PATAT 2014

10th International Conference on the Practice and Theory of Automated Timetabling

York, United Kingdom, Tuesday 26th - Friday 29th August 2014

CONFERENCE PROCEEDINGS

PATAT 2014

Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling

26 - 29 August 2014, York, United Kingdom

Edited by:

Ender Özcan, University of Nottingham, UK
Edmund K. Burke, University of Stirling, UK
Barry McCollum, Queen’s University Belfast, UK

ISBN 978-0-9929984-0-0
Published by PATAT

ASAP
research group

CHORDS

Computational
Heuristics
Operational Research
Decision-Support

EVENTMAP

EURO
The Association of European
Operational Research Societies

WATT
EURO Working Group on Automated Timetabling

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 tenth conference in York, United Kingdom. The PATAT conferences, which are held biennially, serve as a forum for an international community of researchers, practitioners and vendors on all aspects of computer-aided timetable generation and related issues. This includes personnel rostering, educational timetabling, sports scheduling and transportation timetabling.

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. A dedicated stream entitled "Practical Timetabling" is included bringing both practitioners and theoreticians together with the goal of addressing the well recognised gap which exists between the practice and theory of automated timetabling. It is intended that the PATAT conferences continue to integrate and combine both the research and practice agendas across all areas of timetabling.

The programme of this years conference features 76 presentations which represent the state-of-the-art in automated timetabling: there are 5 plenary papers, one of them is from the Scientia Ltd representing practitioners, 26 full papers, 38 extended abstracts, 5 system demonstrations and 2 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 were the cases in Montreal in 2008, Belfast in 2010 and Son in 2012, a post-conference volume of selected and revised papers is to be published in the Annals of Operational Research journal. 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 in a timely manner. As always we are grateful to all authors and delegates. Special thanks go to the organising committee for their 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 York. We hope you enjoy the conference talks and networking opportunities provided.

Ender Özcan, Edmund K. Burke and Barry McCollum

PATAT 2014 Programme Committee

PATAT 2014 Organising Committee

Una Benlic	University of Stirling, UK
Tim Curtois	University of Nottingham, UK
Mike Earl	Loughborough University, UK
Angeliki Gretsista	University of Stirling
Bob John	University of Nottingham, UK
Vicki Lawlor	University of Stirling, UK

PATAT Steering Committee

Edmund K. Burke (Chair)	University of Stirling, UK
Barry McCollum (Treasurer)	Queenś University Belfast, Northern Ireland, UK
Patrick De Causmaecker	Katholieke Universiteit Leuven, Belgium
Jeffrey H. Kingston	University of Sydney, Australia
Ender Özcan	University of Nottingham, UK
Atle Riise	SINTEF, Norway
Hana Rudova	Masaryk University, The Czech Republic
George White	University of Ottawa, Canada

Table of Contents

Plenary Presentations

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Scheduling in an unknown, diverse consumer world!

Different challenges, different response!

Paul Harrington · Geoffrey Forster

Abstract Resource scheduling within the Higher Education environment has been focused on improving the utilisation of ever decreasing resources to construct a schedule that delivers an optimised use of resources in the delivery of a fixed set of outcomes.

With the Higher Education Sector competing globally for a more demanding, consumer oriented, technology savvy group of students institutions are having to offer more student choice. This student demand and flexibility of choice presents challenges to the current generation of scheduling tools and to the working practises of scheduling departments.

The pace of change in the HE sector, all over the globe, has been so fast and transformational in the recent years that the usual responses on the edges and the surface will not suffice.

Introduction

Resource scheduling within the Higher Education environment has been focused on improving the utilisation of ever decreasing resources to construct a schedule that delivers an optimised use of resources in the delivery of a fixed set of outcomes.

With the Higher Education Sector competing globally for a more demanding, consumer oriented group of students, institutions will need to offer more student choice and provide better advice to students. This student demand and flexibility of choice presents challenges to the current generation of scheduling tools and to the working practises of scheduling departments.

The pace of change in the HE sector, all over the globe, has been so fast and transformational in the recent years that the usual responses on the edges and the surface will not suffice. We believe that a re-conceptualization of what

Mohamad Djahanbakhsh, CEO and Geoffrey Forster, Director
Scientia

a next generation planning and scheduling system should provide is required, and the paper describes some of these new concepts.

Scientia is the established market leader in timetabling and resource scheduling solutions for the Higher Education sector worldwide. Our Syllabus Plus products provide solutions to the problems faced by organisations in optimising their assets and resources, most notably time and space, and in a way that best meets the needs of staff and students.

The approach for these solutions has been to define the problem to be solved as early as possible, providing as much concrete information for the scheduling engine to use to create a solution. This requires the working practises within the University to capture requirements early. With the introduction of greater student choice and more emphasis on student satisfaction, these requirements are more fluid until much later in the scheduling process.

Planning, Scheduling and Allocation is business critical with the capability to improve revenues and reduce costs, and will be used by everyone at the institution every day of the year. This will require more effective integration with the other core business systems like SIS and LMS.

Trends in HE that will impact planning and timetabling

Cost pressures to get more for less

The reduction in HE budgets seen in the Western World is continuing to drive the need for further optimisation of resources, not only within the academic schedule, but making maximum use of resources through commercial enterprise.

With these tightening budgets systems will need to be capable of effective cost modelling and scheduling decisions will not only be about meeting the needs of staff and students but also cost effectiveness. These budgetary constraints will have an impact on staff workload planning, with institutions looking to ensure that they have the right balance of research, lectures and teaching. This will be important for attracting the best lecturers and invaluable funding for research.

Enhancing the student experience

The cost of their education and career opportunities are important to many students. They want to know that they will graduate by a particular date, with the courses that they wanted, that fit around their requirements. This could include features such as employability, previous success rates of particular study paths and other advice and guidance.

They expect the university to create a schedule of classes that satisfies their academic requirements and allocates them to a set of classes that takes their personal preferences for attendance into account.

Students may have gaps in attendance in order to clear debts or take internships. Some students may prefer a 2-year degree that condenses 3 years of study into 2 years, and take no holiday.

Short courses for students are offered throughout the academic year and students may take courses from other institutions.

MOOC's & blended learning

Massive open online courses (MOOCs) are a revolutionary element of higher education today. Institutions may embed MOOCs from other institutions in their awards. Scheduling resources for virtual courses to fit the demands of students from around the globe presents a fresh and exciting challenge.

Blended learning will increase the demand for new meetings at short notice.

- Classes offered off campus.
- Blended learning.
- Flipped lectures.
- Classes offered across time-zones.

Rapid change & unpredictable demand

Student meetings are increasingly planned and scheduled at short notice. The requirement still remains that there is an efficient use of resources. This lack of certainty around requirements presents significant challenges to both the systems and working practises of scheduling departments.

Features of a Next Generation planning and scheduling system

Undoubtedly the next generation set of solutions will be based around new and emerging technologies, taking advantage of the capability of the cloud for high computational power that can flex on demand. Collaborative technologies will enable the scheduling departments to work more efficiently with academic departments while designing the schedule, helping to make decisions quickly and effectively.

The system needs to be able to continually optimise throughout the scheduling cycle, with different rule sets firing at differing points of the cycle. It needs to be able to cope with change, recognising that change happens but has to keep disruption to a minimum.

Some key factors in delivering that solution are explored below.

Smarter Anticipation of Demand

One approach to reduce the amount of change in the requirements could be to use past data to better anticipate demand. Using the information about pathways that previous students took on programmes would enable pseudo class sizes to be created and space provided for students to take the course. This of course is what academics will say is done when planning class sizes, in reality this is more likely to be a constraint of supply and restricts the flexibility of choice that is required to meet student satisfaction.

The use of guided allocation when collecting student requirements would also enable the problem to be modelled more effectively. The presentation of information to the student while they are selecting the courses they wish to undertake. This information could include data such as success rates of previous students and pathways that previous successful students took. Collecting students' preferences for courses that meet requirements such as childcare arrangements as early as possible also allows for improved modelling and reduces the potential for change.

An exciting opportunity presents itself in the use of big data to anticipate real world demand for specific programmes. The use of inference engines to analyse the data points such as employability, popularity of subject, scarcity of qualified people (skills gap), number of offerings, career paths etc. provide an opportunity to the university to increase competitiveness by providing programmes that meet real world demand. This anticipation of demand will enable management to more effectively manage the long term decisions of manpower planning. This anticipation ensures that the right mix of staff is recruited to ensure that the courses that are in highest demand are able to meet the demand.

Early Planning or Lazy Allocation?

As has been seen allowing student choice creates uncertainty in the scheduling process, the most common approach to solving this problem is to complete the planning as early as possible in the process presenting a clear statement of the requirements to be scheduled.

Where this isn't possible the challenge is “to introduce the minimal amount of perturbation into the solution already created”. This challenge can be further refined as “to introduce the minimal amount of perturbation into the published elements of the schedule”.

This redefinition of the problem allows the use of “lazy allocation” of resources to any activity to be scheduled. This lack of eagerness to resolve the problem fully early in the scheduling process presents greater degrees of flexibility when requirements do change late in the process. The trick is to allocate the minimum resources required and “reserve” types of resources. This ensures that a theoretical solution exists for the schedule created and allows for problem resolution without being constrained to specific resources to early in the process.

Scheduling for MOOCs

What happens when time and space become almost limitless? When lectures no longer need to be delivered within the boundaries of a classroom or even attended at a set time. The challenges for scheduling MOOCs will present a different challenge with very different demands on access to resources that current systems are not designed to meet.

While lectures may no longer need to be scheduled, access to high quality support will be required to students looking for a blended approach to learning and the Universities that can deliver this will be those most successful in recognising the commercial opportunity that MOOCs undoubtedly bring. This access could be in the form of physical face to face meetings with tutors or online meetings with tutors and mentors. It will become important to consider factors such as geographical location and time zones to ensure accurate matching of students to appropriate resources. This matching of resources will be on increased data set sizes, it is not unknown for MOOC courses to have over 100,000 students.

This new challenge will require very different approaches to scheduling with simpler constraints but higher demand. This will require the matching of students? demand to available resources to be carried out in a just in time manner. This approach could learn much from the resource management of companies such as Amazon where meeting the demands of the consumer is the difference between survival or closure.

Conclusion

It is clear that the current generation of Scheduling Solutions have to evolve to meet the rapidly changing requirements in the HE sector.

The command and control nature of current solutions where certainty is required before an efficient and effective schedule can be created will not be able to meet the challenges of this new consumer focused, student experience led revolution. Systems that can adapt and manage large degrees of uncertainty and are capable of guiding the user and presenting solutions to the problems as they emerge will be the dominant successful solutions in this new diverse consumer world.

Scientia and Syllabus Plus

Syllabus Plus was released in 1991 and was the first commercial product that could generate an optimised schedule for an institution, taking into account human and physical resources and student course requirements. The Syllabus

Plus scheduling engine has hard constraints and preferences and provides multi-user cooperative scheduling, so it can scale to schedule the largest institution. Auto-scheduling uses a heuristic based upon the difficulty of scheduling each activity. Manual scheduling allows a scheduler to reschedule part of a schedule, and with the engine giving advice about the best way to make a change. Important additional products has been added to Syllabus Plus including: Award and course planning tools, staff workload management, student award planning, student self-allocation to classes, staff and student timetables, data analysis reports, and examination scheduling. Enterprise Syllabus Plus was released in 2008, allowing anyone at an institution to access the services that they were allowed to access from a web browser.

