more robust solutions can be obtained even with a 10-12% reduction in current staffing levels.

Keywords Staff Rostering · Decomposition · Robustness

References

Bruco MJ, Jacobs LW, Bongiorno RJ, Lyons DV, Tang B (1995) Improving personnel scheduling at airline stations. Operations Research 43(5):741 – 751

Dowling D, Krishnamoorthy M, Mackenzie H, Sier D (1997) Staff rostering at a large international airport. Annals of Operations Research 72:125 – 147

Properties of Yeditepe Examination Timetabling Benchmark Instances

Andrew J. Parkes · Ender Özcan

1 Introduction

Examination timetabling is a type of educational timetabling which is a highly challenging field for the researchers and practitioners. Examination timetabling problems require a search for the best assignment of examinations into a fixed number of time-slots possibly along with other resources, such as, a set of rooms with certain capacities, subject to a set of constraints. There are two common types of constraints: *hard* and *soft*. The hard constraints must not be violated, while the soft constraints represent preferences that can be infringed. Examination timetabling problems are proven to be NP-complete (Even et al 1976). A recent survey on exam examination timetabling can be found in Qu et al (2009).

There are many variants of examination timetabling problems due to the fact that each educational institution have their own rules, regulations and expectations resulting with various constraints. This situation also makes it extremely difficult to compare different solution methods. Not only comparability but also reproducibility of the results is vital for the research community, as pointed out in Schaerf and Gaspero (2006). McCollum (2006) discusses real world issues in examination and course timetabling. Although practitioners and researchers have to deal with different aspects of examination timetabling, it has been always of interest for both communities to design robust and flexible approaches that can solve new problem instances. ITC2007 (http://www.cs.qub.ac.uk/itc2007/) competition is organised considering the real world examination timetabling complexities and capturing them within the problem instances. The state of the art method for examination timetabling turned out to be a hybrid multistage approach combining Iterated Forward Search (IFS) for feasible initial solution construction and great deluge for improvement as described in Müller (2009). The source code of the solver is available from http://www.unitime.org/itc2007.

Yeditepe University (Faculty of Engineering) data set contains real problem instances from a total of eight semesters in three consecutive years. Bilgin et al (2007) modified the initial data set provided in Özcan and Ersoy (2005) with new properties and also generated a variant of Toronto benchmarks (Carter et al 1996) that fits into the problem formulation which will be referred to as modified Toronto benchmark. This problem is a capacitated variant of examination timetabling. There is a maximum *capacity* of seating available during

School of Computer Science, University of Nottingham, NG8 1JX, UK
http://www.cs.nott.ac.uk/∼{ajp,exo}/

Table 1 Characteristics of the modified Toronto benchmark dataset.

Instance	No. of Exams	No. of Students	No. of Enrolments	Conflict Density	Days	Capacity
car91 I	682	16925	56877	0.13	17	1550
car92 I	543	18419	55522	0.14	12	2000
ear83 I	190	1125	8109	0.27	8	350
hecs92 I	81	2823	10632	0.42	6	650
kfu93	461	5349	25118	0.06	7	1955
lse91	381	2726	10918	0.06	6	635
pur93 I	2419	30029	120681	0.03	10	5000
rye92	486	11483	45051	0.07	8	2055
sta83 I	139	611	5751	0.14	4	3024
tre92	261	4360	14901	0.18	10	655
uta92 I	622	21266	58979	0.13	12	2800
ute92	184	2749	11793	0.08	3	1240
yor83 I	181	941	6034	0.29	7	300

Table 2 Characteristics of the Yeditepe benchmark dataset.

Instance	No. of Exams	No. of Students	No. of Enrolments	Conflict Density	Days	Capacity
yue20011	126	559	3486	0.18	6	450
yue20012	141	591	3708	0.18	6	450
yue20013	26	234	447	0.25	2	150
yue20021	162	826	5755	0.18	7	550
yue20022	182	869	5687	0.17	7	550
yue20023	38	420	790	0.2	2	150
yue20031	174	1125	6714	0.15	6	550
yue20032	210	1185	6833	0.14	6	550

exams at each time slot. The timetable size is fixed with three examination slots per day for a given number of days. The characteristics of each problem instance of modified Toronto and Yeditepe benchmark problem instances are summarised in Table 1 and 2, respectively.

Yeditepe examination timetabling problem has the usual hard constraints:

- Examination conflict (C_1): A student must not sit for more than one examination at any given time.
- Capacity (C_2): At a given period, the overall number of students seated for all examinations should not exceed the fixed capacity.

and the soft constraint

- Examination spread (C_3): Examinations of a student in the same day should not be scheduled consecutively.

As yet, to our knowledge, optimality has not been proven for any solutions of the examination timetabling problem instances in the Toronto and ITC2007 benchmarks, even including the smallest problem instances. This study focuses on the smallest Yeditepe instances which can be solved exactly, and so allows us to test and compare the optimal solutions and the state of the art approach of Müller. Additionally, a multi-objective formulation of the problem based on the trade-off between the room size (capacity) and solution quality is analysed.

Table 3 The results for yue20023.

RoomCap.	penalty	time(secs)	IFS-GD
132	70	1123	86, 94, 86
134	68	1468	100, 115, 105
135	65	935	72, 87, 87
136	64	1022	81, 74, 83
137	59	818	73, 73, 87
146	56	875	80, 65, 67
153	55	304	77, 76, 67
157	54	402	73, 73, 67
166	50	371	58, 65, 67
170	48	295	72, 72, 64
176	47	268	64, 76, 66
187	46	234	48, 63, 63

2 Some Exact Results

In this section, we report results of completely solving one of the smaller instances as a case study; yue20023 (chosen simply because it was the largest that we could solve exactly). It was solved using an encoding[1] of the exam timetabling problem as described in (McCollum et al 2008) into ILOG/IBM OPL and solved using CPLEX 11. The encoding is not optimised (e.g. there is no branch and cut) and so timing results are purely for comparison of the relative hardness of different cases. Also, in order to give a better insight into the problem, the size of the room used was varied, and the effect on the final penalty studied. The results given in Table 3 are all illustrated in Figure 1. The last column simply gives the quality obtained from three 3 separate runs of 1200 seconds each and using Müller's winning submission to the examination timetabling track of ITC2007.

There are two main observations:

Firstly, the Pareto front is not trivial, there are fairly wide ranges of unsupported solutions - that is, solutions that are Pareto optimal but not on the convex hull of the Pareto front, and so are not optimal with respect to any linear combination of the objectives. In this case, for example, the Pareto optimal solutions with room size of 153 or 157 will be missed if solving optimally using a linear combinations of the room size and penalty; as any linear combination will not be able to access the 'indented portion' of the Pareto front.

Secondly, even though the hybrid approach tested was the clear winner of ITC2007, it still did not manage to find optimal solutions. This suggests that even on these small instances there is still significant room for improvement in the performance of meta-heuristics.

3 Summary

We have made available some exam timetabling instances from Yeditepe. Despite their independent origin, they fit reasonably well into the format of the ITC2007 benchmarks, suggesting that this format captures real-world issues (McCollum et al 2008). On the smaller instances, we were able to solve them completely using integer programming. In terms of a multi-objective trade-off between the room size and solution quality, the Pareto fronts were found to be interesting with large unsupported regions. This suggests that weighted sum

[1] The website http://www.cs.nott.ac.uk/∼ajp/timetabling/exam/ gives the encoding, the instances, and other supplementary material.

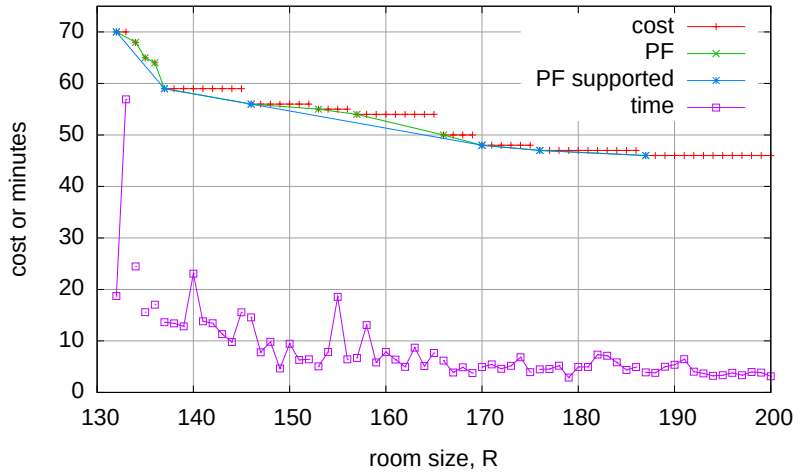


Fig. 1 Results for the instance 'yue20023' as given as a function of the room size. 'Cost' is the optimal (minimal) penalty. The 'PF' are those solutions that are non-dominated. The 'supported PF' are those on the convex hull of the PF line. The 'time' is minutes for CPLEX to solve the instance, which includes the proof of optimality.

methods would potentially miss many interesting solutions. It was also interesting that even the best meta-heuristic solver was consistently failing to find the optimal, suggesting that this research area still has room for considerable improvement.

References

Bilgin B, Özcan E, Korkmaz EE (2007) An experimental study on hyper-heuristics and final exam scheduling. In: Proc. of the International Conference on the Practice and Theory of Automated Timetabling (PATAT'06), Lecture Notes in Computer Science, vol 3867, pp 394–412

Carter MW, Laporte G, Lee S (1996) Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society 47(3):373–383

Even S, Itai A, Shamir A (1976) On the complexity of timetable and multicommodity flow problems. SIAM Journal on Computing 5(4):691–703

McCollum B (2006) University timetabling: Bridging the gap between research and practice. In: Proc. of the 5th International Conference on the Practice and Theory of Automated Timetabling, Springer, pp 15–35

McCollum B, McMullan P, Burke EK, Parkes AJ, Qu R (2008) A new model for automated examination timetabling (under review)

Müller T (2009) ITC2007 solver description: a hybrid approach. Annals of Operations Research 172:429–446

Özcan E, Ersoy E (2005) Final exam scheduler - FES. In: Proc. of the Congress on Evolutionary Computation, IEEE, pp 1356–1363

Qu R, Burke EK, McCollum B, Merlot L, Lee S (2009) A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12(1):55–89

Schaerf A, Gaspero LD (2006) Measurability and reproducibility in timetabling research: State-of-the-art and discussion. In: Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling, pp 53–62

Combined Blackbox and AlgebRaic Architecture (CBRA)

Andrew J. Parkes

1 Context

Combinatorial optimisation methods are concerned with an objective function defined over an (exponentially) large search space. Often these are solved with metaheuristics, such as simulated annealing, various evolutionary strategies, tabu search, and many others. However, such metaheuristics are generally rather static creatures with relatively simple behaviours and with most intelligence delegated to the heuristics, the neighbourhoods, selection/acceptance criteria and/or the local search methods that they oversee. This static simpleness has often paid off in terms of simplicity of implementation: Often the metaheuristic is merely tens to hundreds of lines of code, in comparison to maybe tens of thousands of lines to implement complex local search moves (with the associated data-structures needed for them to run quickly). However, the metaheuristics do have some decisions to make, and simple static decisions tend to lead to an over-reliance on careful tuning of parameters. There is a long-standing, but ever-growing, desire that metaheuristics become much more dynamic, self-adaptive, and with inbuilt learning mechanisms, in short, that they become more intelligent. There have been various attempts to add this intelligence. This short abstract cannot hope to do justice to them all, however, some pointers into the recent literature are: Hyper-heuristics (Burke et al 2003; Chakhlevitch and Cowling 2008; Burke et al 2009; Ross 2005; Özcan et al 2008), reactive search Battiti et al (2008), and self-adaptive genetic algorithms. This position paper is an attempt to give one view of how some different approaches are related. (Of course, this is just one view and is not intended to supplant other views.) The starting point is that optimisation methods can often be classified at a very high level according to whether they are blackbox or whitebox:

Blackbox optimisation A blackbox algorithm typically refers to an algorithm that has no insight into the structure of the objective function. Part of the desired intelligence is surely that there is some learning of the structure of the problem. However, in the blackbox case, only inductive learning will be possible. Unfortunately, as known from the ugly duckling theorem (Watanabe 1969) and later the No-Free-Lunch-Theorem (NFLT) (Wolpert. and Macready 1997), when nothing a priori is known about the structure then induction (learning) can be no better than direct enumeration of the search space.

Whitebox optimisation The full structure of the constraints and the objective function is exposed to the solver, and explicitly exploited by it. The archetypal example would be integer programming; the solver would be looking at the internal structure and trying to do deductive learning about new structures, for example, by deriving new cutting planes.

School of Computer Science, University of Nottingham, NG8 1JX, UK
http://www.cs.nott.ac.uk/∼ajp/

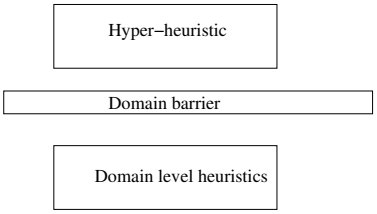


Fig. 1 Hyper-heuristic view of blackbox optimisation. Note that all of the domain is hidden, including all variables.

Such methods are of course not mutually exclusive. The area of constrained blackbox optimisation (cBBO) (e.g. see Wu et al (2006)), roughly speaking, is concerned with solving problems expressed as:

$$\min \quad f^B(x) \tag{1}$$
$$s.t.\ g^W(0) = 0 \tag{2}$$

where the objective function f^B is blackbox, though the constraints g^W are whitebox. In recent work related to timetabling, one can perhaps regard one instantiation of this cBBO framework as the "RAMP" formulation of Bykov[1] in which variables are a set of finite-domain integers, subject to whitebox knapsack and inequality constraints determining feasibility, but also with a blackbox objective function.