"Mine's better than yours" – comparing timetables and timetabling algorithms

Ben Paechter

Automated timetabling papers, along with papers about all types of optimisation methods are full of claims that a particular algorithm, or adaption to an algorithm "out-performs the state-of-the-art". If science is going to move the world forward then we need to be sure what we mean when we make claims such as this and have ways of verifying that the claim is indeed the case. Taking a tutorial approach, this talk begins by looking at how we might decide how good a particular timetable solution might be or how we might, at least, compare one solution with another. Ways of considering the hard and soft constraints are examined, along with methods of dealing with multiple objectives. Three classifications of soft constraints are defined. The need to work with the person using the automated system to fully understand what is "good" about a timetable is underlined. This might include non-obvious criteria, such as the need to reduce the chance that users of the timetable will be able to suggest variations which lead to improvement.

Once we understand how we can compare two solutions to a particular timetabling problem, we can then look at how we might compare two algorithms trying to find good solutions to problems. Factors that might be taken into account are, for example, speed, reliability, closeness to optimum, or the chances of a really super result once in a while. Non-obvious criteria might be for example the extent of the ability for the user to change their mind about what they care about (in terms of either timetable or algorithm quality) in the middle of producing the timetable. Again, in order to be really useful to the world, the emphasis here needs to be on finding out what the users of the algorithm really want from it. The use of standard problem instances is examined along with analysis of the conditions that make this is useful or not. The talk argues that, in order to be useful, problem instances need to be accompanied by one or more standard sets of criteria on which solutions will be measured

Edinburgh Napier University, UK
E-mail: B.Paechter@napier.ac.uk

and, just as importantly, one or more sets of criteria on which algorithms will be measured.

Even when there are clear criteria for judging between algorithms on a specified set of problem instances there can still be problems interpreting the results meaningfully. For example we have to be careful that an algorithm is not just good at solving the particular problem instances, i.e. over-fitted to those instances. One way to try to solve this problem is to have public competitions where algorithms are developed on one set of problem instances, and then compared on another set. Competitions also help to encourage work in a particular area, and work which is genuinely comparative. The talk will discuss the author's experience of running competitions, and what makes a successful competition. The problem of competition entrants using different hardware, compilers, interpreters and operating systems is also discussed. Competitions have largely concentrated on hidden problem instances having the same user criteria. The author will argue that the future may lie in competitions where hidden problems change the user requirements in some way, so as to encourage the development of algorithms which are more generally useful.

Visualising the diversity of benchmark instances and generating new test instances to elicit insights into algorithm performance

Kate Smith-Miles

Objective assessment of optimization algorithm performance is notoriously difficult, with conclusions often inadvertently biased towards the chosen test instances. Rather than reporting average performance of algorithms across a set of chosen instances, we discuss a new methodology to enable the strengths and weaknesses of different optimization algorithms to be compared across a broader instance space. Results will be presented on timetabling, graph colouring and the TSP to demonstrate: (i) how pockets of the instance space can be found where algorithm performance varies significantly from the average performance of an algorithm; (ii) how the properties of the instances can be used to predict algorithm performance on previously unseen instances with high accuracy; (iii) how the relative strengths and weaknesses of each algorithm can be visualized and measured objectively; and (iv) how new test instances can be generated to fill the instance space and provide desired insights into algorithmic power.

School of Mathematical Sciences
Monash University, Australia
E-mail: kate.smithmiles@gmail.com

Passenger Oriented Railway Disruption Management
By Adapting Timetables and Rolling Stock Schedules

Lucas P. Veelenturf · Leo G. Kroon · Gábor Maróti

Abstract In passenger railway operations, unforeseen events require railway operators to adjust their timetable and their resource schedules. The passengers will also adapt their routes to their destinations. When determining the new timetable and rolling stock schedule, the railway operator has to take passenger behavior into account. The capacity of trains for which the operator expects more demand than on a regular day should increase. Furthermore, at locations with additional demand, the frequencies of trains serving that station could be increased.

This paper describes a real-time disruption management approach which integrates the rescheduling of the rolling stock and the timetable by taking the changed passenger demand into account. The timetable decisions are limited to additional stops of trains at stations at which they normally would not call. Several variants of the approach are suggested, with the difference in how to determine which additional stops should be executed.

Real-time rescheduling requires fast solutions. Therefore a heuristic approach is used. We demonstrate the performance of the several variants of our algorithm on realistic instances of Netherlands Railways, the major railway operator in the Netherlands.

Keywords Railways · Disruption Management · Rolling Stock · Timetable

L.P. Veelenturf and L.G. Kroon*
Rotterdam School of Management, Erasmus University
Department of Technology & Operations Management
P.O. Box 1738, 300 DR Rotterdam, The Netherlands
E-mail: lveelenturf@rsm.nl, lkroon@rsm.nl

G. Maróti*
VU University Amsterdam, Faculty of Economics and Business Administration
De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
E-mail: g.maroti@vu.nl

* Netherlands Railways, Process quality & Innovation
Utrecht, The Netherlands

1 Introduction

In passenger railway operations, unforeseen events (such as infrastructure malfunctions, accidents or rolling stock breakdowns) can make parts of the railway infrastructure temporarily unavailable. Then it is not possible to operate the timetable, rolling stock schedule and crew schedule as planned. Within minutes, or even better, seconds, a new timetable and new resource (rolling stock and crew) schedules must be available. In Cacchiani et al. [2] an overview is given of recovering models and algorithms to solve these rescheduling steps. In this overview it becomes clear that, although the schedules are interdependent, most research focuses on rescheduling one of the schedules at a time. By the complexity of the rescheduling problems, there is not enough time to solve the integrated problem. In this paper we partly integrate the rescheduling of the rolling stock plan and the timetable. Our particular focus lies on passenger service, and we take passenger behavior explicitly into account.

We propose a fast passenger oriented rolling stock rescheduling approach which allows to slightly adapt the timetable. Current literature on integrated rescheduling of the timetable and the rolling stock schedule is scarce. Adenso-Díaz et al. [1] and Cadarso et al. [3] did research on integrated rescheduling of the timetable and rolling stock on cases of the Spanish railway operator RENFE. Like the main focus of our paper, Cadarso et al. [3] take the dynamics of the passenger behavior into account. However, the fundaments of the approach in the current paper are from Kroon et al. [8]. In Kroon et al. [8] the focus is on improving passenger service by considering passenger behavior while rescheduling the rolling stock. Kroon et al. [8] use an iterative procedure for rescheduling the rolling stock and evaluating the resulting passenger behavior which is inspired by the iterative framework of Dumas and Soumis [5]. Changing the timetable can also improve the passenger service. Therefore we extend the approach of Kroon et al. [8] by allowing the timetable to be slightly adapted as well.

It is important to focus on the passenger service since a disruption does not only affect the timetable and the resource schedules, but also the passengers. However, for railway operators without a seat reservation system it is difficult to reschedule the passengers. The passengers will make their new travel plan by themselves. If they had planned to take a train which is canceled due to the disruption, they will decide not to travel or to reroute themselves. Rerouting of passengers means that they take other trains to their destination than originally planned. This does not necessarily require the passengers to take a detour: They can also take a later train on the same line.

By the changed passenger flows, the disruption causes changes in the demand for seats. Therefore, a rolling stock rescheduling approach to handle a disruption must take the modified passenger flows into account dynamically, and not the passenger flows of a regular day. For example, since some passengers will take a detour, additional capacity on the detour routes is necessary. One way to handle this is to increase the capacity of the trains on this route.

Another option is to increase the capacity by inserting more train services or by letting trains make additional stops.

The consequences of the timetable adaptations may not turn out to be advantageous for all passengers. For example, an additional stop of a train will delay the train with a few minutes. As a consequence, the original passengers of the train will get an additional delay in favor of reducing the delay of the passengers at the station at which an additional stop is made. The small delay of the train can even lead to a large delay for the passengers if they miss their transfer at a later station. The railway operator has to make trade-offs between the different consequences for the passengers.

In this research we limit the timetable decisions to adapting the stopping patterns. Other timetable decisions to influence the passenger flows, for example by inserting additional trains, are left out since an additional train requires the railway operator also to adapt the crew schedules. By adapting the stopping patterns only, the crew schedule remains feasible.

Delay management, which consist of deciding on whether or not trains have to wait for delayed connecting trains, is another problem in which slight timetable adaptations influence the passenger flows. Delay management is a hard problem on its own and thereby not considered in our approach. We refer the interested reader to Schachtebeck and Schöbel [10], Kanai et al. [7] and Dollevoet et al. [4] for recent work on delay management approaches.

The framework of our rolling stock rescheduling approach is discussed in Section 2. In Section 3 we show a relaxation of the model discussed in Section 2. Instead of solving the relaxation, we make use of an iterative procedure of which in Section 4 up to Section 7 the components are explained. Results of different variants of our approach, based on a scenario in the Netherlands are discussed in Section 8, and Section 9 concludes this paper.

2 Rolling stock and timetable rescheduling with dynamic passenger flows

The performance of the disruption management process investigated in this paper arises from the interaction of 3 factors: (i) the timetable, (ii) the rolling stock schedule (seat capacity), and (iii) the passenger behavior.

We consider disruptions where passenger behavior has a large impact on the performance of the railway system if the timetable and rolling stock schedule are not changed. Examples are disruptions where certain tracks are blocked for a number of hours. Passengers react to these disruptions by finding alternative routes to their destinations. However, the capacity on these alternative routes can be limited, resulting in overcrowded trains and thereby longer dwell times and delays.

Two ways to handle the increased demand on the alternative routes are to enlarge the capacity of the trains and to adapt the timetable. Adapting the capacity of the trains alone is not always enough. For example, it can be impossible to increase the capacity of a train by lack of time and/or reserve rolling stock or due to limited platform lengths. Therefore, timetable adaptations such as adding extra train services, rerouting trains or adding extra stops for trains are worthwhile as well.

By adapting the timetable, the railway operator can influence the passenger flows by providing new alternative travel routes and by influencing the demand for certain trains. For example, a train can make an additional stop at a station to give passengers at that station an additional, earlier, travel option to their destination and to decrease the demand for the next train calling at that station and travelling in the same direction.

In this research we limit the timetable adaptations to adaptations of the stopping patterns of trains. A stopping pattern of a train indicates the stations where the train makes a stop. A *stopping pattern* can contain, next to the scheduled stops, also new stops at stations where the train did not have a scheduled call.

To make a new rolling stock schedule and timetable for the remainder of the day, we assume that the complete characteristics of the disruption are revealed at the moment the disruption starts. For example, at that time, the exact duration of the disruption is known.

Then, a general framework for rescheduling the rolling stock and timetable by considering the passenger behavior based on the model of Kroon et al. [8], with the difference that now also decision variables for the timetable decisions are included, can be stated as follows:

$$\min c(x) + d(y) + e(z) \tag{1}$$

$$\text{subject to } z \in \mathcal{Z} \tag{2}$$

$$x \in \mathcal{X}_z \tag{3}$$

$$y = f(x, z) \in \mathcal{Y} \tag{4}$$

Here $\mathcal{Z}$ is the set of all possible timetables given the disruption, $\mathcal{X}_z$ is the set of all possible rolling stock schedules matching with timetable z, and $\mathcal{Y}$ is the set of feasible passenger flows. The function $f(x, z)$ returns the emerging passenger flow for a given timetable $z \in \mathcal{Z}$ and rolling stock schedule $x \in \mathcal{X}_z$. Note that the chosen timetable z and rolling stock schedule x uniquely determine the passenger flow y by the function f. This means that the only real decision variables are the rolling stock schedule x and the timetable z.

The objective function consists of three terms. The function $c(x)$ gives the system related costs of a rolling stock schedule, which can also be seen as the rolling stock rescheduling costs. The function $d(y)$ gives the service related costs of a passenger flow. The function $e(z)$ that gives the system related costs of a timetable, so the timetable rescheduling costs. The highest priority is given to assigning at least one rolling stock unit to each train, to prevent the train to be cancelled by lack of rolling stock. Such cancellations will not only have a large negative influence on the passenger flows, but also make the crew schedule infeasible.

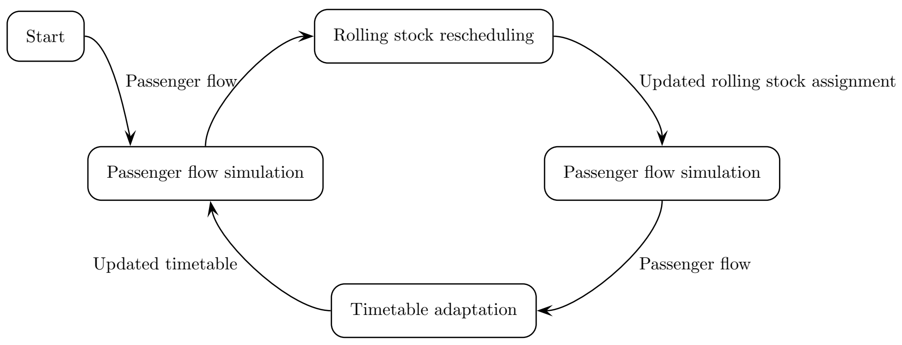

Start
Passenger flow
Rolling stock rescheduling
Updated rolling stock assignment
Passenger flow simulation
Passenger flow simulation
Updated timetable
Passenger flow
Timetable adaptation

Fig. 1 Iterative procedure for solving the rolling stock rescheduling problem with dynamic passenger flows.

2.1 Iterative Procedure

The optimization model (1)-(4) is very difficult to solve directly, mainly due to the complex structure of the objective function f. We are not aware of any algorithmic framework that would be able to handle realistic instances of (1)-(4). Therefore we propose an extension to the iterative heuristic of Dumas and Soumis [5] and Kroon et al. [8]; the approach is sketched in Figure 1.

The input of our algorithm consists of the original (i.e., undisrupted) timetable, the original rolling stock schedule as well as a list of train services that must be cancelled as an immediate reaction to the disruption. The removal of these inevitably cancelled services gives the initially modified timetable.

In each iteration, we evaluate the passenger flows by using a simulation algorithm. The simulation is based on the previous iteration's timetable and rolling stock schedule. Here the rolling stock schedule is only needed because it determines the capacities of the trains. We use the simulation model introduced by Kroon et al. [8]. The details of this simulation model are summarized in Section 4. Note that the first iteration uses the initially modified timetable, and assumes that each train has the same capacity as in the original schedule.

The passenger simulation pinpoints the trains with insufficient capacities. The rolling stock rescheduling model computes a new schedule based on these findings, balancing it with other criteria, such as operational costs. For details we refer to Section 5.

After another round of passenger simulation, we evaluate which adaptations of the timetable could potentially improve the service quality. Each individual adaptation is a minor change, such as requiring a train to make an extra stop. Therefore we can assume that the just computed rolling stock schedule remains feasible. We describe several variants for finding the most promising timetable adaptation in Section 6. Having decided on the timetable, the next iteration will start by launching a passenger simulation.

Our method differs from the framework of Kroon et al. [8] by adding the timetable adaptation step to the loop. Since the passenger flows can be heavily impacted both by a new rolling stock schedule and by an adapted timetable, we carry out passenger simulations after each of them.

The iterative approach is purely heuristic; it does not necessarily converge, and has no optimality guarantee. Motivated by the limited time in real-life applications, we terminate our algorithm after a certain number of iterations, and we report the best solution found. In addition, we compute lower bounds, described in Section 7, in order to be able to judge the quality of the solution.

3 Operator control

The rescheduling process may result in a better outcome if the operator can directly influence the passengers' behavior by appropriately assigning them to the train services (rather than letting the passengers choose their routes). We call this situation *operator control.* In this section we describe an optimization model for operator control which is a relaxation of the model (1)-(4). We are not going to use this model in our computational tests, since it is a computationally large model and we do not want to assume operator control. However, we still want to present this relaxation of the model to give an idea about its complexity, and thereby justifying our use of an iterative procedure.

We split all timetable services into trips $t \in \mathcal{T}$ representing a movement of a train between two consecutive planned stops. The main decision for the rolling stock schedule is to assign *compositions* to trips, where a composition consists of one or more combined train units. Let $\mathcal{G}_t$ be the set of all compositions g which can be assigned to trip t, and the capacity of composition g is denoted by Cap_g. Binary variables $x_{t,g}$ indicate whether composition g is used ($x_{t,g} = 1$) for trip t or not ($x_{t,g} = 0$).

For the timetable decisions every trip $t \in \mathcal{T}$ has a set $\mathcal{J}_t$ of possible stopping patterns for calls at the intermediate stations. Here a stopping pattern indicates a sequence of intermediate stations at which the train makes an additional stop. Binary variables z_j indicate whether stopping pattern j is used ($z_j = 1$) or not ($z_j = 0$).

A passenger $p \in \mathcal{P}$ should take a path from its origin to its destination within his/her proposed deadline, where a path itself is a sequence of rides on trains between two stations. Let K^p be the set of all paths that passenger $p \in \mathcal{P}$ could take and let $K_t^p \subset K^p$ be all paths which passenger p could take with (part of) trip t in it. Note that the paths in K^p and K_t^p can be based on every possible stopping pattern. Let $\bar{\mathcal{J}}_k$ be the set of all stopping patterns j matching with path k. The binary variable y_p^k is 1 if passenger p picks path k and 0 otherwise. The parameter d_p^k indicates the associated cost (delay) of passenger p taking path k.

Then, in case of operator control, the model of (1)-(4) can be relaxed by:

$$\min \quad c(x) + \sum_{p\in\mathcal{P}} \sum_{k\in K^p} y_k^p d_k^p + e(z) \tag{5}$$

$$
\begin{aligned}
s.t. \quad & x \in \bar{\mathcal{X}}_z & & (6)\\
& \sum_{j\in\mathcal{J}_t} z_j = 1 & \forall t \in \mathcal{T} & \quad (7)\\
& \sum_{k\in K^p} y_k^p = 1 & \forall p \in \mathcal{P} & \quad (8)\\
& y_k^p - z_j \le 0 & \forall p \in \mathcal{P}, \forall k \in K^p \text{ and } \forall j \in \bar{\mathcal{J}}_k & \quad (9)\\
& \sum_{p\in\mathcal{P}} \sum_{k\in K_t^p} y_k^p \le \sum_{g\in\mathcal{G}_t} x_{t,g} Cap_g & \forall t \in \mathcal{T} & \quad (10)\\
& y_k^p \in \{0,1\} & \forall p \in \mathcal{P} \text{ and } \forall k \in K^p & \quad (11)\\
& z_j \in \{0,1\} & \forall t \in \mathcal{T} \text{ and } \forall j \in \mathcal{J}_t & \quad (12)
\end{aligned}
$$

The objective function (5) is to minimize the total costs of the rolling stock rescheduling, the passenger flows (sum of delays) and the timetable rescheduling. Constraints (6) compactly summarize the constraints on the underlying rolling stock rescheduling problem. These rolling stock decisions are influenced by the chosen timetable z since there are some minimum process times required in the rolling stock schedule. So, if some trips take longer than planned certain processes can become infeasible. Constraints (7) determine that for every trip exactly one stopping pattern must be selected. Every passenger must pick exactly one path, which is modeled by Constraints (8). Constraints (9) ensure that only matching paths and stopping patterns can be chosen. The chosen paths by the passengers should also match with the available capacity in the trains which is modeled by Constraints (10).

Even this relaxation of the model of (1)-(4) is a complex model to solve in a real-time environment by the interdependence between the rolling stock and the timetable via the passenger flows. Therefore, we will solve the model in (1)-(4) by an iterative procedure as discussed in Section 2.1.

4 Passenger flow simulation

The iterative procedure starts with a simulation of the passenger flows, and each time the timetable or rolling stock schedule is updated a new simulation of the passenger flow is necessary.

To keep the simulation tractable, all passengers with the same characteristics (origin, destination and arrival time at the origin) are aggregated into *passenger groups*.

To simulate the passenger flows we use the simulation algorithm as described in Kroon et al. [8]. It is important to mention that this is a deterministic simulation algorithm to calculate the emerging passenger flows. This means that, given a timetable and rolling stock schedule, there is a uniquely defined resulting passenger flow. Here we shortly summarize the assumptions of the model as described in Kroon et al. [8]. For more details we refer to that paper. We emphasize that the approach is modular, which allows us to replace the simulation model by any other simulation model to model the passenger behavior.

4.1 Assumptions

For the simulation of passenger behavior, Kroon et al. [8] make assumptions on three fundamental issues: (*i*) What information is available to the passengers? (*ii*) Which traveling strategy do passengers apply to the available information? and (*iii*) How do passengers interact?

Information available for the passengers

It is assumed that passengers always know the most recent timetable. This means that if the timetable is updated due to a disruption, they know which trains are canceled, which trains make additional stops, and which trains are delayed. Passengers do not know the future timetables, so they cannot anticipate on cancellations, delays and additional stops before the disruption occurs. Furthermore, they do not know anything about whether or not they fit in the trains they would like to take.

Strategy of the passengers

Each passenger has a *traveling strategy*. This strategy decides for the passenger what will be the preferred path to the destination given the most recent timetable. In Kroon et al. [8] all passengers have the same strategy. In our research we also use this single strategy. The used strategy is that passengers want to reach their destination as early as possible. If several paths have the same earliest arrival time, the passengers prefer the path with the least transfers between trains. If we have multiple paths with the same earliest arrival time and the same minimum number of transfers, the passengers will take the path with the earliest departure time. It is worthwhile to mention that in practice there is a more balanced trade off between transfers and travel time. It seems to be highly unrealistic that passengers are willing to transfer 2 times to save 1 minute of travel time. Note that one could easily include other strategies as well.

Each passenger wants to reach his destination before a certain *deadline*. If a passenger is not able to reach his destination before the deadline, it is assumed that the passenger gives up travelling by train. In this way it is modeled that passengers are not willing to wait endlessly.

Interaction between the passengers

If a train arrives, first passengers who want to leave the train get the option to do so, then the passengers waiting at the platform who want to enter the train compete for the available capacity in the train. It can happen that there is not enough capacity for all passengers. Then it is assumed that the number of passengers from each passenger group who actually board a train is relative to the size of the group. This could lead to a fractional number of passengers but it is assumed that the contribution of fractional flows are neglectable.