The standard selection hyper-heuristic framework is illustrated by the domain barrier of Figure 1 in which the hyper-heuristics are a high-level control system that selects which of various low-level domain-specific heuristics to use, and receives knowledge only of the resulting effect on the objective function. (The term 'hyper-heuristics' also covers other methods, e.g. see Burke et al (2009), but the blackbox selection of heuristics is the most common meaning.) Note that in the classical hyper-heuristic approach the domain is hidden to the extent that the high-level hyper-heuristic controller does not even know of the variables themselves. In contrast, many blackbox methods at least are given a representation of the variables. In genetic algorithm approaches the variables are collected into a chromosome and the blackbox is the objective function.

Hence, overall, there is no unique mix of blackbox and whitebox, which suggests that we should consider more general ways of mixing black and white reasoning. For this, there are 3 general components: the variables, the constraints and the objective(s), and so we discuss each in turn.

Visibility of the variables. The main observation is that work in this area rarely explicitly discusses the black/white nature of the variables or the search space itself. Of course, in any given approach, it is always clear, however, the choices made are often not discussed or modelled explicitly. However, there are a range of possibilities. Here we are driven by some earlier work on the issues of coarse-grained distributed solvers (Parkes 2001) in which the problem is divided into large chunks to be solved on separate solvers. As a means of communication, each chunk is given some partial visibility of the variables of other chunks. Variables within a chunk are hence split into those that are public, x, or private, y. This can also make sense in that encoding many problems results in multiple kinds roles for variables. For example, some are key decision variables, and others are 'mere' auxiliary variables needed to encode some complex expression in the objective.

The primary observation of this paper is that such a public/private or white/black split in the variables should also be explicitly considered as a possibility within general frameworks. In such cases, it seems reasonable that only the public variables, x, are treated as whitebox, and others, y, are hidden inside a blackbox.

The public variables, x, will often simply be a subset of the total set of variables, but in general could be derived variables. So, if there are some underlying total set z of variables then we might express this as:

$$x = p(z)$$

where p is a projection operator that picks out which ones are to be made public. The standard hyper-heuristic corresponds to $p(z) = \{\}$ so that no variables are public. Conversely, genetic algorithms generally

[1] URL http://www.cs.nott.ac.uk/∼yxb/actispec/

take that $p(z) = z$ so that the chromosome is a complete representation. Explicitly introducing such a projection operator could allow an organised study of intermediates between these extremes.

Visibility of the objective(s). There does not seem to be any real, a priori, reason in the cBBO framework, that the entire objective function needs to be completely blackbox, but it could well be a combination of black and white. In this case, by definition, the whitebox portion will only be able to use the public variables, and the objective becomes

$$f^W(x) + f^B(x,y)$$

Visibility of the constraints Generally, there can be whitebox constraints over the public variables, but the blackbox side is also likely to have to account for constraints over all the variables. Note that it could well be that even whitebox constraints are converted to large penalties are then treated in a blackbox fashion (many approaches to cBBO will do this). It is also common that they are treated implicitly within the blackbox portions as hard constraints and so are used to limit the moves in the search space. In the hyper-heuristic approach, it is quite common that the "low-level heuristics" are various neighbourhood moves, but the neighbourhoods are selected so as to preserve feasibility. (For example, in timetabling, the neighbourhood moves might swap times of events only if no new conflicts are created).

Putting all this together the proposed structure is (with $z = (x,y)$):

$$min\ f^W(x) + f^B(z) \tag{3}$$
$$s.t.\quad g^W(x) = 0 \tag{4}$$
$$g^B(z) = 0 \tag{5}$$
$$\text{implicitly with}\quad x = p(z) \tag{6}$$

Whitebox constraints are often written in algebraic form, so I will refer to as a "Combined Blackbox and algebRaic Architecture" (CBRA) formulation.

However, what would be the point of such a formulation? - Why not just go all black or all white? The main potential advantage is that real-world problems are often a mix of components that most naturally lean towards one or the other. For example, in timetabling the overall time and room choices are a mix of graph colouring and assignment problems and so naturally suggest whitebox methods. On the other hand the pattern constraints (no two events in a row, no more than three in a day, etc) tend to be relatively hard to encode in whitebox style and instead are currently better handled by a good local search method hidden in a blackbox. Note that it is effectively necessary to hide the local search in a blackbox because trying to represent it integer programming or similar is generally awkward and ineffective. In particular, the ability of the framework to exploit only public variables in a whitebox fashion could be useful. (For example, the surface and deep formulations for Course Timetabling in Burke et al (2010) might well have a natural expression in this form.)

Implementing this 'architecture' is future work, but can be mostly expected to be a judicious combination of techniques from cBBO, evolutionary algorithms, hyper-heuristic, and others. The main novelty within this is the public/private variable split, so a key issue is whether this can be handled effectively. Note that the CBRA might well want to fix some or all of the public variables, and so this will need to be passed down to the blackbox objective. In standard hyper-heuristics all variables are blackbox and so this would be handled by an operator requesting the blackbox domain-specific component to create an initial state, but without the top-level getting to know anything about it other than its objective value.

The practical value of this proposal is to help move towards a system giving a good way to handle simultaneously both the extremes of

- whitebox 'algebraic' reasoning about 'simple' constraints over simple public variable
- blackbox handling of complex local moves possibly designed to fix some objectives represented by awkward private variables, and that cannot sensibly be handled with any generic whitebox solver

and also to be able to handle a range of intermediates. Note that this discussion has not really relied at all on any particular choice of algorithms, but is attempting to formalise a representational scheme, and associated overall architecture.

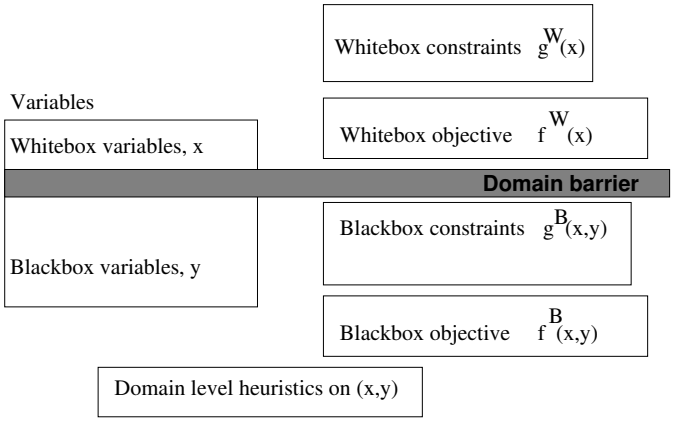


Fig. 2 Sketch of CBRA structure.

2 Conclusions

A framework is proposed that includes many existing variants of solvers from fully blackbox to fully whitebox, and most specifically proposes that black/white and mixtures considerations be explicitly applied to variables as well as to objective functions and constraints. In a sense, this paper can be taken as the suggestion that theoretical developments in this direction might well take as much account of the partial hiding of variables as much as they do of the (partial) hiding of the details of objectives.

Two final comments:

Changing black to white? In some cases, in principle, it might well be possible to convert blackbox components to white again by combining promises about the structure of the objective with analysis of results of calls to obtain its value. For example, if the whitebox reasoner where promised that the blackbox just encodes a TSP, then maybe with sufficient acumen it could actually deduce the edge lengths that were hidden and so would then be able to reason directly. This is not so likely to be a practical issue, but could well place limitations on what can be formally proven about such systems: the extra promise that is is an encoding of some particular domain might sometimes lead to breakage of the domain barrier.

Adding "solution features"? In a machine learning hyper-heuristic framework we might well want some features of the solution other than just its objective. For the whitebox components CBRA can use whatever features it decides are useful, but it might well need to also use features of the hidden variables. Presumably these can simply be returned along with the objective function.

References

Battiti R, Brunato M, Mascia F (2008) Reactive Search and Intelligent Optimization, Operations Research/Computer Science Interfaces Series, vol 45. Springer

Burke EK, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003) Handbook of Meta-Heuristics, Kluwer, chap Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pp 457–474. URL http://www.asap.cs.nott.ac.uk/publications/pdf/hhchapv002.pdf

Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E, Woodward J (2009) A classification of hyper-heuristic approaches. Tech. Rep. NOTTCS-TR-SUB-0907061259-5808, University of Nottingham, School of Computer Science.

Burke EK, Mareček J, Parkes AJ, Rudová H (2010) Decomposition, reformulation, and diving in university course timetabling. Comput Oper Res 37(1):582–597, DOI http://dx.doi.org/10.1016/j.cor.2009.02.023

Chakhlevitch K, Cowling P (2008) Hyperheuristics: Recent developments. Studies in Computational Intelligence 136:3–29, in Cotta C, Sevaux M, Sorensen K (eds) Adaptive and Multilevel Metaheuristics

Özcan E, Bilgin B, Korkmaz EE (2008) A comprehensive survey of hyperheuristics. Intelligent Data Analysis 12(1):3–23

Parkes AJ (2001) Exploiting solution clusters for coarse-grained distributed search. In: IJCAI01 Workshop on Distributed Constraint Reasoning, Seattle, Washington, USA

Ross P (2005) Hyper-heuristics. In: Burke EK, Kendall G (eds) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, chap 17, pp 529–556

Watanabe S (1969) Knowing and Guessing: A Quantitative Study of Inference and Information. New York: Wiley

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1:67

Wu Y, Ozdamar L, Kumar A (2006) Nonconvex Optimization and Its Applications, vol 85, chap Parallel Triangulated Partitioning for Black Box Optimization, pp 487 – 506

Solving the Airline Crew Pairing Problem using Subsequence Generation

Matias Sevel Rasmussen · David M. Ryan · Richard M. Lusby · Jesper Larsen

Abstract Good and fast solutions to the airline crew pairing problem are highly interesting for the airline industry, as crew costs are the biggest expenditure after fuel for an airline. The crew pairing problem is typically modelled as a set partitioning problem and solved by column generation. However, the extremely large number of possible columns naturally has an impact on the solution time.

In this work in progress we severely limit the number of allowed subsequent flights, i.e. the subsequences, thereby significantly decreasing the number of possible columns. Set partitioning problems with limited subsequence counts are known to be easier to solve, resulting in a decrease in solution time.

The problem though, is that a small number of deep subsequences might be needed for an optimal or near-optimal solution and these might not have been included by the subsequence limitation. Therefore, we try to identify or generate such subsequences that potentially can improve the solution value.

Keywords Airline crew pairing · Crew pairing · Subsequence generation · Column generation · Limited subsequence

M.S. Rasmussen
Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, 2800 Kgs. Lyngby, Denmark
Tel.: +45-45254442
Fax: +45-45933435
E-mail: mase@man.dtu.dk

D.M. Ryan
Department of Engineering Science, University of Auckland, 70 Symonds Street, Auckland 1010, New Zealand

R.M. Lusby
Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, 2800 Kgs. Lyngby, Denmark

J. Larsen
Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 424, 2800 Kgs. Lyngby, Denmark

1 Introduction

Crew costs are the second largest expenditure in the airline industry. Only fuel costs are larger, see [1]. Therefore, airline crew scheduling has received a lot of attention in the literature, and consequently, optimisation is heavily used by the airlines. The airline crew pairing problem which is dealt with in this work is a part of a larger series of optimisation problems that together produce the schedule for an individual crew member. In [1] a recent survey of airline crew scheduling can be found.

A *pairing* or a *tour-of-duty* is a sequence of flights which can be flown be a crew member. A pairing must start and end at the same crew base and comply with several rules and regulations in order to be feasible. The *airline crew pairing problem* then finds the set of pairings that exactly covers all flights at minimum costs.

2 Solution Method

The pairing problem is modelled as a set partitioning problem. Each row corresponds to a flight and each column corresponds to a pairing. Let m be the number of rows and n be the number of columns, and let c_j be the costs of column $j \in \{1, \dots, n\}$. The entries of $\mathbf{A}$, a_{ij}, are one if column $j \in \{1, \dots, n\}$ covers row $i \in \{1, \dots, m\}$ and zero otherwise. The decision variables x_j for $j \in \{1, \dots, n\}$ are binary. The mathematical programme can be written as

$$\begin{aligned} &\text{minimise} && \mathbf{c}^\top \mathbf{x} \\ &\text{subject to} && \mathbf{A}\mathbf{x} = \mathbf{1} \\ & && \mathbf{x} \in \{0,1\}^n \ . \end{aligned}$$

The number of possible pairings in the set partitioning formulation is very large, so the pairings are typically only enumerated implicitly by column generation. In the present approach we will, however, not perform column generation, but *subsequence generation*.

The *subsequences* for a flight f are the set of subsequent flights that can follow f in a feasible way in a pairing. In general terms for a zero-one matrix $\mathbf{A}$, the *subsequence count*, $\mathrm{SC}(s)$, for any row s is given by

$$\mathrm{SC}(s) = |\{t : [a_{sj} = 1, a_{ij} = 0 \text{ for } s < i < t, a_{tj} = 1], j = 1, \dots, n\}| \ .$$

Matrices with $\mathrm{SC}(s) \leq 1$ for all $s \in 1, \dots, m$ are said to have *unique subsequence*, and such matrices are balanced, see [2]. Exploiting results from graph theory, we know that the LP relaxation of an SPP with a balanced $\mathbf{A}$ matrix has an integral optimal solution. Also shown in [2], the closer we get towards unique subsequence, the closer we get to naturally integral LP solutions.

Therefore, we severely limit the subsequence count for each flight when generating pairings. This results in significantly fewer possible pairings and, as mentioned, fewer fractions when solving the LP relaxation. The disadvantage, however, is that we might exclude some optimal subsequences. To remedy this, we use the information in the dual vector to identify missing subsequences. The dual vector is passed on to one or several column generators that produce negative reduced costs columns on a richer set of subsequences. These columns are analysed in order to identify potentially "good" subsequences.

The goal is, of course, to be able to, as early as possible, identify the subsequences that will end up in the optimal or near-optimal solution. Whenever a subsequence is identified as a potentially "good" subsequence, the whole set of columns which include the new subsequence are added to the LP. Furthermore, to prevent the LP from growing too big, subsequences can be removed from the LP, i.e. the set of columns containing the subsequence are removed.