It is possible that not all members of a passenger group are able to board the train: Some of them have to stay behind. We say that these passengers are *rejected* by the train. In case of rejections, the passenger group is split into two: those passenger who were able to board the train and those who were not. The rejected passengers must find a new preferred path from their current location, while the boarded passengers can just follow their previously computed preferred path.

4.2 Evaluating the passenger flow

In this research we evaluate the passenger flows by the delays which passengers face in comparison with their original expected arrival times and by the number of passengers who did not reach their destination within their set deadline. In our experiments we try different ways to penalize delay minutes. In one setting the delay minutes are penalized uniformly and in another setting longer delays are penalized more, since one may argue that longer delays are worse than several small delays. For passengers who are not able to reach their destination within their deadline we penalize passengers leaving the system by the difference between the deadline and the expected arrival time of the intended traveling path. For each passenger, his delay or penalty for not reaching his destination within his deadline is called his *inconvenience.*

5 Rolling stock rescheduling

The rescheduling of rolling stock follows the procedure of Kroon et al. [8]. In this procedure the rolling stock is rescheduled based on the model described in Nielsen et al. [9] (which is an extension of Fioole et al. [6]). The basic decisions in the model are to assign a rolling stock composition to each trip such that as many of the passengers are accommodated. The difficulty of the rolling stock rescheduling is that a composition consists of multiple combined train units.

During operations the operator can change the compositions by decoupling or coupling units in the front or the back of the train. These operations are called *shunting operations.* Shunting personnel must be arranged to facilitate these operations. Therefore, changing the shunting operations also includes new tasks for the shunting personnel, which is not preferred.

Since a composition can consist of different types of train units, the order in which they are combined within a composition matters (i.e. one could not decouple a unit in the middle).

As discussed, the main objective of our approach is to prevent cancellations caused by lack of rolling stock. Therefore we once determine how many trains need to be cancelled due to lack of rollings stock. To do this we run the rolling stock rescheduling approach on the initially modified timetable with the single goal to find a feasible rolling stock schedule by minimizing the number of trains without rolling stock. This means that we have only a penalty for trains which do no get rolling stock assigned to them. All other penalties are set equal to 0. The result shows how many trains need to be cancelled inevitably by lack of rolling stock. In the rolling stock rescheduling steps we enforce the number of cancelled trains to be equal to this to ensure that no more trains than necessary are cancelled. Still the rolling stock rescheduling approach has freedom in which trains it does not assign rolling stock to.

For all remaining rolling stock rescheduling steps, the model has two objectives: It consists of a trade of between minimizing the rolling stock rescheduling costs and the inconvenience for the passengers. The rolling stock rescheduling costs are mostly based on how much the rolling stock schedule is changed. For example one does not want to make too many new shunting operations, since these new shunting operations must be communicated (with a certain probability of miscommunication) and it requires personnel to perform them.

The inconvenience for passengers is based on the latest simulation run with the timetable and rolling stock schedule of the last iteration. Penalties are defined for assigning a certain composition to a trip. The penalties are determined by estimating the effect of the train capacities on the total passenger inconvenience measured as discussed in Section 4.2.

Per trip the average inconvenience per passenger who was not able to board the train is computed. To do this, per passenger group the difference in inconvenience between passengers who were not able to board and passengers who were able to board is determined. Then, the weighted (based on group size) average of these differences is considered as the average inconvenience per passenger who is not able to board the train. In the objective function the number of seat shortages is multiplied with this average inconvenience per rejected passenger. For more details we refer to Kroon et al. [8].

Kroon et al. [8] reported that the approach of updating the objective function could lead to cyclical behavior if the feedback from earlier iterations is ignored. We follow the described exponential smoothing procedure in Kroon et al. [8] (which is based on Dumas and Soumis [5]) to take feedback from earlier iterations into account as well. We use the setting which performed best in their case. This setting means that feedback from earlier iterations is weighted for 35 percent.

6 Timetable adaptations

The disruption management process admits timetable decisions in order to better facilitate the passenger flows. In this paper we limit the allowed timetable modifications to adding stops to timetable services.

In this paper, adapting the stopping patterns means that trains may call at stations where they normally just pass through. Making an additional stop results in new traveling options for some passengers but also in an increased travel time for others. Therefore it is necessary to make a trade off between the positive and negative effects of the changed stopping pattern. The objective of this research is to minimize the sum of the delays of all passengers. Therefore, we only allow timetable changes that do not increase the total delay of all passengers. We assume that an additional stop will delay all further trips of a train by a fixed number of minutes and that those delays will not influence other train traffic.

The (greedy) procedure to adapt the stopping patterns goes as follows: (i) we have a list of candidate timetable adaptations, (ii) we evaluate for each candidate the consequences, (iii) we apply the timetable adaptation with the most positive consequences.

First of all, this approach requires that in step (i) a list of candidate timetable adaptations is given. The dispatchers can give this as input to the approach. In the extreme case, every timetable service is allowed to make an additional stop at every station it passes.

The effect measured in step (ii) indicates how much the total delay of the passenger will change if only that single timetable adaptation will be applied. Therefore, in step (iii) we limit ourselves to allow only one timetable adaptation per iteration of the solution approach. If no candidate timetable adaptation reduces the total delay of the passengers, no timetable change is made.

For all candidate timetable adaptations, the consequences of applying the adaptation needs to be computed. In Sections 6.1-6.4 we discuss several methods and approximations to compute these consequences. In Section 6.1 we discuss a method to compute the exact effect of the additional stop. The exact effect can be computed since we use a deterministic passenger flow simulation. However, computing the exact effect can be time consuming. Therefore, we also suggest a faster approximation algorithm in Section 6.2. Furthermore, we introduce two heuristics in Sections 6.3 and 6.4 which are more transparent for use in practice.

In the different variants we evaluate the effects of different candidate timetable adaptations. We refer to train i as the train of which the stopping pattern an be adapated within the candidate timetable adaptation. Furthermore, the station at which train i will make the additional stop within the candidate timetable adaptation is called station b.

6.1 Exact effect of an additional stop (*EXACT*)

To determine the exact consequences of an additional stop, we need to run the simulation algorithm which is discussed in Section 4 twice. First the simulation is performed with the current timetable and then the simulation is performed with the timetable which results from the timetable adaptation. From both simulations we get the total delay minutes of the passengers. The difference between these two total delay minutes shows the consequences of the candidate timetable adaptation. This approach measures the exact effect of the additional stop and is called *EXACT*.

A variant of *EXACT*, denoted by *EXACT**, will not use the current capacity of train i in the simulation but the capacity of the largest possible composition allowed for train i. The difference between this simulation and the simulation of the current timetable will then not measure the exact effect of the extra stop but the potential (which can be larger) effect of the additional stop. In this case it is left to the rolling stock rescheduling phase to check whether it is possible to increase the capacity of train i.

6.2 Estimated effect of an additional stop (*EST*)

In this section we introduce an approach to *estimate* the effect of an additional stop. In this variant, both the positive and the negative effects of the additional stop are considered. For all passengers their preferred path to their destination is known. If we change the timetable by including an additional stop, some passengers might get another preferred path to their destination.

Some passengers arrive earlier at their destination due to an additional stop at their origin or destination. Other passengers might profit from the delay of the train caused by the additional stop, since due to the delay they were able to catch this train which departs earlier than the train they were intending to take. All these passengers originally did not have train i on their preferred path, but in case the additional stop is executed they have train i on their preferred path.

However, since the additional stop takes some time it also causes some delays for passengers who had train i on their preferred path in the current timetable. Due to the delay of train i it can be that their preferred path changes. Also if the preferred path does not change it can still mean that the passengers are delayed if the trip on train i was the final trip in their path.

To estimate the effect of the additional stop we compute for each passenger his preferred path in the current timetable and the preferred path in the timetable in which train i makes an additional stop. The sum of all these differences is our estimate of the consequences of the additional stop. Note that this is an estimate since this method assumes that every passenger can take his preferred path which might not be true by the capacity of the rolling stock.

We make variants of this approach by assuming different durations of the additional stop in the determination of the shortest path. This means that we can use for the estimation of the effect another duration of the additional stop than the real duration of the additional stop. For example, if we assume a shorter duration of the additional stop, we over-estimate the potential effect of the additional stop. However, maybe this will lead to a good optimization direction. The duration of the extra stop used in the estimation approach will be called the *extra stop penalty.*

6.3 Rule of thumb: Do not pass passengers who did not fit in a previous train (*PRACT1*)

If passengers did not fit in a train, then they have to wait for the next train in the same direction. Especially for these passengers, since they were already rejected, it is very frustrating if a train in their direction passes them without stopping. A rule in practice could be that it is not allowed to pass a group of passengers who were rejected by a previous train. This is an easy to use rule of thumb. We will evaluate the performance of this rule which is called $PRACT1$.

To decide whether or not train i needs to make an additional stop we have to consider two other trains. The first considered train h is the last train which arrived at station b before the passing time of train i at station b. If there were passengers with destination b rejected to board train h at the departure from the last station a before station b, train i will make an additional stop at station b to let these rejected passengers travel from station a to b.

The second considered train j (which is in most cases the same as train h), is the last train which departed from station b in the same direction as train i before train i passes station b. If there were passengers rejected to board train j at station b, train i will make an additional stop at station b to let these rejected passengers in.

Since we allow one timetable adaptation per iteration we have to make a comparison on how effective the additional stop will be: Therefore, we sum up the advantages for all passengers who were rejected to board train h in station a or train j in station b. For the passengers rejected to board train h in station a the advantage is measured by the difference between the arrival time of train i at station b and the arrival time of the first train from a to b after train i. For the passengers rejected to board train j in station b the advantage is estimated by the arrival time of train i at the first station after station b where both trains h and i call and the arrival time at the same station of the first train departing after train i from station b.

In this measurement we assume that all passengers are able to board train i, so that train i is assumed to have infinite capacity. We do not use the actual capacity since in the rolling stock rescheduling phase the capacity of train i could be increased.

6.4 Rule of thumb: including the negative effects *PRACT2*

The approach *PRACT1* based on a rule of thumb only considers the positive effects of an additional stop. This results in a situation that even if only one passenger may profit from the additional stop, the stop will be executed. In another rule of thumb, *PRACT2*, the delay for passengers traveling by train i caused by the additional stop is included. In this approach, the advantages for passengers are measured in the same way as approach *PRACT1*, and the inconvenience per passenger who travels by train i at the moment train i makes an additional stop at station b will be equal to a fixed parameter. This parameter is equal to the duration of the extra stop plus an eventual penalty. Note that for practitioners this rule requires more knowledge. In *PRACT1* the dispatchers only need monitor whether there are trains where some passengers did not fit in the train. In *PRACT2* the dispatchers also need to know how many passengers did not fit in the train, and how many passengers are in the next train passing station b.

7 Lower bound

The proposed approach does not guarantee to converge to an optimal solution. To consider the quality of our solutions, we check the gap between a lower bound and the value of our solution. Depending on the nature of the disruption, the lower bound on the rolling stock rescheduling costs will not differ that much from 0, but the lower bound on the passenger delays can be quite interesting. In this section we come up with a lower bound which takes the positive effects of an additional stop into account.

This lower bound can be reached by assuming infinite capacity on all trains, together with assuming that all extra stops are executed and that an extra stop does not cause any arrival delay.

To be more precise, in this lower bound all extra stops are executed and the departure times at stations after the additional stop are delayed by the time an extra stop will take, and all arrival times are kept the same. This way we ensure that no passenger faces an arrival delay caused by the additional stop and that passengers who may profit from a delayed train caused by an additional stop still have the opportunity to enter the train. Then a simulation run with infinite capacity on the trains and with the timetable as described above gives a lower bound on the total passenger delay.

It can happen that passengers have an advantage by a delayed train since they can pick a train earlier than their planned train. This must be considered in the lower bound. Therefore we cannot just add the extra stops and leave all departure and arrival times the same.

By delaying the departure times we have a lower bound which is valid for both cases, with and without the extra stop. No one gets an arrival delay, and some passengers arrive earlier since they have an extra travel opportunity by the extra stop.

This lower bound represents the delay of the passengers which the operator cannot prevent by increasing the rolling stock capacities or adapting the timetable. This delay is caused by the train services that are inevitably cancelled due to the unavailability of infrastructure caused by the disruption.

8 Computational Results

We tested the proposed approach on instances based on cases of Netherlands Railways (NS) which is the major railway operator in the Netherlands. In these instances, a disruption, due to some blocked switches, caused that fewer trains than normally can be operated on certain tracks.

8.1 Detailed case description

The instances take the busiest part the Dutch railway network into account which is represented in Figure 2. The original passenger flows are constructed conform a regular weekday of Netherlands Railways, which resulted in 15064 passenger groups with a total of about 450,000 passengers.

In almost each of the trajectories we have four Intercity trains per hour in each direction. A *Intercity train* is a train which only calls at the larger stations. All intercity trains in this network are considered, furthermore the *regional trains*, that stop at every station, between The Hague (Gv) and Utrecht (Ut) are also considered.

For the rolling stock rescheduling, four types of rolling stock are available; two types for regional trains and two types for intercity trains. The regional train types can be coupled together, which leads to 5 possible compositions, and the Intercity train types can also be coupled together in 10 different compositions.

In Figure 2 the dotted line represents the disrupted area. On those tracks on a normal day each hour 4 Intercity trains and 4 regional trains run in each direction. We constructed two instances with a disruption in the rush hours between 7:00 A.M. and 10:00 A.M. In the first instance (*ZTM1*), 2 regional trains per hour per direction are canceled. In the second instance (*ZTM2*) also 2 Intercity trains per hour per direction are canceled. This means that in instance *ZTM1* in each direction 6 trains per hour still run between Gouda (Gd) and The Hague (Gv), and only 4 trains per hour in each direction in instance *ZTM2*.

8.2 Parameter settings

The objective function consists of system related costs for the timetable adaptation and the rolling stock rescheduling, and costs for the passenger delays. For the timetable adaptations we do not consider any penalties other than that we assume that an additional stop will delay a train by 3 minutes.

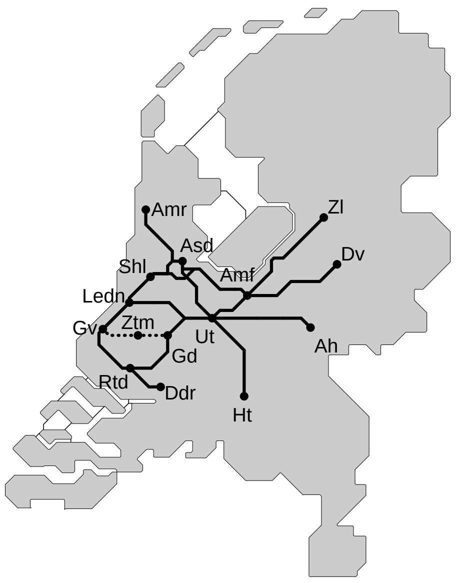


Fig. 2 Part of the Dutch railway network

Table 1 Rolling stock rescheduling costs

Type of costs	value
New shunting operation	500
Changed shunting operation	500
Canceled shunting operation	100
Off balances at the end of the day, per unit	200
Seat shortage per seat per kilometer	0.1
Carriage Kilometers	0.0001

The rolling stock rescheduling costs are given in Table 1. Most important is that the rolling stock schedule should not change too much from the original plan, since changed plans require communication between the dispatchers and the personnel, and a failure in this communication is easily made. Therefore we introduce costs for having other shunting operations than planned. Changing the shunting operations also includes new tasks for the shunting personnel, which is not preferred. We consider the carriage kilometers as least important.

The passenger service costs consist of the passenger delay minutes as discussed in Section 4.2, where the penalties for passengers who left the system because of not reaching their end station within their deadline are also measured in delay minutes.

The approach will make at maximum 15 iterations.

To solve the composition model of the rolling stock rescheduling we used CPLEX 12.5. The test instances are run on a laptop with a Intel(R) Core(TM) i7-3517U 1.9/2.4 Ghz and 4.0 GB RAM.

Table 2 Results

Solution Method	Lower bound	Objective	Passenger delay minutes	Rolling stock rescheduling costs	Extra stops	Iteration of best solution	Computation time (sec)
Instance *ZTM1*							
(EXACT)	33526	55927	55874	53	4	4	563
(EXACT*)	33526	55927	55874	53	4	9	573
(EST 0min)	33526	57372	56818	554	3	3	349
(EST 1min)	33526	58857	58304	554	4	4	352
(EST 2min)	33526	57372	57318	53	5	9	352
(EST 3min)	33526	91534	90980	554	0	1	291
(PRACT1)	33526	64304	64251	53	3	4	282
(PRACT2)	33526	64304	64251	53	3	4	307
(NO_STOP)	49626	91534	90980	554	-	1	235
Instance *ZTM2*							
(EXACT)	110848	139588	136527	3061	3	3	456
(EXACT*)	110848	139588	136527	3061	3	8	427
(EST 0min)	110848	139630	136368	3262	3	4	352
(EST 1min)	110848	139630	136368	3262	3	4	315
(EST 2min)	110848	162228	159167	3061	2	6	351
(EST 3min)	110848	177123	173861	3262	0	1	291
(PRACT1)	110848	152062	149000	3061	4	4	320
(PRACT2)	110848	166356	163295	3061	3	7	326
(NO_STOP)	120373	177123	173861	3262	-	1	249

8.3 Results

This section provides the results of the two test instances. For the timetable rescheduling part we had different approaches to decide which Intercity trains should make an additional stop. We compare the effect of the different approaches on the final solution. We also compare our approach (which includes the option to adapt the timetable) with the method of Kroon et al. [8] (which does not have an option to adapt the timetable) referred to as (*NO_STOP*). In the approach (*EST*) we estimated in the timetable rescheduling step the effect of an additional stop. Within this estimation we discussed that we could assume different lengths of the additional stops. This assumed length of the additional stop is also called the extra stop penalty. In our experiments we used 0, 1, 2 and 3 minutes for the extra stop penalty. Note that an extra stop penalty of 0 minutes means that it is assumed that nobody faces negative effects of the additional stop. Furthermore, note that the realized timetable adaptation always includes a 3 minute delay caused by the additional stop.

In Table 2 we provide the best result found in the iterative procedure for each of the variants of the approach. Note that the iterative procedure does not necessarily converge to an optimal solution and thus the best solution can be found at any iteration. Therefore, we included the number of the iteration where the best solution was found. For each approach the *lower bound* as discussed in Section 7 is given. Furthermore, the table contains the value of the *objective* function which consist of the sum of the *rolling stock rescheduling costs* (by considering the parameters in Table 1) and the passenger inconvenience (measured in *passenger delay minutes* as discussed in 4.2). We also report how many *extra stops* are included in the timetable of the best result. The *computation time* is measured in seconds and reports the computation time over all 15 iterations, and not just the computation time up to the moment the best solution is found. The latter would not be fair, since beforehand it is not known at which iteration the best solution will be found.

Performance

In the approach *NO_STOP* based on Kroon et al. [8], timetable adaptations were not allowed. Our results show that allowing the stopping patterns to be adapted can reduce the passenger delays dramatically by about 25 to 35 percent.

From Table 2 we can deduce that the approach *EXACT* led in both cases to the lowest passenger delay minutes and the lowest rolling stock rescheduling costs. *EXACT**, the variant of the approach *EXACT*, reaches the same solutions, but it takes longer to get there. The estimation approach works well as long as we overestimate the positive effects of the additional stop by having a lower extra stop penalty (0 or 1 min) than the realized delay (3 min).

The performance of the approach *EST 0min* is surprising. It underestimates the negative effects and overestimates the positive effects of the additional stop but it is still able to reach solutions which do not differ much from the solutions reached by the approach *EXACT*. In deciding on which train should make an additional stop, the approach *EST 0min* assumes that an additional stop does not cause any delay and thereby no one faces negative effects of the additional stop. In every iteration an additional stop is introduced (by assuming that every additional stop has at least some positive effect).

On the other hand, the bad performance of the approach *EST 3min* is also surprising. Especially since in this approach the duration of the additional stop in the estimation approach matches the realized duration of an additional stop. However, this approach finds it never worthwhile to make an additional stop. Since this approach does not consider the capacities of the trains, it does not take rejected passengers into account. The *EST* approaches, thereby underestimate the positive effect the additional stop could have for rejected passengers. It seems that in *EST 0min* and *EST 1min* this underestimation is balanced by the overestimation of the other positive effects, but in the *EST 3min* approach the underestimation is not corrected by another overestimation.

The rules of thumb approaches *PRACT1* and *PRACT2* are outperformed by our exact approach *EXACT* and by our estimation approaches *EST 0min* and *EST 1min*. This shows that our more complex approaches are able to come to better solutions.

Iterative behavior

In Figures 3 - 8 we show for six of the variants how the solution of the case *ZTM1* changes over the iterations. The black dot indicates the first solution