3 Computational Results

In order to gain better understanding of the method, we have generated a set of set partitioning instances with a cost structure reflecting the cost structure from crew pairing problems. The results from the generated instances indicate that we can identify the missing subsequences in reasonable time.

Currently, we are in the process of performing tests on a set of real-world crew pairing problem instances.

4 Future Work

The results this far clearly justify further development. Firstly, as mentioned, real-world crew pairing problems will be tackled. Secondly, the subsequence identification process has room for improvements. Thirdly, the method is based on the dual vector, therefore dual stabilisation is likely to speed up the method, as dual stabilisation would make the duals more reliable. Lastly, the column generators can be run in parallel on different processors.

References

1. Gopalakrishnan, B., Johnson, E.L. (2005). Airline crew scheduling: state-of-the-art. *Annals of Operations Research*, 140, 305–337
2. Ryan, D.M., Falkner, J.C. (1988). On the integer properties of scheduling set partitioning models. *European Journal of Operational Research*, 35, 442–456

Grouping Genetic Algorithm with Efficient Data Structures for the University Course Timetabling Problem

Felipe Arenales Santos · Alexandre C. B. Delbem

Keywords Grouping Genetic Algorithm · Timetabling Problem

1 Introduction

An efficient grouping genetic algorithm (GGA) was proposed by R. Lewis and B. Paechter (2007) to find feasible timetables for a version of the university course timetabling problem (UCTP). The algorithm proved to be efficient to construct feasible solutions for the UCTP benchmark instances (International Timetabling Competition 2002). In addition, the authors proposed a new set of harder instances, in order to better analyze their algorithm.

In this paper, we propose an adequate data structure for the algorithm and pre-processing techniques that signficantly improve its efficiency. The organization of the paper is as follows. Section 2 defines the UCTP and presents the main characteristics of the GGA. Section 3 and 4 describes our contributions for the GGA applied to UCTP. Finally, Section 5 presents computational experiments and the conclusions.

2 The University Course Timetabling Problem

A typical university timetabling problem consists of assigning a set $\mathbf{u}=\{u_1, ..., u_e\}$ of e events (classes, exams, lectures, etc.) to t timeslots and r rooms in such a way as to satisfy a set of constraints and to optimize an objective function (R. Lewis and B.

Felipe Arenales Santos
University of Sao Paulo
Sao Carlos, SP
Brazil
E-mail: fearenales@grad.icmc.usp.br

Alexandre Claudio Botazzo Delbem
University of Sao Paulo
Sao Carlos, SP
Brazil
E-mail: acbd@icmc.usp.br

Paechter 2007). The constraints are classified into two types (Burke et al. 1997): (i) hard constraints, which must be satisfied to a feasible timetable, and; (ii) soft constraints, which should be satisfied if possible. The hard constraints considered to this problem are:

1. Events requiring the same student should not be assigned to the same timeslot;
2. One room in one timeslot admits only one event;
3. All of the features required by an event should be satisfied by the room where the event is allocated, which has adequate capacity.

The problem is NP-Complete (Garey and Johnson 1979), and the number of possible assignments grows exponentially with the input data. As a consequence, exact methods for UCTP are not proper for large instances. Therefore, heuristic-based methods have been widely investigated in the literature (Costa 1994; Abramson, Krishnamoorthy and Dang 1996; Thompson and Dowsland 1998; Schaerf 1999; Paechter et al. 1998; Socha and Samples 2003).

Lewis and Paechter (2007) describe the applicability of a GGA to the UCTP, and propose a two-phase approach. In the first phase, the GGA improves the quality of the solutions (within a specified time) using a set of solution-builder heuristics, which speeds up the evolutionary process. In the second one, a local search algorithm does final improvements to promising solutions. However, as the input data increases too much (big cases of the proposed set), the convergence becomes slower and the first phase ends with high infeasibility level solutions. It should be worth noting that this paper deals with only the first phase and hard constraints.

3 Data Structures

Some of the data structs used by the GGA are trivial, such as the set $\mathbf{u}$, implemented as an array, and the timetable, represented by the $\mathrm{r_x t}$ matrix $\mathbf{T}$, where the element t_{ij} is the event assigned to room i in timeslot j. Other matrices give the relation between students and events and between features and rooms. For further information, see Lewis and Paechter (2007).

The most complex data structure envolved in this GGA, called feasibility matrix $\mathbf{F}$, stores the places (one place is one room in one timeslot) where the events can be feasibly allocated. Its most efficient implementation (by the processing aspect) consists of a matrix whose rows correspond to places, and columns to events. It can be done by initially converting a two-dimensional matrix ($\mathbf{G}$, rooms by timeslots), that contains the feasibility information of a given event, to a one-dimensional array of places $\mathbf{H}$. Figure 1 depicts this step.

$$\mathbf{G} = \begin{pmatrix} g_{11} & \cdots & g_{1t} \\ \vdots & \ddots & \vdots \\ g_{r1} & \cdots & g_{rt} \end{pmatrix} \rightarrow \mathbf{H} = \begin{pmatrix} g_{11} & \cdots & g_{1t} & \cdots & g_{r1} & \cdots & g_{rt} \end{pmatrix} = \begin{pmatrix} h_1 & \cdots & h_p \end{pmatrix}$$

Fig. 1: Linearization of the feasibility matrix to an event

Next, the resulting structure **H** is replicated e times (one for each event), resulting in the structure **F**. The **F** element $f_{ij} = 1$ if event j can be feasibly assigned to a place i, and 0 otherwise, i = 1,...,p and j = 1,...,e, see Figure 2.

$$\mathbf{F} = \begin{pmatrix} H_1^T & \cdots & H_e^T \end{pmatrix} = \begin{pmatrix} f_{11} & \cdots & f_{1e} \\ \vdots & \ddots & \vdots \\ f_{p1} & \cdots & f_{pe} \end{pmatrix}$$

Fig. 2: Feasibility matrix construction

In order to find out if one given event can be feasibly assigned to a place, no search is required, since the triple (*event*, *room*, *timeslot*) can be transformed into the pair (*event*, *place*) that corresponds to only one element of **F**. In this way, the cost to access this information is (in big-O notation) $O(1)$ (Cormen et al. 2001; Knuth 1998). It speeds up the generation of solutions, i.e. the GGA can performe a larger number of generations for the same period of time compared to other relatively higher cost data structures, such as dynamic lists and other non-static data structures. Moreover, it is important to highlight that the larger the number of generations, the better the quality of solutions at the end of the GGA.

4 Preprocessing Techniques

The values of the **F** elements are only related with the hard contraints satisfaction (see Section 2). Based on this, the hard constraints can be classified into two types: (i) static contraints, which define the assignment feasibility regardless the timetable state (i.e. the events allocated in it), as constraint 3; and (ii) dynamic constraints, which can change the feasibility of an assignment depending on the events already allocated in a timetable place, as constraints 1 and 2. Thus, the feasibility related to constraint 3 can be determined *a priori*. If an event requires a set of features and is required by n students, all the rooms that do not have all the required features or has the capacity less than n can not be feasibly assigned to this event anytime. This information is stored in matrix **A** (event by rooms) that lists the event-rooms conflicts. Element a_{ij} is 1 if the event i can not be assigned to the room j, i=1...e, j = 1,...,r; $a_{ij} = 0$ otherwise. Thus, the values of **F** to a new solution can be obtained directly from **A**, avoiding unneeded processing.

In addiction, the restriction number 1 can be modeled as a matrix **B** (event by event) that lists the event conflicts. The element $b_{ij} = 1$ if the event i requires a student also required by the event j, i = 1,...,e, j = 1,...,e, $i \neq j$; $b_{ij} = 0$ otherwise. Observe that every iteration of a solution building inserts one unallocated event into the solution being built.

5 Computational Results

The GGA and the proposed improvements was implemented in ANSI C under Linux (Ubuntu 8.04 distribution) on a computer with an Intel Pentium Dual-Core 1.73 GHz processor and 1 GB RAM. Considering the parameters as follow: timeslots number $t = 45$ (five days a week of five timeslots), mutation rate $m_r = 2$, inversion rate $i_r = 4$, recombination rate $rr = 1.0$, and population size $\rho = 50$. The input data is given by the instance sets. The experimental results was compared to the obtained by Lewis and Paechter (2007). The Figure 3 depicts the results obtained by the original implementation of the GGA and the implemented with improvements. The vertical axis corresponds to the distance to feasibility of the best solution present in the population. See Lewis and Paechter (2007) for further details about parameters and GGA implementation and run.

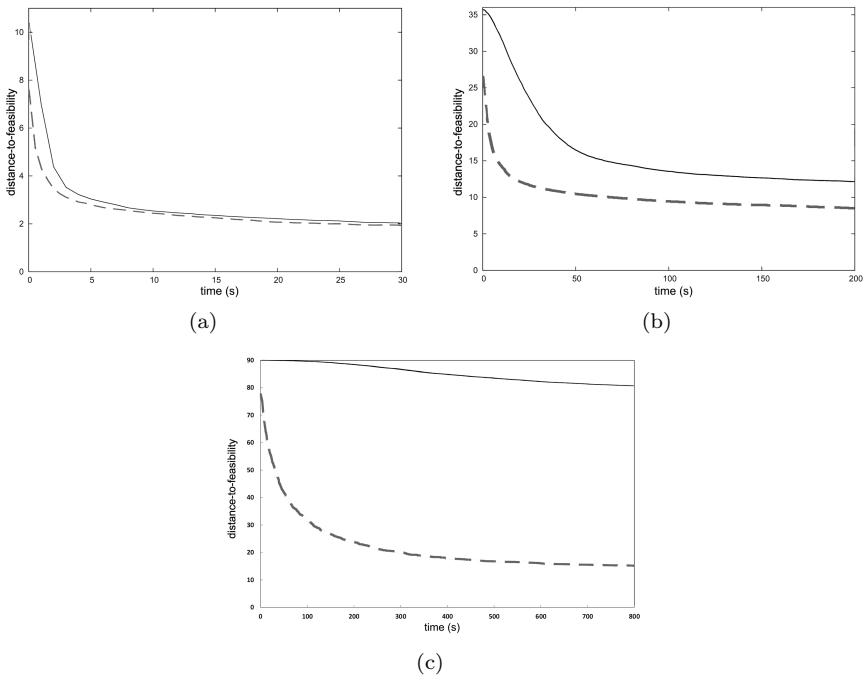


Fig. 3: Behavior of the algorithm to (a) small, (b) medium, and (c) large instances.

As can be noticed, the larger the input data, the greater the difference between the results obtained by the two implementations. It happens because as the input grows, the solution building time becomes significantly greater and the time saved is more expressive, showing the efficiency of the contributions proposed.

References

1. Abramson, D.; Krishnamoorthy, H.; Dang, H., Simulated Annealing Cooling Schedules for the School Timetabling Problem, *Asia-Pacific Journal of Operational Research*, vol. 16, pp. 1-22 (1996).
2. Burke, E. K.; Kingston, J.; Jackson, K.; et al., Automated university timetabling: the state of the art, *The Computer Journal*, vol. 40(9), pp. 565-571 (1997).
3. Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C., *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill (2001).
4. Costa, D., A tabu search algorithm for computing an operational timetable, *European Journal of Operational Research*, vol. 76, pp. 98-110 (1994).
5. Garey, M. R.; Johnson, D. S., *Computers and intractability: a guide to the theory of NP-completeness*. Freeman and Company, New York (1979).
6. Knuth, D., *The Art of Computer Programming*, Volume 1: *Fundamental Algorithms*, Third Edition. Addison-Wesley, pp. 107-123 (1997).
7. Lewis, R. and Paechter, B., Finding Feasible Timetables Using Group-Based Operators, *IEEE Transactions on Evolutionary Computation*, vol. 11(3), pp. 397-413 (2007).
8. Paechter, B.; Rankin, R.; Cumming A.; Fogarty, T., Timetabling the Classes of an Entire University with an Evolutionary Algorithm, in *Parallel Problem Solving from Nature* (PPSN) V (Lecture Notes in Computer Science, vol. 1498), Baeck T.; Eiben A.; Schoenauer M.; Schwefel, H., Eds. Berlin, Germany: Springer-Verlag, pp. 865-874 (1998).
9. Socha, K. and Samples, M., Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art, in *Evolutionary Computation in Combinatorial Optimization* (EVOCop) III (Lecture Notes in Computer Science vol. 2611), Dorigo, M.; Di Caro, G.; Sampels, M., Eds. Berlin, Germany: Springer-Verlag, pp. 334-345 (2003).
10. Thompson, J. M. and Dowsland, K. A., A Robust Simulated Annealing based Examination Timetabling System, *Computers and Operations Research*, vol. 25, pp. 637-648 (1998).

Modelling and Solving the Generalised Balanced Academic Curriculum Problem with Heterogeneous Classes

Andrea Schaerf · Marco Chiarandini · Luca Di Gaspero

1 Introduction

The *Balanced Academic Curriculum Problem* (BACP) consists in assigning courses to teaching periods satisfying prerequisites and balancing students' load in terms of credits and number of courses. The BACP planning horizon is divided in *academic years*, and each academic year is divided into *terms*. Each term of a year is a *teaching period* in which courses can take place. The problem consists in finding an assignment of courses to periods that satisfies certain load limits and prerequisites.

The first formulation of BACP has been proposed by Castro and Manzano [2], and it has been included in CSPLib [5, prob. 30] along with three benchmark instances. BACP, in the CSPLib formulation, has been studied by Hnich *et al* [6], Castro *et al* [1], Lambert *et al* [7], and Monette *et al* [8]. However, BACP is a very simplified model of the real problem that universities have to solve in practice. In fact, in its formulation it is implicitly assumed that a student takes all delivered courses without personal choices; whereas in practice a student can select among alternatives.