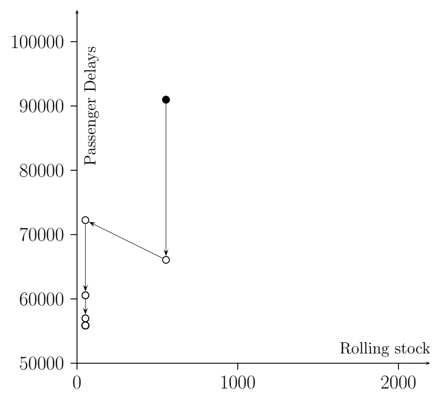


Fig. 3 *EXACT* case *ZTM1*

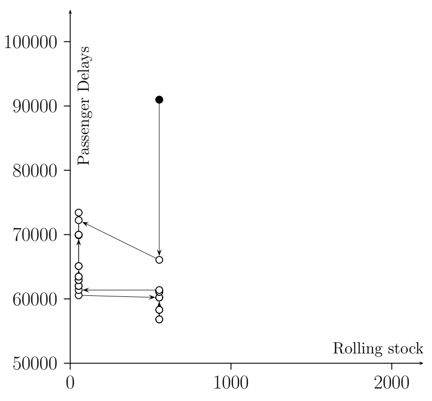


Fig. 4 *EST 0min* case *ZTM1*

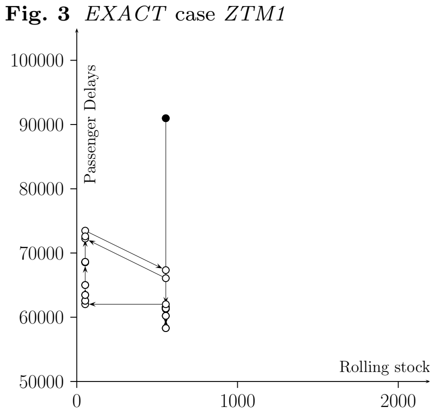


Fig. 5 *EST 1min* case *ZTM1*

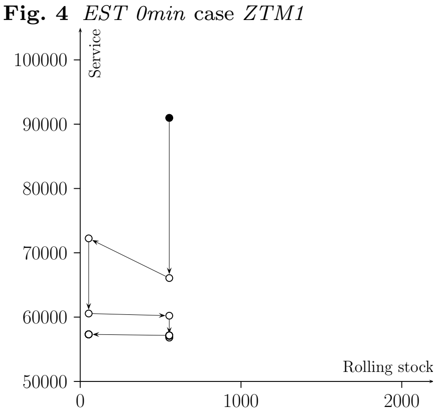


Fig. 6 *EST 2min* case *ZTM1*

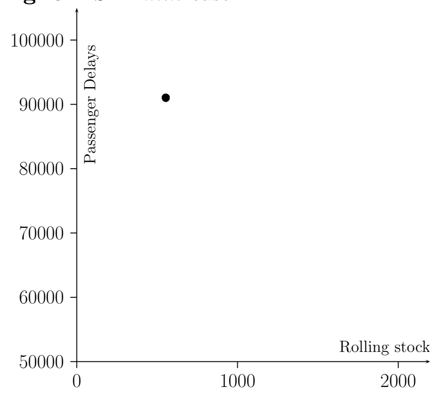


Fig. 7 *EST 3min* case *ZTM1*

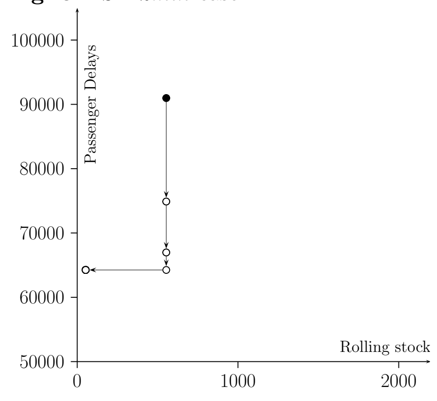


Fig. 8 *PRACT1* case *ZTM1*

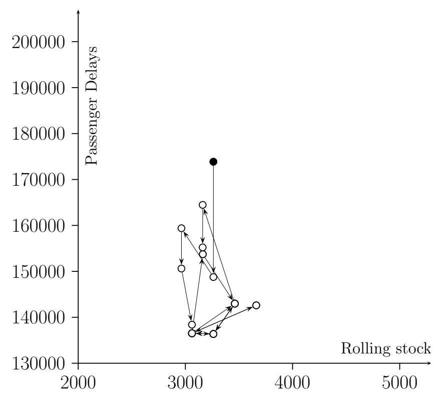


Fig. 9 *EXACT* case *ZTM2*

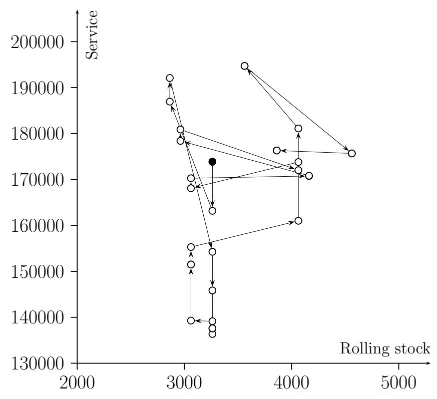


Fig. 10 *EST 0min* case *ZTM2*

and by arrows we indicate how the solution evolves. On the horizontal axis we have the rolling stock rescheduling costs and on the vertical axis we have the passenger delays. With the rolling stock rescheduling both the rolling stock rescheduling costs and the passenger delays can change. However a timetable adaptation only influences the passenger delays, and therefore, a vertical drop or increase in the figure can most of the time be associated with a timetable adaptation.

The *EXACT* approach has a quite clear converging path to its best solution by decreasing passenger delays and rolling stock rescheduling costs. The solution of the approaches *EST 0min* and *EST 1min* first goes to solutions with low passenger delays and low rolling stock rescheduling costs, but from a certain moment, the passenger delays are increasing again. The approaches *EST 2min* and *PRACT1* converge like the *EXACT* approach to their best solution, but especially the solution of *PRACT1* does not come close to the solution of *EXACT*. The approach *EST 3min* has in every iteration the same solution.

In Figures 9 and 10 the iterative behavior of our best performing variants (*EXACT* and *EST 0min*) on case *ZTM2* are given. All variants did not show converging behavior for this case. The figures demonstrate that the *EXACT* approach explores a smaller region of solutions. The approach *EST 0min* first goes to solutions with large passenger delays, then gets to solutions with low passenger delays and in the end it goes again into the direction of solutions with high passenger delays.

Computation time

Our approach added a module which adapts the timetable to the iterative procedure of the approach *NO_STOP* of Kroon et al. [8]. This means that we assume that by the additional computations our approach cannot be faster than the *NO_STOP* approach.

Table 3 Results with passenger flow costs times 10

Solution Method	Lower bound	Objective	Passenger delay minutes	Rolling stock rescheduling costs	Extra stops	Iteration of best solution	Computation time (sec)
Instance *ZTM1*							
(EXACT)	335260	558793	55874	53	4	7	568
(EXACT*)	335260	558793	55874	53	4	14	541
(EST 0min)	335260	568737	56818	554	3	3	346
(EST 1min)	335260	583589	58304	554	4	4	351
(EST 2min)	335260	568737	56818	554	3	6	350
(EST 3min)	335260	910355	90980	554	0	1	363
(PRACT1)	335260	642566	64251	53	3	7	297
(PRACT2)	335260	642566	64251	53	3	7	296
(NO_STOP)	496260	910355	90980	554	-	1	210
Instance *ZTM2*							
(EXACT)	1108480	1363686	135920	4064	3	11	431
(EXACT*)	1108480	1363686	135920	4064	3	5	430
(EST 0min)	1108480	1375949	137189	3262	4	4	346
(EST 1min)	1108480	1366944	136368	3262	3	4	298
(EST 2min)	1108480	1593181	158992	3262	2	4	331
(EST 3min)	1108480	1741873	173861	3262	0	1	359
(PRACT1)	1108480	1473060	146900	4064	2	2	313
(PRACT2)	1108480	1628519	162446	4064	3	12	293
(NO_STOP)	1203730	1741873	173861	3262	-	1	237

If we use the *EXACT* approach, an instance is solved in about double the time of the *NO_STOP* approach. The other variants of the approach solve the instances faster (within 6 minutes).

The first rolling stock rescheduling step, to determine the number of trains without rolling stock, is carried out in about 80 seconds. Then, next rolling stock rescheduling steps take 4 to 5 seconds per iteration. The timetable rescheduling phase takes 8 to 15 seconds per iteration within the *EXACT* approach, since multiple simulations must be carried out. The computation time of the timetable rescheduling phase drops to 1 to 6 seconds per iteration for the estimation approaches *EST* and to less than 1 second for the approaches *PRACT1* and *PRACT2*.

8.4 Additional tests

To see what happens with the solutions if we put more weight on the passenger delays, we run all approaches also with an objective in which the costs of the passenger flow is multiplied by 10. The results are presented in Table 3. The results and the performance are almost similar to the results in Table 2. In one third of the cases the rolling stock rescheduling costs are slightly higher to reach lower passenger delays.

In a third test we experiment on how the approaches behave if we give additional penalties to longer delays. In these tests we penalize delays between 15 and 30 minutes with an additional 5 minutes delay and delays longer than 30 minutes with an additional 10 minutes delay. Again one can see from the

Table 4 Results with 5 minutes additional penalty for delays larger than 15 minutes and 10 minutes additional penalty for delays larger than 30 minutes

Solution Method	Lower bound	Objective	Passenger delay minutes	Rolling stock rescheduling costs	Extra stops	Iteration of best solution	Computation time (sec)
Instance *ZTM1*							
(EXACT)	36171	59342	55874	53	4	8	572
(EXACT*)	36171	59342	55874	53	4	8	563
(EST 0min)	36171	60777	56818	554	3	3	341
(EST 1min)	36171	62297	58304	554	4	4	348
(EST 2min)	36171	60777	56818	554	3	3	361
(EST 3min)	36171	94969	90980	554	0	1	360
(PRACT1)	36171	67684	64251	53	3	8	290
(PRACT2)	36171	67683	64251	53	3	8	273
(NO_STOP)	52741	94969	90980	554	-	1	214
Instance *ZTM2*							
(EXACT)	114988	144513	136527	3061	3	5	417
(EXACT*)	114988	144513	136527	3061	3	5	492
(EST 0min)	114988	144540	136368	3262	3	4	312
(EST 1min)	114988	144540	136368	3262	3	4	331
(EST 2min)	114988	167383	159167	3061	2	5	344
(EST 3min)	114988	182073	173861	3262	0	1	339
(PRACT1)	114988	155026	146930	3061	4	4	324
(PRACT2)	114988	167639	159563	3061	2	5	309
(NO_STOP)	124813	182073	173861	3262	-	1	201

results in Table 4 that these additional penalties do not influence the solutions. In more than half of the cases, the best solution found is the same as in the situation without these additional penalties (as presented in Table 2). For the other cases the differences were not large.

These two additional tests show that our approach is not sensitive to changes in the evaluation of the passenger inconvenience.

9 Conclusions and further research

In this paper we proposed a disruption management approach which integrates the rescheduling of rolling stock and the adaptation of stopping patterns with the aim of improving passenger service.

Computational tests are performed on realistic large scale instances of the Dutch railway network. The two tested instances show that allowing the timetable to be adapted can reduce the total delay of passengers by more than 20 percent without increasing the rolling stock rescheduling costs. We suggested several variants of the approach, with the difference lying in the way of how the timetable changes are evaluated. These variants lead to different results and different computation times, but the results per variant are not quite sensitive to the exact cost parameter settings.

Our solution approach does not necessarily converge to an optimal solution. The lower bounds indicate that the gap between the solution and the lower bound is decreased by allowing stopping pattern adaptations. However the gap is still significant, which is probably caused by the weak lower bound. This is a topic for future research.

In future research we will incorporate other timetable decisions as well, for example reroutings of trains. Furthermore we want to proceed with integrating delay management decisions into the model, which will be quite challenging since the delay management approach is already a difficult problem to solve on its own.