To try to overcome such limitation and to deal with the real cases, the formulation has been extended by Di Gaspero and Schaerf [4]. The problem defined by Di Gaspero and Schaerf [4], called GBACP (G for Generalised), makes it possible to specify several *curricula*, with courses shared among them. A curriculum is a set of courses representing a possible complete selection of a student. In this new formulation, courses have to be balanced for each single curriculum. Moreover, GBACP includes professor's preferences for teaching in specific terms, which are often taken into account in real situations.

Andrea Schaerf · Luca Di Gaspero
DIEGM, University of Udine
via delle Scienze 208, I-33100, Udine, Italy
E-mail: {schaerf,l.digaspero}@uniud.it

Marco Chiarandini
IMADA, University of Southern Denmark
Campusvej 55, DK-5000, Odense, Denmark
E-mail: marco@imada.sdu.dk

Instance	Periods (Years × Terms)	Courses	Curricula	Courses per curr.	Courses per period	Prereq.	Pref.
csplib8	8 (4 × 2)	46	1	46	5.75	33	0
csplib10	10 (5 × 2)	42	1	42	4.2	33	0
csplib12	12 (6 × 2)	66	1	66	5.5	65	0
UD1	9 (3 × 3)	307	37	34.62	3.847	1383	270
UD2	6 (2 × 3)	268	20	27.8	4.633	174	158
UD3	9 (3 × 3)	236	31	29.81	3.312	1092	198
UD4	6 (2 × 3)	139	16	25.69	4.281	188	80
UD5	6 (3 × 2)	282	31	34.32	5.72	397	162
UD6	4 (2 × 2)	264	20	27.15	6.787	70	110
UD7	9 (3 × 3)	302	37	33.89	3.766	1550	249
UD8	6 (2 × 3)	208	19	22.58	3.763	149	120
UD9	9 (3 × 3)	303	37	34.08	3.787	1541	255
UD10	6 (2 × 3)	188	15	25.07	4.178	214	110

Table 1 Statistics on the GBACP instances.

Di Gaspero and Schaerf [4] introduce six instances, called UD1 – UD6, obtained from real data from University of Udine, which are much larger than the CSPLib ones. They also propose a solution based on local search (LS), that finds easily the optimal solution for the CSPLib instances and provides a solution for the instances UD1 – UD6. Such instances turn out to be much harder to solve than the CSPLib ones, and their optimal cost remains unknown.

The GBACP has been further investigated by Chiarandini et al. [3] who study two solution approaches, one based on integer programming and one on local search. This latter is an improved version of the approach by Di Gaspero and Schaerf [4]. In addition, four new instances, called UD7–UD10 (still from University of Udine), have been added to the repository. All data on GBACP is available on the web at `http://www.diegm.uniud.it/satt/projects/bacp/` together with a program that validates the solutions.

Table 1, taken from [3], summarises the main features of the available instances.

2 GBACP with Heterogeneous Classes

In this work, we address another feature of academic curriculum scheduling that may be of interest to university. In order to gain more flexibility in their planning, some universities may wish to have *heterogeneous classes*, that is, having students attending a course in different years of their curricula while still having the course taught only once per year. In the GBAC model previously studied we did not allow this because each course was assigned to a period, which corresponds to a pair term/year that has to be the same for all students attending the course. Here, we extend that model to include also the possibility of *heterogeneous classes*. We call the extended model, GBACP with heterogeneous classes (GBACP-HC).

There are different ways to approach the new feature that a course can be attended in different years by different curricula in GBACP-HC. The model we investigated in this preliminary work consists in pairing a course with the curricula in which it appears and scheduling each pair separately in the term and the year. Then, since a course is taught only once during the academic year, the terms for all pairs with the same course must be imposed to be equal. Instead, the assignment of the year can be different for each pair. However, it is generally not advisable to have classes with large discrepancies in the academic age of the students because

for pedagogical reasons the level of academic maturity should not differ too much. Hence, excessive spread of year of the students taking a course must be penalized.

3 Local Search for GBACP-HC

We solved the model for GBACP-HC described in the previous section by local search methods. Let C be the number of courses, Q the number of curricula, Y the number of years, T the number of terms, and $P = Y \times Q$ the number of periods. In the LS procedure for GBACP [3], the *search space* consists of the assignment $A : C \to P$. The *neighbourhood relation* is defined by moves that either change the period p assigned to one course c or swap the periods p_1, p_2 of two courses c_1 and c_2.

For GBACP-HC, we designed the *search space* to be made of two separate assignments: courses to terms ($A_1 : C \to T$) and pairs course/curriculum to years ($A_2 : C \times Q \to Y$). This new search space leads us to the definition of two different *neighbourhood relations*. The first one is originated by the moves that either change the *term* of one course or swap the *term* of two courses. The second one is originated by the moves that either change the *year* of one pair course/curriculum or swap the *year* of two courses in one given curriculum.

The *objective function* is defined by the same components used in the GBACP [3], load balance and professors' preferences, plus a new component determined by the sum over the courses of the largest difference between the years in which students are assigned to take the course.

Experiments of various local search procedure with different combinations of the above given neighbourhoods on the instances UD1 – UD10 are currently ongoing and the results will be presented in the forthcoming full paper.

Two preliminary observations are worth mentioning. First, it is easy to see that a solution to GBACP is also a solution to GBACP-HC, in which the same year is assigned to all curricula. Therefore, one may wonder whether the best solutions obtained by LS for GBACP are also the best obtainable by LS for GBACP-HC. Unless an extremely high penalisation of year discrepancies is defined, this is not the case. Experimentally, we verified that all best known GBACP solutions provided as initial solutions for GBACP-HC are improved by the local search procedure that works on the space of the GBACP-HC solutions.

The second observation is that for solving a GBACP-HC instance it is helpful to solve first the GBACP model on the same instance and then refine the search on the GBACP-HC model. Indeed, at present the current best solutions for GBACP-HC is a two phase local search obtained by first executing the LS procedure for solving GBACP, and then executing the specific LS for GBACP-HC. This overall procedure obtains results that are much better than those obtained by executing directly the LS procedure for GBACP-HC.

References

1. Carlos Castro, Broderick Crawford, and Eric Monfroy. A quantitative approach for the design of academic curricula. In *HCI*, volume 4558 of *Lecture Notes in Computer Science*, pages 279–288. Springer, 2007.

2. Carlos Castro and Sebastian Manzano. Variable and value ordering when solving balanced academic curriculum problems. In *6th Workshop of the ERCIM WG on Constraints*, 2001.
3. Marco Chiarandini, Luca Di Gaspero, Stefano Gualandi, and Andrea Schaerf. The balanced academic curriculum problem revisited. Available from `http://www.optimization-online.org/DB_HTML/2009/12/2506.html`, December 2009. Submitted for publication to a journal.
4. Luca Di Gaspero and Andrea Schaerf. Hybrid local search techniques for the generalized balanced academic curriculum problem. In Maria J. Blesa, Christian Blum, Carlos Cotta, Antonio J. Fernández, José E. Gallardo, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, volume 5296 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 2008.
5. I. P. Gent and T. Walsh. CSPLib: a benchmark library for constraints. Technical report, Technical report APES-09-1999, 1999. Available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99), pp. 480-481, Springer-Verlag, LNCS 1713, 1999.
6. B. Hnich, Z. Kızıltan, and T. Walsh. Modelling a balanced academic curriculum problem. In *CP-AI-OR 2002*, pages 121–131, 2002.
7. Tony Lambert, Carlos Castro, Eric Monfroy, and Frédéric Saubion. Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. In *Artificial Intelligence and Soft Computing - ICAISC 2006*, volume 4029 of *Lecture Notes in Computer Science*, pages 410–419. Springer, 2006.
8. J.N. Monette, P. Schaus, S. Zampelli, Y. Deville, and P. Dupont. A cp approach to the balanced academic curriculum problem. In B. Benhamou, B.Y. Choueiry, and B. Hnich, editors, *Symcon'07, The Seventh International Workshop on Symmetry and Constraint Satisfaction Problems*, 23/09/2007 2007.

Optimizing Railway Schedules for the Simplon corridor*

Ralf Borndörfer† Thomas Graffagnino‡ Thomas Schlechte† Elmar Swarat†

Today the railway timetabling process and the slot allocation of trains is one of the most challenging problems to solve by a railway infrastructure company. Especially due to the deregulation of the transport market in the recent years several suppliers of traffic have entered the market. This leads to an increase of slot requests and then it is natural that conflicts occur among them. Our goal is to resolve them by producing a feasible and conflict free timetable where a maximum of utilization is attained.

From a mathematical point of view the optimization problem can be stated as a multi-commodity flow problem through a extremely large network in space and time with certain additional constraints. The problem is well known in the literature, but only recently practical problem sizes are tractable due to development of improved models and algorithms. Nevertheless a decomposition of the problem can be observed. On the one hand for networks, or at least for long railway corridors, only simplified macroscopic models with a simplified routing through the railway infrastructure are considered, as in [7, 4, 10, 2, 5, 3, 6, 12, 1]. On the other hand, routing through complex stations can be considered on a more detailed, but of course only on a local level, see [15, 8, 14, 11]. The only recent reference, to the best knowledge of the authors, describing the interaction of both approaches is [9] by using a top-down approach.

In this paper an bottom-up approach of automatic simplification to complex microscopic railway infrastructure data is presented and applied for the Simplon corridor. This aggregation technique condenses a microscopic representation of the railway system to its relevant parameters from a planning and optimization point of view. We prove error estimations for the transformation and evaluate the re-transformed solution schedules in the microscopic simulation tool `OpenTrack`, [13]. We present computational results to different optimization scenarios for the Simplon corridor using the integer programming based solver `TS-OPT`, [1]. Furthermore, the dimension of the Simplon corridor allows for extensive optimization experiments with several aggregation levels in space and time. Finally, we present a sensitivity analysis for the corridor capacity w.r.t. different discretizations.

References

[1] Ralf Borndörfer, Berkan Erol, and Thomas Schlechte. Optimization of macroscopic train schedules via TS-OPT. In *3rd International Seminar on Railway Operations Modelling and Analysis - Engineering and Optimisation Approaches*, 2009.

*This work was funded by the Federal Ministry of Economics and Technology (BMWi), project *Trassenbörse*, grant 19M7015B.

†Zuse Institute Berlin (ZIB), Takustr. 7, 14195 Berlin-Dahlem, Germany, Email {`borndoerfer, schlechte, swarat`}`@zib.de`

‡SBB AG Bern, Infrastruktur/Trassenmanagement, Switzerland, `thomas.graffagnino@sbb.ch`

[2] Ralf Borndörfer, Martin Grötschel, Sascha Lukac, Kay Mitusch, Thomas Schlechte, Sören Schultz, and Andreas Tanner. An auctioning approach to railway slot allocation. *Competition and Regulation in Network Industries*, 1(2):163–196, 2006.

[3] Ralf Borndörfer and Thomas Schlechte. Models for railway track allocation. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[4] U. Brännlund, P. O. Lindberg, A. Nou, and J.-E. Nilsson. Railway timetabling using langangian relaxation. *Transportation Science*, 32(4):358–369, 1998.

[5] Valentina Cacchiani. *Models and Algorithms for Combinatorial Optimization Problems arising in Railway Applications*. PhD thesis, DEIS, Bologna, 2007.

[6] Valentina Cacchiani, Alberto Caprara, and Paolo Toth. A column generation approach to train timetabling on a corridor. *4OR*, 6(2):125–142, 2008.

[7] X. Cai and C. J. Goh. A fast heuristic for the train scheduling problem. *Comput. Oper. Res.*, 21(5):499–510, 1994.

[8] Gabrio Caimi, Dan Burkolter, and Thomas Herrmann. Finding delay-tolerant train routings through stations. In Hein A. Fleuren, Dick den Hertog, and Peter M. Kort, editors, *OR*, pages 136–143, 2004.

[9] Gabrio Caimi, Dan Burkolter, Thomas Herrmann, Fabian Chudak, and Marco Laumanns. Design of a railway scheduling model for dense services. *Networks and Spatial Economics*, 9(1):25–46, March 2009.

[10] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.

[11] Alberto Caprara, Laura Galli, and Paolo Toth. Solution of the train platforming problem. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *ATMOS*, volume 07001 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[12] Frank Fischer, Christoph Helmberg, Jürgen Janßen, and Boris Krostitz. Towards solving very large scale train timetabling problems by lagrangian relaxation. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[13] Daniel Huerlimann. Opentrack - railway simulation.

[14] R. Lusby, J. Larsen, D. Ryan, and M. Ehrgott. Routing trains through railway junctions: A new set packing approach. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2006.

[15] Peter J. Zwaneveld, Leo G. Kroon, H. Edwin Romeijn, Marc Salomon, Stephane Dauzere-Peres, Stan P. M. Van Hoesel, and Harrie W. Ambergen. Routing Trains Through Railway Stations: Model Formulation and Algorithms. *Transportation Science*, 30(3):181–194, 1996.

A hyper-heuristic approach for assigning patients to hospital rooms

Wim Vancroonenburg[1] · **Mustafa Mısır**[1] · **Burak Bilgin**[1] · **Peter Demeester**[1,2] · **Greet Vanden Berghe**[1,2]

One of the tasks hospital admission administrators are faced with is the problem of optimally exploiting the infrastructure. Patients need to be assigned to a room that is suitably equipped and staffed for treating the patients' clinical condition, taking into account policies imposed by the hospital organisation. Given that the number of (especially elderly) patients is growing, it is becoming increasingly difficult for administrators to handle this task manually in an efficient manner.

Demeester et al. (2010) introduced the Patient Admission Scheduling problem in order to support this decision process. The problem can be described as follows. For a given planning period, patients need to be assigned to a hospital room for each night of their stay. Patients are characterised by their age, gender and pathology, and should be treated in rooms and departments that are in correspondence with these characteristics. In addition to that, it is assumed that patients are already attributed an admission date, and that an expected average length of stay has been determined for their diagnosis. Patients will stay in the hospital for this period without returning home. This period is fixed and cannot be changed.

Treatment of the patient's pathology may require the presence of certain equipment in the assigned room. For example, certain patients might require a ventilation machine to be present in their room. As such, their stay is also characterised by a set of needed and preferred room properties. Patients may also express some preferences concerning the room they wish to stay in. Finally, patients do not necessarily need to stay in the same room for their entire stay. It is possible that it is beneficial to move a patient to another room at some point during their stay. Transfers of patients are thus allowed, but should be avoided as they cause patient discomfort.

The hospital is divided into several departments, each of which are specialised in treating certain kinds of pathologies and which may or may not have some age

[1] KaHo Sint-Lieven, Information Technology, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
E-mail: {Wim.Vancroonenburg, Mustafa.Misir, Burak.Bilgin, Peter.Demeester, Greet.VandenBerghe}@kahosl.be

[2] Katholieke Universiteit Leuven Campus Kortrijk, Computer Science and Information Technology, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium

restriction (for example, paediatrics may be restricted to patients younger than 16). Each department has a set of rooms, which are equipped variably and have a certain bed capacity. This capacity may not be exceeded. Furthermore, rooms can be restricted to patients of a certain gender. A room can be restricted to male or female patients, or it can be so that patients of a different gender may not be assigned to the same room, but it does not matter whether either male or female patients stay in the room. In other words, the gender of the room in this case is determined by the gender of the first patient.

In practice, most of these constraints are considered soft. For the patient admission scheduling problem, only the room capacity constraint and the fixed period of stay are modelled as hard constraints. All other constraints are modelled as soft constraints. The goal is to find a complete assignment of patients to rooms that minimises violations of these (weighted) soft constraints.

Demeester et al. (2010) tackled the patient admission scheduling problem using a hybrid tabu search method. In this study, we apply a hyper-heuristic method to select perturbative low-level heuristics for solving this optimisation problem. Hyper-heuristics have been progressively used for solving a wide range of combinatorial optimisation problems. The reason for applying them to different problems is related to their generic nature. The generality is provided by isolating hyper-heuristics from any problem dependent structure: a hyper-heuristic does not know anything about the problem to be solved. It manages a set of lower level heuristics and tries to apply them to the problem in an appropriate fashion. While solving a problem instance, a hyper-heuristic determines which heuristics should be called in which order and when. In effect, the hyper-heuristic method that we consider can be described as a "*heuristic to choose heuristics*" (Burke et al. 2003a).

The characteristics of the low-level heuristics may affect the performance of a hyper-heuristic. Their improvement capabilities and their speed concerning moving from one solution to another solution are some important elements regarding the quality of a heuristic set. Taking these elements into account during a heuristic selection process may help to give more meaningful decisions. In Chakhlevitch and Cowling (2005) a heuristic set with many heuristics was reduced into a small heuristic set for determining a better heuristic subset. In Burke et al. (2003b) and Kendall and Hussin (2005a,b) simple tabu strategies were proposed to make some heuristics tabu based on their performances. In Han and Kendall (2003) a similar approach was utilised to prevent the heuristics that do not affect the value of the objective function being called in a genetic algorithm based hyper-heuristic.

In this study, we apply a heuristic exclusion procedure based on these elements for improving the overall quality of the heuristic set. We divide the search into phases of a certain number of iterations, which is a parameter to our algorithm. During each phase we measure the performance of all heuristics. The performance metric used, is based on the relative improvement per execution time of each heuristic. At the end of a phase the better performing heuristics are retained, while the lesser performing heuristics are made tabu for a certain number of phases (which is also a parameter to the algorithm). This way we try to obtain an elite subset of heuristics, while still allowing for variation of this subset as the search progresses. It is important that this subset can vary over time. It is quite possible that a computationally expensive heuristic is not interesting at the beginning of the search (where simpler, less expensive heuristics are also effective), but necessary at the end of the search where it is more difficult to find solutions that improve on the current solution.

For the patient admission scheduling problem we use the heuristics described by Demeester et al. (2010) and Ceschia and Schaerf (2009), as well as some variations on these heuristics. We will test our method on benchmarks available from the patient admission scheduling website (Demeester et al. 2008) and we will compare our results with those available from the literature.

References

E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Handbook of Meta-Heuristics*, chapter Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer Academic Publishers, 2003a.

E.K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, 9(3):451–470, 2003b.

S. Ceschia and A. Schaerf. Multi-neighborhood local search for the patient admission problem. In *Hybrid Metaheuristics* Ceschia and Schaerf (2009), pages 156–170.

K. Chakhlevitch and P. Cowling. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. *LNCS*, 3448:23–33, 2005.

P. Demeester, B. Bilgin, and G. Vanden Berghe. Patient admission scheduling benchmark instances. http://allserv.kahosl.be/∼peter/pas/index.html, 2008.

P. Demeester, W. Souffriau, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine*, 48 (1):61–70, 2010.

L. Han and G. Kendall. An investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Proceedings of Congress on Evolutionary Computation (CEC'03)*, volume 3, pages 2230–2237, 2003.

G. Kendall and N.M. Hussin. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In *Multidisciplinary scheduling: theory and applications: 1st International Conference, MISTA'03: Nottingham, UK, 13-15 August 2003: selected papers*, page 309. Springer Verlag, 2005a.

G. Kendall and N.M. Hussin. A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. In *Proceedings of the 5th Practice and Theory of Automated Timetabling (PATAT'04)*, volume 3616 of *LNCS*, pages 270–293. Springer, 2005b.

The Design and Implementation of an Interactive Course-Timetabling System

Anthony Wehrer and Jay Yellen

Department of Mathematics and Computer Science
Rollins College
Winter Park
Florida, 32789 USA
awehrer@rollins.edu jyellen@rollins.edu

We describe the design and implementation of a multi-objective course-timetabling system for the Science Division at Rollins College. In the traditional vertex-coloring approach to timetabling, all conflicts are regarded as equally undesirable, but when all such conflicts are considered, including those that might involve only one or two students, a conflict-free timetable is rarely attainable. A more realizable objective is to minimize total conflict severity, where conflicts are assigned different levels of severity. This is a more natural model for an actual timetabling problem, where the undesirability of assigning various pairs of courses to overlapping timeslots varies significantly. Our second objective is to create timetables that result in relatively compact schedules for the professors and students.

We report on our progress toward building a robust decision-support system for course timetabling whose strategies are based on a weighted-graph model that the second author has been developing since the early 1990's [Kiaer and Yellen (1992)]. Starting from the exam-timetabling system developed in [Carrington, Pham, et al (2007)] and [Burke, Pham, et al (2008)], there were several major changes and complications we had to confront in adapting the exam-timetabling system and its weighted-graph model to our course-timetabling system.

The primary objective (*hard constraint*) for the Toronto timetabling benchmark problems on which the exam-timetabling system was applied is to produce conflict-free schedules, and the secondary objective (*soft constraint*) is to minimize the number of students taking back-to-back exams, or, more generally, taking exams in close proximity [Carter, Laporte, et al (1996)]. Those objectives are in sharp contrast to those described above for our course-timetabling system. Moreover, for the Toronto problems, a conflict occurs only when one or more students are taking two exams at the same time. For our Rollins course-timetabling problem, two courses offered at the same time can conflict for several reasons of varying severity. The conflict that occurs when two courses are taught by the same professor or require the same room or equipment is clearly more severe than one that occurs when a few students want to take both courses. Moreover, there are several gradations of conflict severity between these two extremes.

Our weighted-graph model has several attributes that take into account gradations in conflict severity and desirability for compact schedules, as well as other complications such as room and timeslot suitability/availability and shared resources. The vertex- and color-selection heuristics that drive our construction are derived from the additional information that our model carries.

For the Toronto problems, the *n* timeslots are represented by the integers 1 through *n*, whereas, at Rollins, as with most actual course-timetabling problems, the timeslots usually consist of multiple days of the week, e.g., MWF 9-9:50, TTh 9:30-10:45, etc. This makes conflict and proximity considerations more complicated. In particular, different timeslots (colors) can still overlap, and the proximity between two timeslots is no longer a simple function based on the difference between the corresponding integers. Also, when a course is assigned a timeslot, there must be a suitable and available room for that timeslot. Room assignments are not considered in the Toronto problems.

Our system includes a graphical user interface (GUI) that enables the user to participate in the input, construction, and modification of a timetable. In the input phase, course incompatibility, instructor and student preferences, and desire for compact schedules all require subjective judgments. The GUI allows the user to quantify and convert this information to the weighted-graph model. In the construction and modification phase, the GUI enables the user to directly assign or reassign courses to timeslots while guided by heuristics.

Our recent work also includes the design and testing of continuous analogues of certain heuristics that were used in [Carrington, Pham, et al (2007)] and new combinations of other heuristics previously used, and we report on those results. In addition, our construction now includes a *backtracking* component driven by many of those same heuristics. For example, if, during the initial construction, a vertex is selected for which there is no satisfactory color assignment (according to some pre-defined threshold), then one or more vertices are selected for uncoloring to free up a satisfactory color for the given vertex. Having implemented these changes, we compare the timetables our system generates to the actual Rollins timetable that was manually created.

Finally, we discuss how our current system lends itself to incorporating a learning mechanism and feedback loop that uses characteristics of the solution generated to adjust various weighted-graph parameters. This could lead naturally to a hyper-heuristics approach (see, e.g., [Qu and Burke (2009)]).

References

[Burke, McCollum, et al (2008)] Burke, E.K., McCollum, B., McMullan, P., and Yellen, J., Heuristic Strategies to Modify Existing Timetables, presented at PATAT08, Montreal, Canada, 2008.

[Burke, Pham, et al (2008)] Burke, E.K., Pham, N., Qu, R., and Yellen, J., Linear Combinations of Heuristics for Examination Timetabling, submitted to Annals of Operations Research, November 2008.

[Carrington, Burke, et al (2007)] J.R. Carrington, N. Pham, R. Qu, J. Yellen, An Enhanced Weighted Graph Model for Examination/Course Timetabling, Proceedings of 26th Workshop of the UK Planning and Scheduling, 2007, 9-15.

[Carter, Laporte, et al (1996)] Carter, M. W., Laporte, G., and Lee, S., Examination Timetabling: Algorithmic Strategies and Applications, Journal of the Operations Research Society 47 (1996), 373-383.

[Kiaer and Yellen (1992)] Kiaer, L., and Yellen, J., Weighted Graphs and University Timetabling, Computers and Operations Research Vol. 19, No. 1 (1992), 59-67.

[Qu, Burke, et al (2006)] R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, and S.Y. Lee. A Survey of Search Methodologies and Automated Approaches for Examination Timetabling, Technical Report NOTTCS-TR-2006-4, School of Computer Science, University of Nottingham, 2006.

[Qu and Burke (2009)] Qu, R. and E. K. Burke, Hybridisations within a Graph Based Hyper-heuristic Framework for University Timetabling Problems, Journal of Operational Research Society 60, (2009), 1273-1285.

The Erlangen Advanced Time Tabling System (EATTS) Version 5

Peter Wilke

Abstract The latest version of the Erlangen Advanced Time Tabling System EATTS, a development and production environment to generate optimized timetables, is introduced. EATTS allows to describe timetabling problem with its associated resources, constraints, events and solutions using XML files for information interchange. A broad bandwidth of algorithms is available to generate solutions using single-core, multi-core or cluster of networked computers.

Keywords Implementation · Distributed Time Tabling System · Time Tabling Problem

1 Introduction

The Erlangen Advanced Time Tabling System EATTS is a development and production environment to generate optimized timetables. Timetabling problems are quite common and come in different versions, among them rosters, schedules and school timetables. They have in common that given resources have to be used as efficient as possible and that this requires planning with respect to the given constraints to obtain a decent plan.

The example shown here is a school timetabling problem. EATTS is designed to be capable to solve all types of timetabling problems, so the example shown here is selected because it is quite common and requires no special explanation. Here we focus on the overall structure of the systems. Its ability to describe timetabling problems is describes in detail in a paper focusing on the XML-based specification [Ost10].

Peter Wilke
Universitaet Erlangen-Nuernberg, Informatik 5, Martensstrasse 3, 91058 Erlangen, Germany
Tel.: +49 (9131) 85-27998
E-mail: Peter.Wilke@Informatik.Uni-Erlangen.DE

2 Resources

When looking at a school timetable the events to be planned are lessons, to which the resources like teachers, classes ans rooms have to be assigned. All resources are typed. Each type has as many attributes created by the user as required. Planning a timetable usually begins with compiling the resources, either by reading in a file or typing in them manually. The screenshot fig. 1 shows the input dialogue to enter the attribute values for a room. All resources of type "Room" have attributes name, capacity and groups. Groups indicated what type of subject can be taught in this room and to which pools the rooms belongs.

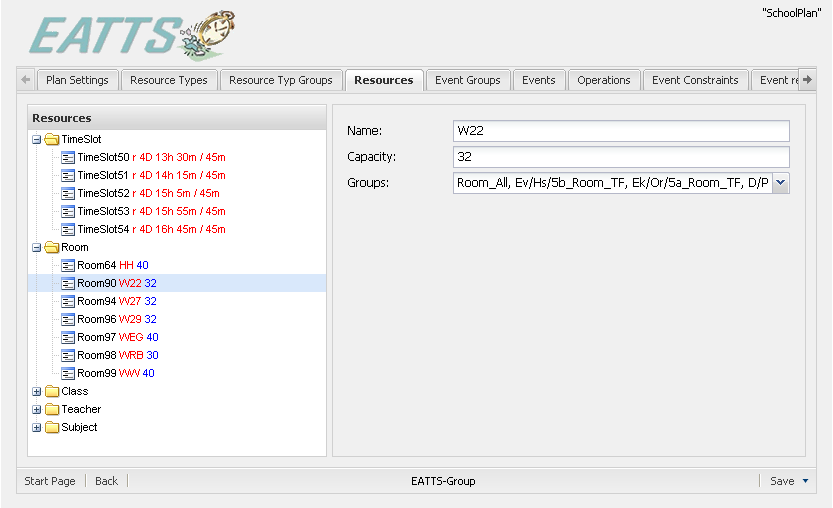


Fig. 1 The EATTS dialog to edit resources

3 Constraints

The specification of constraints is usually more complex than describing resources or events. On one hand a precise specification is required and on the other hand the current setting should be presented clearly to find gaps and/or inconsistencies, which can't be avoided automatically by EATTS but its user. Constraints come in different flavours, therefore a flexible way to specify them is necessary. EATTS allows to refer to resources, their object classes and all attributes. Depending on the type of the attributes, among them are integer, double and string, arithmetic and logical operators can be used to specify the constraint. In addition the parameters of the cost function are set, to compute the correct penalty point if the constraint is violated. The screenshot fig. 2 shows a constraint assigned to an event. Here the capacity of the class room "Room Size" must be greater than or equal (geq) to the number of students "Class Size". The cost function is specified as $50 * (1 * x^0 + 2)$. X denotes the number of violations of this constraint and the constants a chosen by the user according to the problem, e.g. severity of the constraint violation.