References

1. B. Adenso-Díaz, M. Oliva González, and P. González-Torre. On-line timetable rescheduling in regional train services. *Transportation Research Part B: Methodological*, 33:387–398, 1999.
2. V. Cacchiani, D. Huisman, M. Kidd, L. Kroon, P. Toth, L. Veelenturf, and J. Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63(0):15 – 37, 2014.
3. L. Cadarso, Á. Marín, and G. Maróti. Recovery of disruptions in rapid transit networks. *Transportation Research Part E: Logistics and Transportation Review*, 53:15–33, 2013.
4. T. Dollevoet, D. Huisman, M. Schmidt, and A. Schöbel. Delay management with rerouting of pasengers. *Transportation Science*, 46:74–89, 2012.
5. J. Dumas and F. Soumis. Passenger Flow Model for Airline Networks. *Transportation Science*, 42(2):197–207, 2008.
6. P.J. Fioole, L.G. Kroon, G. Maróti, and A. Schrijver. A Rolling Stock Circulation Model for Combining and Splitting of Passenger Trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.
7. S. Kanai, K. Shiina, S. Harada, and N. Tomii. An optimal delay management algorithm from passengers' viewpoints considering the whole railway network. *Journal of Rail Transport Planning & Management*, 1:25–37, 2011.
8. L.G. Kroon, G. Maróti, and Nielsen L.K. Rescheduling of Railway Rolling Stock with Dynamic Passenger Flows. *Transportation Science*, 2014.
9. L.K. Nielsen, L. Kroon, and G. Maróti. A rolling horizon approach for disruption management of railway rolling stock. *European Journal of Operational Research*, 220:496 – 509, 2012.
10. M. Schachtebeck and A. Schöbel. To wait or not to waitand who goes first? delay management with priority decisions. *Transportation Science*, 44:307–321, 2010.

Pushing the Envelope: the role of slot scheduling in optimising the use of scarce airport resources

Konstantinos G. Zografos

The rapid growth of demand for air transport services coupled with political, physical and institutional constraints for building new airport capacity has resulted in acute airport congestion in UK and across Europe. Demand is expected to exceed capacity by as much as 2.3 million flights (or 11%) in the most-likely growth forecast scenario by 2030 at 138 Eurocontrol airports[1]. Imbalances between traffic and capacity generate serious undesirable externalities for air transport and the society at large. Almost one third of flights delayed in ECAC area in 2013, with the average delay per delayed flight exceeding 26 minutes[2]. Similarly, ATM inefficiencies in EU[3] were estimated to result in 10.8 million minutes of ATFM delays in 2012, costing around €11.2 billion to airspace users and passengers and producing 7.8 million tonnes of wasted CO2. Increasing complications for expanding capacity render a pure supply-side solution both expensive and practically difficult to implement. In effect, a more sustainable approach being able to better cope with the congestion problem with existing resources is called for.

Solutions aiming to manage congestion through the optimum allocation of scarce airport capacity have received a great deal of consideration from the airport community, policy makers, and researchers. Capacity at schedule co-ordinated airports is expressed in slots and allocated within the framework of

Department of Management Science
Lancaster University Management School
Lancaster, UK
E-mail: k.zografos@lancaster.ac.uk

[1] Eurocontrol, 2008. Long-Term Forecast: IFR Flight Movements 2008-2030. Forecast prepared as part of the Challenges of Growth 2008 project, Brussels, Belgium.

[2] Eurocontrol, 2014. CODA Digest: Delays to Air Transport in Europe Annual 2013. Report prepared by Eurocontrol's Central Office for Delay Analysis (CODA), Brussels, Belgium.

[3] International Air Transport Association (IATA), 2014. Fact Sheet: Single European Sky (SES). Available online at: `http://www.iata.org/pressroom/facts_figures/fact_sheets/pages/ses.aspx` (accessed May 20, 2014).

voluntary guidelines developed and evolved over the years under the auspices of IATA. A slot identifies a time interval, specific date and time, during which a carrier is permitted to use the airport infrastructure for landing or take-off at a slot-controlled airport. A fundamental concept in the slot allocation process is declared capacity and particularly its rationing and allocation on the basis of a complicated set of administrative rules, criteria, and priorities. Declared capacity represents an "artificial" measure of capacity specifying and controlling the number of slots available per unit of time. Therefore, slot scheduling and the setting of optimum declared capacity levels are closely interdependent and both lie at the heart of optimising the allocation and use of scarce airport resources.

The objective of this presentation is to provide an overview of the evolution of slot scheduling and declared capacity modelling, identify open research issues, and underline the potential of slot scheduling in optimising the allocation and use of scarce airport resources.

The existing slot allocation process produces poor outcomes in that it fails to properly match requested slots with those allocated to airlines. On top of that, slot misuse sharpens the capacity shortage due to poor use of scarce capacity. Even at airports where slot demand exceeds capacity, over 10% of the allocated slots go unused[4]. ACI Europe[5] estimated that slots unused due to their late return account for losses of around €20 million per season at large, congested European airports.

In order to deal with inefficiencies and limitations of the existing allocation practice, the policy and research community has placed the focus on two alternative (and potentially complementary to each other) directions: i) approaches introducing alternative, market-driven mechanisms aiming to allocate capacity among competing users by considering real market (or approximations of) valuations of access to congested airport facilities (e.g., congestion-based pricing schemes, primary/secondary trading, auctioning of part of or the entire slot pool) and ii) efforts aiming to improve the allocation efficiency of the IATA-based allocation mechanism from a slot scheduling point of view.

Slot scheduling signifies a challenging stream of research due to its potential to generate quick and drastic capacity utilisation improvements and the complexity and size of the resulting mathematical problems. The slot scheduling procedures currently in use suffer from a number of limitations such as the following: i) they are very simplistic in modelling the objectives, as well as the operational and regulatory constraints of the stakeholders involved in and affected by the slot allocation process, ii) they fail to consider the inherent dependency and complementarity of slots allocated to a network of airports, iii) they do not sufficiently capture the dynamic nature and uncertainty as-

[4] Steer Davies Gleave, 2011. Impact Assessment of Revisions to Regulation 95/93. Study prepared for the European Commission (DG MOVE), London, UK.

[5] Airport Council International (ACI) Europe, 2009. ACI Europe position on the proposed revision of the Council Regulation (EEC) No 95/93 on common rules for the allocation of slots at Community airports. Presentation at the TRAN Meeting at the European Parliament, March 25, Strasbourg, France.

sociated with airport capacity, iv) they employ empirical or ad hoc processes for determining (rather than assessing/computing) declared capacity, and v) they hardly address strong interdependencies among various resource allocation problems at strategic, (pre)tactical, and operational level. In addition, difficulties and inefficiencies in current allocation practice at a single airport feed into the hugely complex network-wide slot allocation problem for which there is no available decision support to authorities responsible for slot allocation. As a result, existing slot scheduling procedures do not realistically address the complexities of the real-world problem and apply an oversimplified approach that is eventually in the expense of allocation efficiency and utilisation of scarce airport resources.

Existing research has mainly focused on the tactical, ground holding problem, while more recently simplified modelling approaches of the strategic, single-airport, single-objective slot allocation problem were examined in the literature. The latter considered replacing the current allocation mechanism by mathematical models focusing mostly on the strategic (2-12 months before operation) allocation of slots at single-airport level. A common objective for the single-airport slot allocation problem is the minimisation of a delay-based cost function such as the so-called “schedule delay”. Schedule delay is a distance-based measure expressing the difference between requested and allocated slot times (often modelled as linear cost functions) subject to declared (mainly runway) capacity, turnaround, and slot/flight assignment constraints.

Future research towards the next generation of slot scheduling models should capitalise on existing models and expand their capabilities in several directions such as: i) new, realistic modelling representations of the strategic slot allocation problem taking into account various operational and regulatory constraints, the dynamic nature of both demand and capacity, the uncertainty of air transport operations, and, most importantly, the inherent interaction and complementarity of slots at the airport network level, ii) simultaneous consideration of multiple objectives (e.g., schedule delay, operational/queuing delay, resource utilisation, fairness and equity, environmental externalities), iii) alternative formulations of the objectives (e.g., non-linear cost functions for delay) and/or the constraints (e.g., rolling capacity, turnaround, flight connectivity) of the allocation problem coping with the trade-off between complexity and accuracy of the solution, and iv) powerful new adaptive search algorithms which can provide high quality solutions to complex, large-scale, real-world problems. Furthermore, despite its utmost importance and substantial influence on the efficiency of the allocation process, the declared capacity determination process has not been sufficiently examined in the literature. In particular, the setting of optimal declared capacity levels with view to the modelling of trade-offs between declared capacity levels, allocation efficiency, service level (e.g., actual operational/queuing delays), and utilisation of scarce airport resources merit further research in the years to come.

Existing research has already demonstrated the large room for improvement of the allocation outcomes. Improvements in slot allocation affect the demand-capacity mismatch and eventually reflect on the efficient use of scarce airport resources. Due to the intrinsic complexity and large scale of the slot allocation problem, the full potential of such improvements can only be viewed under the prism of airport network synergies. Fundamentally new mathematical structures, solution techniques and methodologies pave the way and bring promises for an enormous economic, environmental and societal impact with clear benefits for airlines, airports, passengers and the society at large.

Full Papers

Assigning and Scheduling Hierarchical Task Graphs to Heterogeneous Resources

Panayiotis Alefragis[1], Christos Gogos[3], Christos Valouxis[1,2], George Goulas[1,2], Nikolaos Voros[1], and Efthymios Housos[2]

[1]Technological Educational Institute of Western Greece. Dept. of Computer & Informatics Engineering, Greece

[2]University of Patras-Greece. Dept. of Electrical and Computer Engineering, Greece

[3]Technological Educational Institute of Epirus. Dept. of Accounting and Finance, Greece

Keywords: *Hierarchical Task Graphs, integer programming, parallel processing*

Abstract

Task Scheduling is an important problem having many practical applications. More often than not, precedence constraints exist between tasks, and a common way to capture them is through Directed Acyclic Graphs (DAGs). A DAG might contain a great number of tasks representing complex real life scenarios. It might be the case that logical groupings of tasks exist giving a hierarchical nature to the graph. Such Hierarchical Task Graphs (HTGs) have nodes that are further analyzed to DAGs or to other HTGs. In this paper a method of solving an HTG problem is presented based on the idea of gradually solving the problem by replacing subgraphs with virtual nodes. Integer Programming is used to generate virtual nodes that replace a subgraph, results from solving the subgraph problem using. So a series of subproblems are solved and starting from the deeper levels of the HTG a solution to the full problem emerges.

Introduction

Hierarchical task graphs (HTG) are directed graphs where nodes can either be simple or composite activities. Each hierarchy level is a set of interconnected directed acyclic graphs (DAG). Simple tasks are considered atomic, requiring a single resource to be used for their execution while a composite activity can use multiple resources at different hierarchical levels. Each composite activity can be represented as a subtree in the HTG. HTG is a typical high level representation of computer program kernels, but they can also be used in the scheduling of multiple development teams that are involved in multiple concurrent projects where each activity generates results that are used by other activities or involve recurring activities. The usual goal of these problems is to reduce the global makespan of the presented problem.

In this paper, we present a method that uses a MIP model to solve individual DAG sub-problems to optimality and a heuristic approach to solve the whole problem using a bottom up traversal of the HTG tree. The example is derived from the solution of an HTG for the assignment and scheduling of computational kernels to embedded multicore architectures. In the example presented in Figure 1, a two level HTG with 6 nodes at the top level is presented. Three resources are available to perform the tasks, in our case they are heterogeneous processors. The top level is presented on the left side, where node 5 is a composite activity and all other nodes are atomic ones. Node 5 is a DAG only containing simple nodes and is presented on the right side of the same figure.

Table 2. After the execution of activity, the result has to be "communicated" to the dependent tasks if these tasks are not performed by the same resource. This can be perceived as the communication time of variable values in our example, or the transportation time between two sites in a production example, or the collaboration time between two individuals that work on the same problem. In this simplified example, the "communication" cost is depicted by the values on the arcs of the DAGs. In the more general case, it can be the result of a function that involves the communicating resources, the time that this communication occurs, other available resources that help to perform the communication, etc. The required processing time to perform composite task 5 contains currently contains no values as it may be executed by different processors in parallel. Which processors and for how long task 5 will occupy will be the result of solving the sub-problem represented by the DAG at the lower level of HTG.