4 Modes of Operation

EATTS could be run in normal mode which interprets constraints the useual way. A unique property of EATTS is the option to specify a constraint not only as "has to be fulfilled (hard)" or "should be fulfilled (soft)" but also as "can be violated in exceptional case (soft hard)". This allows to violate constraints when it is acceptable, e.g. a room isn't available due to a broken water pipe, a teacher isn't available due to a traffic jam. In these cases a timetable should be created which is similar to the one currently in effect but violates some constraints to minimize changes. The mode is called the emergency mode as this is a typical approach in these cases, but it is not restricted to theses cases, e.g. finding out what effect a change in resources has on the resulting timetable: is it critical or is an investment in this resource required or efficient.

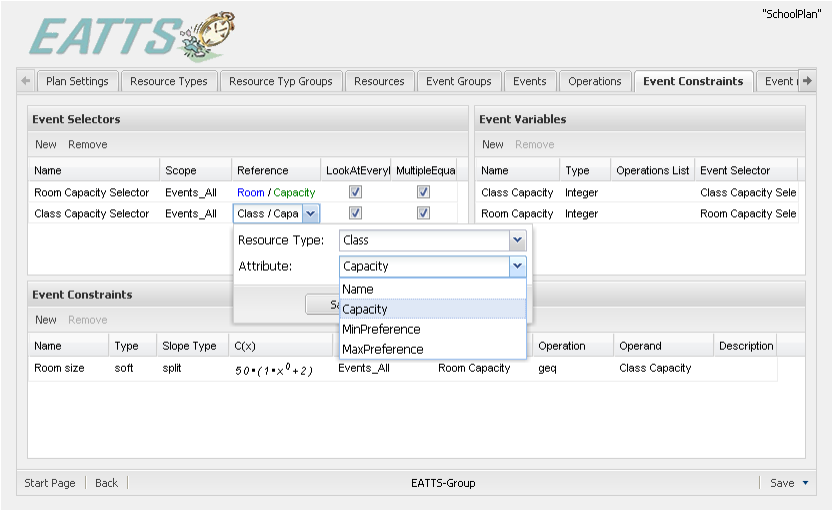


Fig. 2 The EATTS dialog to edit constraints

5 Events

The planning process is all about events. The timetable - as a result of the planning process - determines which resource is assigned to which event. For every event it can be declared how many resources of a certain resource type have to be assigned as minimal or maximal value so that the resulting timetable is a valid solution. Two additional subsets of resources of this type can be defined: one which is assigned fixed to this event and/or one subset of resources which could be assigned, at least one of them must exist.

6 Algorithms

Core of the planning are the algorithms. Depending on the nature of the given constraints and desired properties of the timetable the user can choose among a large number of available algorithms, among them Genetic Algorithms - Evolutionary Algorithms - Branch-and-Bound - Tabu Search - Simulated Annealing - Graph Coloring - Soft Computing - Swarm Intelligence - Great Deluge - Ant Colony - Jump Up Walk Down. The beginner might choose one of the pre-configured algorithms while the experienced user might prefer to set up his own parameter set or implement his own algorithms. All algorithms can be arranged in experiments which allows arbitrary computational sequences. The screenshot fig. 3 shows the parallel execution "ParallelAlgorithm" of two algorithms, namely "Simulated Annealing" and "Tabu Search". The results are compared and the best result is returned "BestResultMerger". All parameters of an algorithms can be displayed and edited. The configuration of an experiment can be saved and be used as template for a new experiment or just executed one more time.

EATTS is written in Java. As all algorithms use the same interface to the framework it is possible to excecute implementations of algorithms provided by the user. A tool to generate a template for the source code is under development, the byte code will be accessed via a class loader.

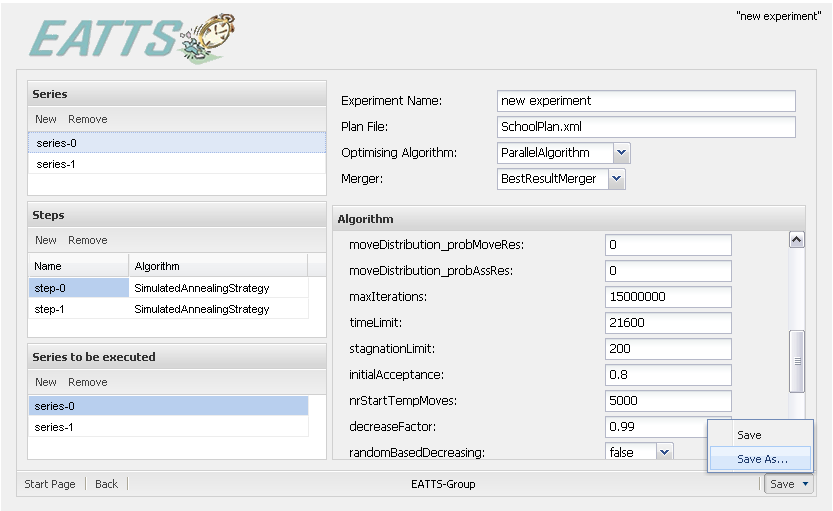


Fig. 3 The EATTS dialog to create experiments and to control algorithms

7 Running Experiments

The algorithms can be executed on a dedicated server or can be distributed over a TCP/IP network on additional computers. The screenshot shows the dialogue where the user can select the experiments and start their execution. The browser contacts the server regularly and updates the status information, including the costs of the best

plan found so far and an estimated remaining computation time. At the end of the computation results are stored and the data required for the views are generated. Now the user decided whether the results are satisfactory or additional computations are required.

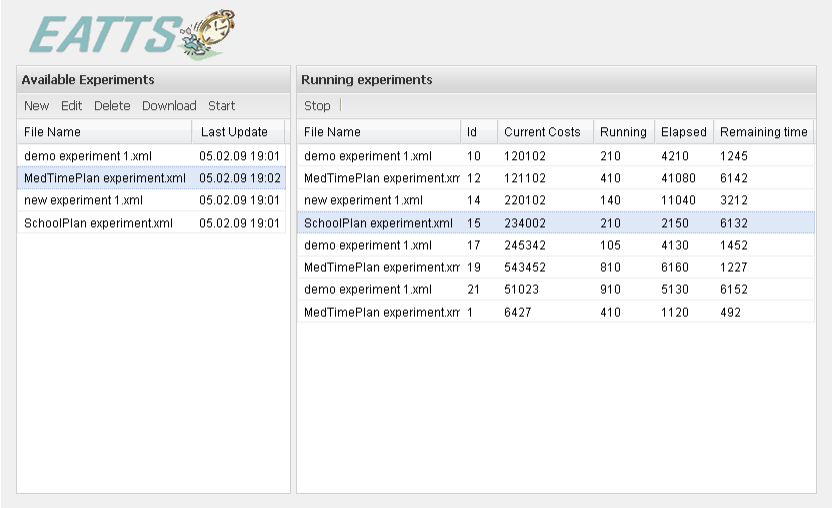


Fig. 4 The EATTS dialog to control the parallel computations

8 Results

Timetables are the output of the planning algorithms and can be stored in different file formats and views. The screenshot fig. 5 shows the view of a student on the generated plan, e.g. he sees his personal timetable consisting of the lessons he has to attend to. Other views can be created instantly, for a teacher, e.g. a headmaster or a caretaker. All users access the EATTS through a browser providing an interface according to the privileges of the user.

A common view indicates which constraints are currently not satisfied naming the events and resources. Based on this information the administrator can decide if he would like to edit the resources, events or constraints or use a different algorithm. The screenshot shows timeslot clashes (Monday, 9:45, and Tuesday, 9:45) and therefore the algorithm should be given more time to find a solution or another algorithm should be given an shot.

9 Implementation

The software is implemented in Java and available for numerous platforms, among them Windows and Linux operating systems. To run EATTS the following free-ware software products are required:

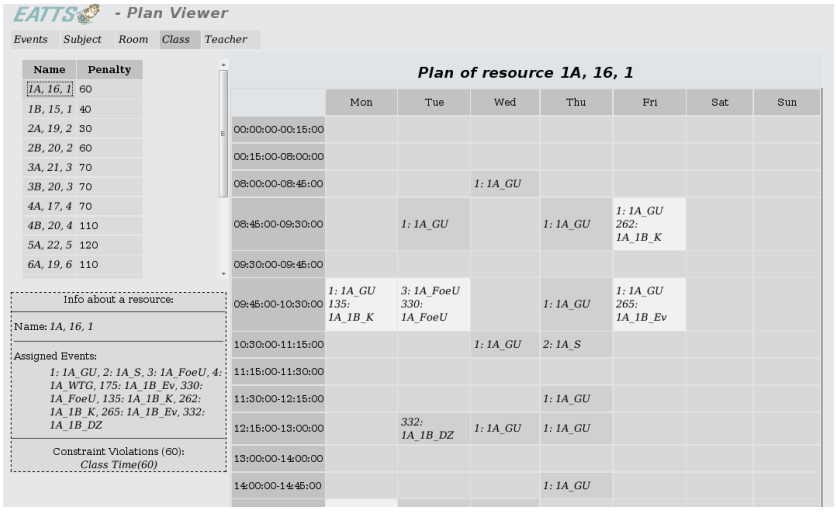


Fig. 5

– Browser to display the user interface, JavaScript must be enabled
– Java Runtime Environment (JRE), v5.0 or above
– computers connected via TCP/IP, if additional computing power is required (optional)

The EATTS Server is a software server, i.e. a PC or compute server can be used to host the EATTS Server.

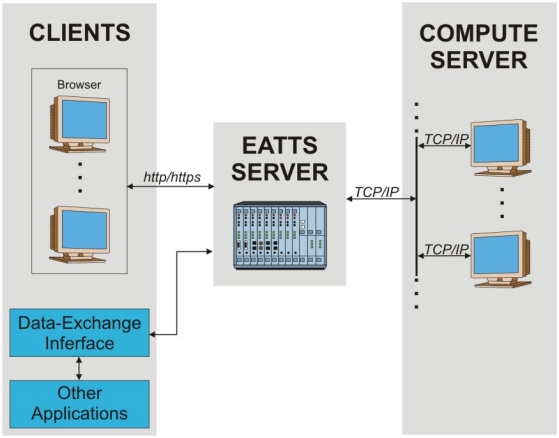


Fig. 6 The components of the Erlangen Advanced Time Tabling System

10 Conclusion

the Erlangen Advanced Time Tabling System is a framework to design and develop solutions for all kinds of timetabling problems. It can be used to experiment with algorithms and used as production system for real world systems.

If you would like to test the Erlangen Advanced Time Tabling System please feel free to contact the author for details.

References

[Ost10] Peter Osterler, Johannes; Wilke. The erlangen advanced timetabling system (eatts) unified xml file format for the specification of timetabling systems. page 18p, 2010.

Acknowledgements We would like to thank all our colleagues at EATTS for their support and special thanks to: Norbert Oster, Stefan Büttcher, Dr. Matthias Gröbner, Dimo Korsch, Sabine Helwig, Hichem Essafi, Marlene Gagesch, Monic Klöden, Georg Götz, Andreas Konrad, Tarek Gasmi, Sabeur Sarai, Kerim Merdenoglu, Helmut Killer, Eugen Kremer and Johannes Ostler.

Timetabling the major English cricket fixtures

Mike Wright

University of Lancaster

m.wright@lancaster.ac.uk

Every year the England and Wales Cricket Board (ECB) has to timetable all the major cricket fixtures in England (plus a few in Wales, Scotland, Ireland and the Netherlands), apart from those international matches which are timetabled by the International Cricket Council. This is a major undertaking which currently involves 21 clubs (mostly representing counties) playing about 420 matches in three overlapping competitions, as well as another 70 or so matches involving women's teams, touring teams, 'A' teams, Under 19 teams and university teams. Some of these matches will require four days, others only a single day or just an evening.

Over the years the structure of the competitions has changed substantially, with no one year being exactly like the last, with the speed of change increasing in recent years. This is likely to continue as the competitive environment changes, as marketers and broadcasters continue to try to maximise their commercial returns and as the cricket authorities continue to strive to produce competitions and formats which best serve the interest of the sport.

The ECB has several stakeholders to consider when timetabling the fixtures, including clubs, spectators, administrators, sponsors, broadcasters and the England national team. It is impossible to give all of these exactly what they want, so the problem inevitably involves compromise, with the aim of giving a good enough outcome for all stakeholders.

For the past 19 years this has been achieved using a semi-automated computer system which has always been successful in producing timetables that are at least satisfactory for everyone. Recently the use of the system has been extended to undertake "what if" exercises for different structures that the ECB are considering. Without such help it could be dangerous for the ECB to decide upon significant changes, since there would always be the danger of unexpected negative consequences, or even of the problem becoming infeasible. All runs are

made by myself, the system creator, rather than by the ECB themselves, as the levels of complexity are so high.

The first stage of the process is to ask the stakeholders what their constraints and preferences are, following which some quick preliminary analysis is undertaken leading to conversations between me and the ECB. For example, in 2010 there was one particular day on which more than half of the clubs in a division wanted a home match, which is clearly impossible, so discussions had to be undertaken before deciding who was going to be disappointed. In addition, there are usually points of clarification to be discussed concerning the precise meaning of the clubs' requests and decisions to be made as to their relative importance. While the main focus is on the 420 matches in the main three competitions, the requirements of the other 70 matches need also to be considered.

The timetabling problem for the matches in the three main competitions is set up as an optimisation model with large numbers of objectives and constraints, derived from a wide variety of stakeholder requirements and preferences. The solution procedure has changed slightly over the years, but has always been some kind of metaheuristic approach. The current method involves a variety of simulated annealing which uses a modified acceptance criterion depending on the effect not only of the overall change in cost but also the change in the individual subcosts, since this is a multi-objective problem.

However, before the whole timetable can be produced, the first stage is to produce matches for TV that satisfy the broadcasters, since they are the most powerful people involved. When doing this it is important to bear in mind the effect of the televised matches on the overall pattern of remaining matches; this effect can be quite substantial, especially for the home/away balance of the 4-day competition. Thus preliminary analysis is essential before selecting possible TV matches, including a series of "what if" runs of the model, so as to ensure that a good schedule can still be produced for the other stakeholders.

When the TV matches have been agreed, the rest of the timetable is addressed. However, this also cannot be a completely automatic process; interaction between user and computer is necessary for a really good solution to be produced. This is mainly because the weightings for the objectives have to be determined to some extent by trial and error; since the nature of the problem changes from year to year, it is never absolutely obvious which will be the hardest criteria to satisfy.

The proposed timetable is given to the ECB, who usually make some requests for minor changes, and further runs are made to try to accommodate these if possible. Then the timetable is passed on to the clubs, and again there will be a few requests for changes, some of which may be reasonable and others not, and of the reasonable ones some will be achievable but others not. Eventually, the modified timetable is accepted and published, and while one or two stakeholders may end up not entirely satisfied with what they have been given, the overall satisfaction levels are high.

SYSTEM DEMONSTRATION
Timetabling a University Dental School

Hadrien Cambazard[1], Barry O'Sullivan[1], John Sisk[1],
Robert McConnell[2], and Christine McCreary[2]

[1]Cork Constraint Computation Centre, University College Cork, Ireland
[2]Dental Surgery School, University College Cork, Ireland
{h.cambazard|b.osullivan|j.sisk}@4c.ucc.ie,
{r.mcconnell|c.mccreary}@ucc.ie

Abstract. We present a constraint model for a real-world University Dental School Timetabling problem, similar to post-enrolment timetabling. This model is used in a timetabling system we have developed for this application.

1 Introduction

We present a timetabling problem that arises in the Dental School of University College Cork (UCC). Due to an increasing number of students the school has begun considering automated approaches to generating the timetable for their five year academic programme. A diverse variety of university timetabling problems exist, but three main categories have been identified [3, 2, 5]: school, examination and course timetabling. The Dental School at UCC is dealing with a problem, which is similar to the post-enrolment university course timetabling problem [4] that occurs in a context whereby a set of courses that have been chosen by students must be scheduled into timeslots. University timetabling problems are usually solved using local search methods coupled with a wide range of meta-heuristics. Complete methods such as constraint programming (CP) are not yet scalable to real world instances. However, the problem tackled here is relatively small and we took this opportunity to develop a CP approach reusing many of the simple modeling ideas from [1]. A key feature of the resulting system is that it can prove that the standard set of constraints used by the Dental School admits no feasible timetables in the context of increasing student numbers. Therefore, the school must perform simulations to study potential remedies for this situation, e.g. by increasing capacity or by opening new timeslots. The application has been tested at UCC and has been embedded within an online timetabling system.

2 Problem Definition

The weekly timetable to be designed comprises two or three timeslots per day (morning, midday and afternoon) for five days which gives ten to fifteen timeslots. A number of events are to be timetabled. The events taking place are each characterized by a group of students and a subject. Several subjects exist, e.g "Dental Surgery", "Ortho", "OTL/Tutorial", "OTL/Pros", "Paedo", "Restorative", and "Study" (see Figure 1).

		Monday	Tuesday	Wednesday	Thurday	Friday
Morning	Dental_Surgery	[0/10]	4.2(8)[8/10]	4.3(8)[8/10]	4.1(9)[9/10]	4.4(8)[8/10]
	Ortho	4.4(8)[8/10]	4.3(8)[8/10]	4.2(8)[8/10]	5.2(9)[9/10]	1.1(10)[10/10]
	OTL_Tutorial	[0/12]	4.5(7)[7/12]	4.4(8)[8/12]	4.2(8)[8/12]	[0/12]
	OTL_Pros	3.1(11)[11/20]	3.1(11)[11/20]	3.1(11)[11/20]	3.2(10)3.3(10)[20/20]	3.2(10)3.3(10)[20/20]
	Paedo	[0/10]	4.4(8)[8/10]	4.1(9)[9/10]	4.3(8)[8/10]	[0/10]
	Restorative	5.1(10)5.2(9)5.4(9)[28/32]	5.1(10)5.2(9)5.4(9)[28/32]	5.1(10)5.3(9)5.4(9)[28/32]	5.1(10)5.3(9)5.4(9)[28/32]	4.1(9)4.2(8)4.3(8)4.5(7)[32/32]
	Study	[0/18]	5.3(9)4.1(9)[18/18]	5.2(9)4.5(7)[16/18]	4.4(8)[8/18]	[0/18]
Midday	Dental_Surgery	[0/10]			5.1(10)[10/10]	[0/10]
	Ortho	1.1(10)[10/10]			4.5(7)[7/10]	5.4(9)[9/10]
	OTL_Tutorial	[0/12]			4.1(9)[9/12]	4.3(8)[8/12]
	OTL_Pros	3.4(10)3.5(10)[20/20]			3.2(10)3.5(10)[20/20]	3.3(10)3.4(10)[20/20]
	Paedo	[0/10]			4.2(8)[8/10]	5.3(9)[9/10]
	Restorative	5.3(9)3.2(10)3.3(10)[29/32]			5.2(9)5.3(9)3.4(10)[28/32]	5.2(9)3.1(11)3.5(10)[30/32]
	Study	[0/18]			5.4(9)4.3(8)[17/18]	5.1(10)4.2(8)[18/18]
Afternoon	Dental_Surgery	[0/10]	5.2(9)[9/10]	5.3(9)[9/10]	4.5(7)[7/10]	5.4(9)[9/10]
	Ortho	[0/10]	5.3(9)[9/10]	5.1(10)[10/10]	4.1(9)[9/10]	[0/10]
	OTL_Tutorial	[0/12]	[0/12]	[0/12]	[0/12]	[0/12]
	OTL_Pros	[0/20]	3.4(10)3.5(10)[20/20]	[0/20]	[0/20]	[0/20]
	Paedo	[0/10]	5.4(9)[9/10]	5.2(9)[9/10]	5.1(10)[10/10]	4.5(7)[7/10]
	Restorative	4.1(9)4.2(8)4.3(8)4.5(7)[32/32]	4.1(9)4.2(8)4.4(8)4.5(7)[32/32]	4.1(9)4.3(8)4.4(8)4.5(7)[32/32]	4.2(8)4.3(8)4.4(8)[24/32]	4.4(8)2.1(10)[18/32]
	Study	[0/18]	5.1(10)[10/18]	5.4(9)[9/18]	5.2(9)5.3(9)[18/18]	[0/18]

Fig. 1. An example of a solution to the UCC Dental School timetabling problem.

Each has a specific maximum *capacity*, limiting the number of students who can attend the class at the same time. In other words, the same subject is always taught in the same room because each subject needs specific equipment. In a pre-processing step performed by the school, the students of each year are allocated to groups, which simply correspond to sets of students who are following the same set of subjects. A student group is defined by its *curriculum*, i.e., a multiset of subjects, and a *size*, i.e. the number of students in the group. Notice that the same subject can occur several times in a weekly period. For example, if group G_1 comprises ten students who have to attend one "Ortho" session twice during the week, one "Restorative" session and one "Paedo" session, $size(G_1) = 10$ and the curriculum, $c(G_1), is\{Ortho, Ortho, Restorative, Paedo\}$.

The events allocated in the timetable are basically defined by the curriculum of the groups. Event $e = (i, j)$ is associated with group i and the jth element of the curriculum of i. For example event $e = (1, 2)$ represents Group 1 attending "Ortho". The goal is to allocate all events to a timeslot of the timetable knowing that:

1. a group can only attend one subject in a given timeslot;
2. a group must attend all the subjects of its curriculum;
3. the number of students assigned to a given subject and timeslot must be smaller than the capacity of the subject;
4. some pairs of groups cannot follow the same subject at the same time;
5. some timeslots are initially forbidden for some groups and subjects.

Figure 1 presents an example of a solution to the problem. The groups are labelled $X.Y(s)$ in this example where X denotes the year of the group, Y its index and s is the size of the group. With each subject, we also indicate in square brackets the amount of space used compared to the space available. Restorative the monday morning is filled with 28 students out of the 32 seats available.

3 The Constraint Model

We use the following notation:

– S: the set of subjects. For a subject $s \in S$, $capa(s)$ is its capacity.
– G: the set of groups. For a group $g \in G$, we denote by $c(g)$ the curriculum of g and $size(g)$ the number of students in g.
– E: the set of events. For a given event $e \in E$, its group and index of the subject of the corresponding group's curriculum will be written $grp(e)$ and $sub(e)$. For $e = (1, 2)$, $sub(e) = 2$ and $grp(e) = 1$.
– I: the set of pairs of events with the same subject that are incompatible because the corresponding pair of groups cannot attend the subject at the same time.
– F : the set of events, timeslots pairs expressing the initial forbidden timeslots for some specific group and subject.
– nbt : the number of timeslots.

The timetabling is done using a Java open source constraint programming (CP) system, *Choco*[1]. The CP model is based on a variable x_{ij} per event expressing the timeslot in which the event is scheduled: $\forall i \leq |G|, j \leq |c(i)|\,, x_{ij} \in \{0, \ldots, nbt - 1\}$. The constraints are the following:

$$
\begin{array}{ll}
C_1 : \forall\, i < |G| & \textsc{AllDifferent}(x_{i1}, \ldots, x_{i,|c(i)|}) \\
C_2 : \forall\, t < nbt, s < |S| & \sum_{e=(i,j)\in E \text{ with subject } s}(x_{ij} = t) \times size(i) \leq capa(s) \\
C_3 : \forall (e_1, e_2) \in I & x_{grp(e_1),sub(e_1)} \neq x_{grp(e_2),sub(e_2)} \\
C_4 : \forall (e_1, s) \in F & x_{grp(e_1),sub(e_1)} \neq s
\end{array}
$$

Constraint C_1 states that a group cannot be in two rooms at the same time whereas C_2 enforces the total number of students attending a given subject to be lower than the capacity of corresponding subject/room; recall that subjects are taught in uniquely equipped rooms. C_3 states the incompatible groups and C_4, the forbidden timeslots.

The problem can be seen as list coloring, where events can be mapped to nodes and timeslots to colors, combined with knapsack constraints for each color limiting the *amount* of color that can be used in the coloring. One subject defines a 0-1 multi-knapsack problem, *i.e* a bin-packing. The size of this timetabling problem is relatively small, and this constraint model turns out to be efficient in practice. Nevertheless, hard instances were met when increasing the number of timeslots to accomodate the new students. Several redundant constraints can be added to improve performance:

1. If two events cannot fit together for a given subject because of their size, a redundant inequality constraint can be added to the model. These constraints are redundant given the knapsack but the presence of maximum cliques can considerably strengthen the constraint propagation.
2. Cliques in the coloring graph can be exploited using ALLDIFFERENT constraints.
3. Among all the events that must be assigned to a given subject, we can compute the k smallest that overload the capacity of the subject. The number of occurrences of a given value (timeslot) amongst all the events that are related to the corresponding subject is, therefore, bounded by k. A GLOBALCARDINALITYCONSTRAINT can be stated over all the corresponding events enforcing an upper bound on the number of occurrences of each timeslot to k. Notice that this constraint is a relaxation of all the knapsacks associated with a given subject, but it performs reasoning on

[1] `http://choco.emn.fr`

all the timeslots together as opposed to the knapsack constraints which operate independently on each timeslot.
4. All events related to the same group and the same subject are symmetrical. Such a set of events $e_1, \ldots, e_k$ can be ordered: $x_{grp(e_1),sub(e_1)} < x_{grp(e_2),sub(e_2)} < \ldots < x_{grp(e_k),sub(e_k)}$ as all permutations of these variables in the same solution are also solutions.
5. All events corresponding to groups of the same size with the same initial domain and the same curriculum are also symmetrical and can be permuted in any solution. We can safely order them in the timetable as explained above.

Redundant Constraints 1, 2, 4, 5 turn out to be critical to prove inconsistency of some instances. The problem differs from the Post Enrolment University Course Timetabling Problem [4] in terms of "room allocation". In the Dental School problem a set of events allocated to the same timeslot is subject to a knapsack constraint rather than a matching (events-room) constraint. The model presented here augmented with the redundant constraints seems to be efficient, but the decomposition strategy presented in [1] could be applied. In this context we would postpone the resolution of the knapsacks (using only the relaxation based on the redundant Constraint 3) once a coloring has been found, and infer cuts expressing that subsets of events cannot be together.

4 Conclusion

We have developed a simple timetabling system based on constraint programming for the Dental School at University College Cork. Its ability to prove inconsistency is its main originality and offers to the school a tool to simulate various scenarios to deal with increasing student numbers. A online application was designed and proved to be very useful to the school. We plan to add the computation of explanations to our system to offer feedback to the school to help overcome situations where no consistent timetables exist. By comparison, traditional timetabling system focus on minimizing the degree of violation of hard constraints to counter inconsistency. We believe that it is interesting to evaluate both approaches on this problem from the user's perspective.

Acknowledgements. Supported by Science Foundation Ireland (Grant 05/IN/I886).

References

1. Hadrien Cambazard, Emmanuel Hebrard, Barry O'Sullivan, and Alexander Papadopoulos. Local search and constraint programming for the post-enrolment-based course timetabling problem. *accepted to Annals of Operation Research*, 2009.
2. M. W. Carter and G. Laporte. Recent developments in practical course timetabling. In *PATAT*, pages 3–19, 1997.
3. D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, February 1985.
4. R. Lewis, B. Paechter, and B. McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Technical report, Cardiff University, 2007.
5. A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

System Demonstration of Interactive Course Timetabling

Tomáš Müller · **Keith Murray** · **Hana Rudová***

1 Introduction

This system demonstration presents an approach to interactive timetabling used by the UniTime university timetabling system. This application, which is publicly available under an open source license, has been successfully applied at Purdue University [8], a large public university (39,000 students) with a broad spectrum of programs at the undergraduate and graduate levels. The complete system includes course timetabling, examination timetabling, event management, and student sectioning.

UniTime has a completely web-based interface using the Enterprise Edition of Java (J2EE). Hibernate is used to persist data in an SQL-enabled relational database (e.g., MySQL or Oracle) and an XML interface can be used to tie the application with other systems used by a university. The course timetabling, examination timetabling and student sectioning problems are modeled as constraint satisfaction and optimization problems (CSOP) and solved using the solver library [3]. This constraint-based local search framework has also been successfully applied to the International Timetabling Competition 2007, where it was among the finalists in all three tracks and the winner of two [4].

A major goal of the system design has been to facilitate requests for changes in the timetable that inevitably occur. Interactive timetabling was previously explored in [7], concentrating on interactive removal of clashes. [1] investigated explanations in constraint programming to handle dynamic changes in the timetable. Generation of a timetable was interactively controlled by the user in [2]. Earlier work by the present authors also examined construction of a timetable using minimal changes to an initial solution [5]. The work pre-

* Hana Rudová is supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419.

T. Müller · K. Murray
Space Management and Academic Scheduling, Purdue University
400 Centennial Mall Drive, West Lafayette, IN 47907-2016, USA
E-mail: muller@unitime.org, kmurray@unitime.org

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

sented here encompasses an interactive mode for exploring possible changes, and easily making them, which was also found to be necessary. While this system demonstration focuses on making interactive changes to the course timetable, a similar approach is also used within the application for examination timetabling and event scheduling. Furthermore, an interactive phase of student sectioning (referred to as online student sectioning [6]) is currently being developed.

2 Interactive Timetabling

At Purdue, the complete university timetabling problem has been decomposed into a series of subproblems solved at the academic department level. Although each subproblem is solved separately, each solution considers all of the other problems for which a timetable has already been created. This coordination across problems is especially important when making interactive modifications to an existing timetable that may impact several others.

The course timetabling user interface consist of two parts: a data entry portion and a course timetabling solver. Basic data related to rooms, instructors, and courses (including all constituent classes) are entered via a series of web forms along with any associated preferences or requirements. Once all data have been entered into the system, the timetabling solver is used to create an automated timetable for the given (sub)problem. Subsequently, most changes are made using interactive timetabling. An exception is when multiple changes are desired in the input data. In this case, a new timetable is built from scratch or by using the minimal perturbation solver [5] which creates a solution to the modified timetabling problem while trying to minimize changes between the original and the new solution.

The same constraint model is used for both automated and interactive timetabling, with the same objective function consisting of satisfaction of

- preferences on time and rooms,
- distribution preferences that can be put between two or more classes (e.g., same room, back-to-back, or precedence),
- student conflicts (i.e., students that are expected to take two classes that either overlap in time or are back-to-back in rooms that are too far apart),
- divergence from the original solution, expressed as the number of students affected by a time change (room changes are usually consider less harmful),

along with several less important criteria. During interactive timetabling, the solver does not make any decisions, It does, however, provide users with a set of feasible solutions (and their associated costs) that can be reached via a backtracking process of limited depth. The user then determines the best tradeoff between accommodating a desired change and the costs imposed on the rest of the solution with a knowledge of what those costs will be. Moreover, to avoid a need for changing the input data, some of the hard constraints can be relaxed in the interactive mode. This means, for instance, that the user can put a class into a room different from the ones that were initially required. This is accomplished by making these hard constraints soft, but with too large of penalty imposed for the solver to suggest a change violating the constraint.

The timetabling user interface contains a set of pages that display various aspects of the current timetable. The user can view the classes in a time-resource grid for each resource (room, instructor, etc.), a list of assigned classes, or a list of yet-to-be-assigned classes. Changes to preferences or requirements made between the original and the current timetable, a history of the changes made using the interactive solver (which can also be used to easily

undo such a change if needed), and various reports displaying room utilization, student conflicts, and violation of other soft constraints are also available.

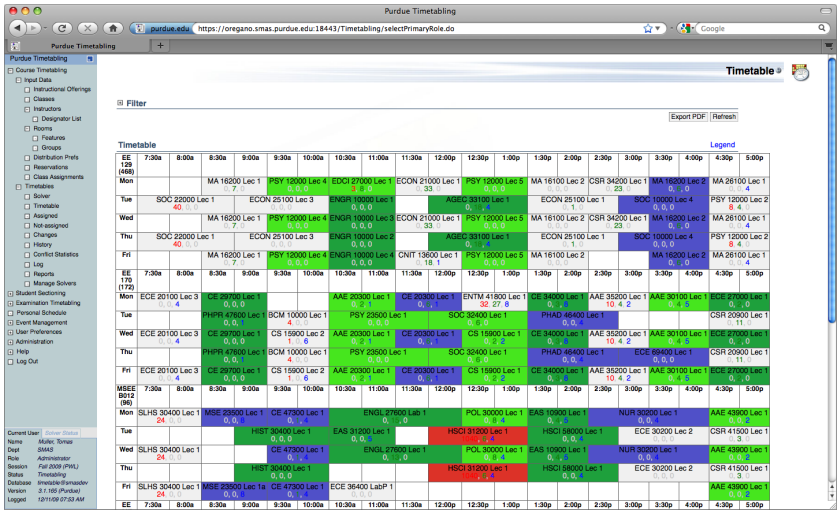


Fig. 1 Display of a timetable for given rooms.

A consistent color coding is used throughout the application. Required times or rooms are marked in blue, prohibited in red, light green and dark green for preferred and strongly preferred, yellow and orange, respectively, for discouraged and strongly discouraged. Figure 1 illustrates the timetable display provided for several rooms. The user can modify the assignment of one or more classes by clicking on a class in any of these views.

Each user interaction with the timetable can be seen as a sequence of changes to individual classes (beginning with the user selecting a class that needs to be changed). Figure 2 illustrates the information available to the user during each step of considering changes to the selected class of interest. The user may explore different options, consider various types of changes to the class, commit selected choices, or discard all changes considered. *Selected Assignments* describe changes already made to the timetable during the current interaction. *Conflicting Assignments* inform the user of any conflicts created in the timetable as a result of the selected assignments. *Suggestions* are optional changes the user may choose from that result in a feasible timetable. In each step (of an interaction), the user has the following possibilities:

- *Commit the change.* At this point, the actual timetable is modified, all the selected assignments are assigned and the conflicting classes, if any remain, are unassigned. The interaction is terminated.
- *Abandon the change.* The interaction is terminated without making any changes to the actual timetable.
- *Select a suggestion.* One of the suggestions is selected by the user, it is displayed together with the selected assignments. The user can still select another suggestion or try assign the class manually in the next step, e.g., if he or she is not satisfied with the resultant quality of the solution.

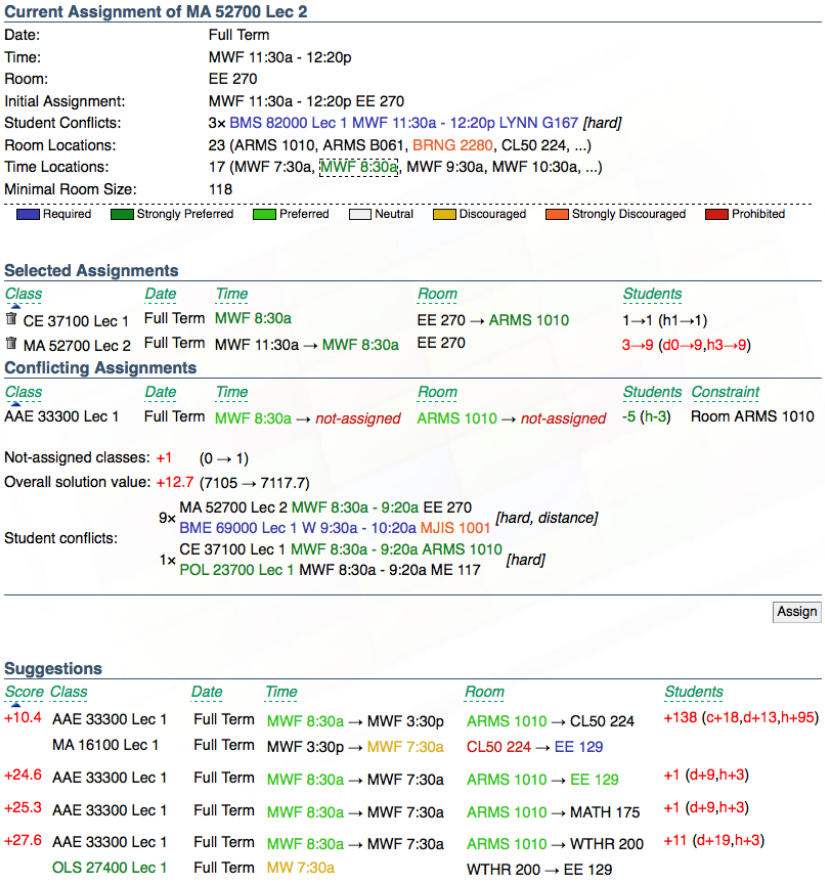


Fig. 2 Interactive solver interface for MA 52700 after selection of a new time assignment.

- *Select a placement.* Instead of choosing a suggestion, user picks a time and/or a room manually (for the selected class). This assignment is added into the list of selected assignments and the list of conflicts is recomputed together with the suggestions. The user may choose a different placement in the next step, e.g., if there are too many classes conflicting with the new assignment.
- *Remove a selected assignment.* An assignment is removed from the list of selected assignments, conflicts and suggestions are recomputed.
- *Select another class.* A different class (e.g., one of the conflicting classes) is selected. Suggestions are recomputed to include the selected class.

Besides the above actions, the user has a wide variety of additional choices that may help to find a desired change. For instance, the list of available suggestions can be filtered by additional criteria or the number of allowed additional changes of the provided suggestions can be increased.

Suggestions are computed using a branch and bound algorithm of a limited depth (initially allowing 2 additional changes). This process starts from the list of selected assignments, trying all possible placements for the selected class and resolving hard conflicts cre-

ated by these changes. Only solutions that do not violate any hard constraints are allowed, which along with the quality of the n-th best solution found (n being the number of suggestions to be displayed) bounds the search. A more formal description of the algorithm is available in [8]. It is also shown here, based on comparison runs with no time limit on the branch and bound solutions, that a 5 second time limit is sufficient to present an optimal suggestion incorporating up to two additional changes in more than half of the cases (for the timetabling problem at Purdue). This time limit is important to keep the user interface interactive; however, when the limit is reached, the user has the option (e.g., when no desired suggestion is found) to recompute these suggestions with an increased time limit.

3 Conclusion

An extension of the UniTime course timetabling application has been presented that allows users to modify automatically computed timetables when a small number of changes are necessary to accommodate new parameters added after a timetable has been published. This interactive component allows the user to to find high quality options for meeting the additional problem parameters and deciding whether to modify the previous solution. The presented application is publicly available under an Open Source license[1], and can be downloaded from the UniTime web site `http://www.unitime.org`. This site also contains information about ongoing research, online documentation for the described system, and various real-life benchmark data sets for course timetabling, examination timetabling and student sectioning problems.

References

1. Hadrien Cambazard, Fabien Demazeau, Narendra Jussien, and Philippe David. Interactively solving school timetabling problems using extensions of constraint programming. In Edmund Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling V*, pages 190–207. Springer-Verlag LNCS 3616, 2005.
2. Hans-Joachim Goltz and Dirk Matzke. Combined interactive and automatic timetabling. In *Proceedings of the Practical Application of Constraint Technology and Logic Programming*, pages 529–535. The Practical Application Company Ltd, 1999.
3. Tomáš Müller. *Constraint-based Timetabling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2005.
4. Tomáš Müller. ITC2007 solver description: a hybrid approach. *Annals of Operations Research*, 127:429–446, 2009.
5. Tomáš Müller, Roman Barták, and Hana Rudová. Minimal perturbation problem in course timetabling. In Edmund Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling, Selected Revised Papers*, pages 126–146. Springer-Verlag LNCS 3616, 2005.
6. Tomáš Müller and Keith Murray. Comprehensive approach to student sectioning. In *PATAT 2008 – Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada, 2008.
7. Sylvain Piechowiak, Jingxua Ma, and René Mandiau. An open interactive timetabling tool. In Edmund Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling V*, pages 34–50. Springer-Verlag LNCS 3616, 2005.
8. Hana Rudová, Tomáš Müller, and Keith Murray. Complex university course timetabling. *Journal of Scheduling*, 2010. To appear.

[1] Constraint-based solver, including course timetabling, examination timetabling and student sectioning extensions is available under GNU Lesser General Public License (LGPL), the complete timetabling application is available under GNU General Public License (GPL).

Author Index

Author Index

Author Index

Author Index

PATAT 2010

8th International Conference on the Practice and Theory of Automated Timetabling

Queen's University Belfast, Northern Ireland, 10th - 13th August 2010