Two approaches to solve the problem of scheduling a Hierarchical Task Graph will be presented. The first approach embeds each subgraph to the graph that contains it. This process can occur recursively until no more composite tasks exist. Then the resulting flat DAG can be assigned and scheduled at the available resources using either a heuristic method like HEFT or if the size of the problem permits it using an ILP solver. The second approach uses an ILP formulation in order to solve each subgraph in turn and uses the generated subgraph schedule as a resource based clustering of the contained tasks to form the schedule of the graph that contains the subgraph.

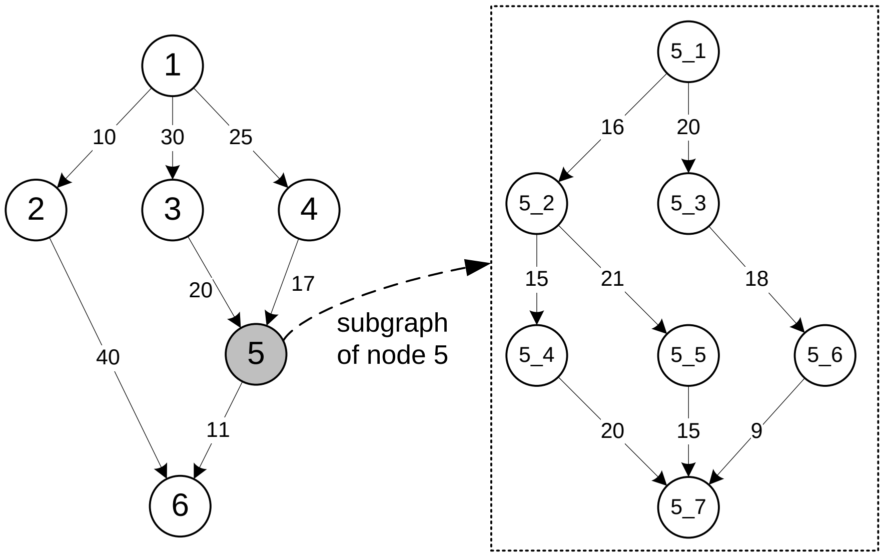


Figure 1: Example of a Hierarchical Task Graph

Table 1: Execution Times for Tasks of the HTG

	Tasks					
Resources	1	2	3	4	5	6
R0	15	16	38	19	-	17
R1	22	31	40	20	-	25
R2	37	42	51	30	-	22

Table 2: Execution Times for Tasks of the Subgraph

	Tasks						
Resources	5_1	5_2	5_3	5_4	5_5	5_6	5_7
R0	10	29	38	19	20	15	29
R1	15	32	40	25	28	36	31
R2	37	27	26	32	47	30	42

DAG makespan optimization

DAG scheduling is a well-studied subject [1],[2]. Several methods exist that can be used in order to assign tasks to resources and create individual schedules for each resource. What is interesting is that the individual resource schedules cannot be decided in isolation due to task dependencies and the "communication" cost they imply. Heuristics based methods are able to efficiently solve DAGs with several hundreds of nodes. These methods either belong to the category of list scheduling with prominent examples being Earliest Time First (ETF) [3], Heterogeneous Earliest Finish Time (HEFT) [4],[5] and Critical Path on a Processor (CPOP) [6] or to the category of clustering where examples of such algorithms are Dominant Sequence Clustering (DSC) [7] and Linear Clustering Method (LCM) [8]. Mathematical programming based methods exhibit much bigger execution time and do not scale well for bigger instances, but are able to find high quality solutions [9],[10],[11],[12].

ILP Model for solving a task DAG

As our test case is part of a compiler tool chain and each execution requires the solution of 100 - 10000 DAG it was a requirement that each DAG formulation and solution should take no more that some seconds. A practical observation was that if the total number of nodes for each DAG is less than 30 then the problem can be efficiently solved using an ILP solver within the available time budget. In our experiments this restriction has been satisfied using the open source IP solver COIN-CBC running on a current PC. The topography of the DAG seems to have little impact on the execution time of the approach due to the fact that equations are effectively generated for all pairs of nodes.
The mathematical model uses the set of tasks T and the set of available resources P. Each resource can process a given task at different time horizon, i.e. that the problem model is heterogeneous which in our example means that the processors are not identical. For project planning problems this would mean that the worker will have different skills and could handle the same task with a different execution duration. The execution time of each task t by each resource p is given by the parameter w_{tp}. The "communication" cost between task t and task t', when they are not assigned at the same resource is given by $c_{tt'}$, 0 otherwise. We have two sets of decision variables. The binary variables y_{tp} depict an assignment of task $t \in$

T to resource $p \in P$ and take the value 1 if task *t* is assigned at processor *p* and 0 otherwise. The integer variables x_t are defined over each $t \in T$ and their value represents the start time of task *t* by the resource corresponding to the y_{tp} variable with value 1. Since the problem is described as a DAG with a set of nodes *V* and a set of edges *E*, it should be noted that the set of tasks and the set of nodes are conceptually identical, while the set of edges represents precedence constraints between tasks with weights of edges associated with communication costs.
The objective function of the model is shown in equation (1) representing the target of minimizing the total schedule length, which is also known as makespan.

$$\text{minimize}\, x_{t_s} \qquad t_s \text{ is the sink node of the DAG} \quad (1)$$

Three groups of constraints are defined. The first one ensures that each task should be assigned at exactly one resource (2).

$$\sum_{p \in P} y_{tp} = 1 \qquad \forall t \in T \qquad (2)$$

For each task *t* let T_t be the set of tasks that have to be completed before *t* starts execution. The second group of constraints states that for each task *t* the corresponding start time should be greater than all the finish times of tasks that belong to T_t. In addition, when task *t* is scheduled to a different resource than a task *t'* that it depends on, the "communication" cost between them should also be considered. In order to model this constraint three new variables are introduced et_t, $z_{tt'}$ and $k_{ptt'}$. et_t is an integer variable and corresponds to the execution time of task t derived from equation (3). Variable $z_{tt'}$ is binary and equals 1 when both tasks *t* and *t'* are assigned at the same resource and 0 otherwise. It is defined by variable $k_{ptt'}$ which corresponds to the product of binary variables $y_{t'p}$ and $y_{t'p}$. Since the product between variables violates the linearity of the model variable $k_{ptt'}$ assumes its value indirectly through inequalities (4), (5) and (6). $z_{tt'}$ is defined by equation (7) as the sum of products between variables $y_{t'p}$ and $y_{t'p}$. Finally, inequality (8) states that the difference between execution times for task *t* and task *t'* provided that *t* depends on *t'* should be no less than the processing time of task *t* on the designated resource plus the extra communication time needed when tasks are not assigned at the same resource.

$$et_t = \sum_{p \in P} w_{tp} y_{tp} \qquad \forall t \in T \qquad (3)$$

$$k_{ptt'} \le y_{tp} \qquad \forall p \in P; \forall t \in T, t' \in T_t \qquad (4)$$

$$k_{ptt'} \leq y_{t'p} \qquad \forall p \in P; \forall t \in T, t' \in T_t \qquad (5)$$

$$k_{ptt'} \geq y_{tp} + y_{t'p} - 1 \qquad \forall p \in P; \forall t \in T, t' \in T_t \qquad (6)$$

$$z_{tt'} = \sum_{p \in P} k_{ptt'} \qquad \forall t \in T, t' \in T_t \qquad (7)$$

$$x_t - x_{t'} \geq et_t + c_{tt'}(1 - z_{tt'}) \qquad \forall t \in T, t' \in T_t \qquad (8)$$

For each task *t*, let *T'* be the set of tasks for which there is no path in the graph that connects then to task *t*. Two tasks *t* and *t'* are considered to be independent if by starting from *t* (or *t'*) and then recursively examining all predecessors of this task, the entry node is met before node *t'* (or *t*). The third group of constraints guarantees that when two independent tasks are assigned at the same resource, they will not overlap. Given that two independent tasks *t* and *t'* are are assigned at the same resource then $z_{tt'}$=1. If x_t, the start time of task *t*, is less than $x_{t'}$, the start time of task *t'*, then x_t plus the execution time of task *t (et_t)* should be less than $x_{t'}$ (9). Likewise, when $x_{t'}$ is less than x_t inequality (10) assures that no execution overlap occurs. Binary variable $m_{tt'}$ ensures that exactly one of inequalities (9) and (10) should hold.

$$(1 - z_{tt'})M + x_{t'} - x_t \geq et_{t'} - M(1 - m_{tt'}) \; \forall t \in T, t' \in T'_t \qquad (9)$$

$$(1 - z_{tt'})M + x_t - x_{t'} \geq et_{t'} - M(m_{tt'}) \qquad \forall t \in T, t' \in T'_t \qquad (10)$$

The proposed model is adequate to solve real world problems that arise during the mapping and scheduling of sequential Scilab code in an automatic parallelizing compiler toolflow that targets embedded multicore architectures in the context of the EU founded FP7 ICT ALMA project.

In the case where an HTG does not contain-subgraphs that represent recurring events (loops), it is possible to embed these subgraphs that are deeper in the hierarchy of the HTG to the subgraph that contains them. In a computer program this can be perceived as function inlining, as each subgraph is a DAG of tasks. In a project schedule this can be perceived as including the activities of each department in the global scheduling of a store and then the produced solution will be embedded in the assignment and scheduling problem of the firm, where staff members (resources) that perform the tasks do not work for a single department or store. After flattening, the resulting graphs will be a DAG. For example, Figure 2 represents the HTG of Figure 1 after the flattening transformation is performed.

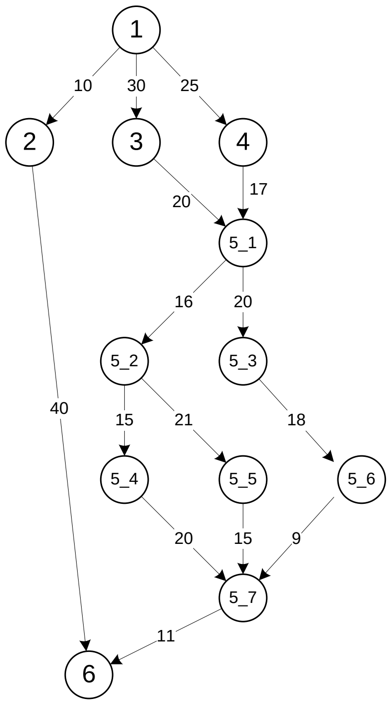


Figure 2: Flattened DAG of Figure 1 HTG

At the lowest level of an HTG most subgraphs will be DAGs that contain only simple tasks. After solving such a subgraph, the emerged solution will specify the assigned resource that each task should be scheduled on and for each task the sequence and the estimated time that it will start executing relative to the execution start time of the initial task. For example, when the subgraph of Figure 1 is solved, the solution presented in Figure 3 may arise. This solution schedules tasks 5_1, 5_2, 5_5, 5_4 and 5_7 in resource 0, schedules no tasks to resource 1 and schedules tasks 5_3 and 5_6 on resource 2. The makespan of the given schedule is 124 time units. If the generated graph can be efficiently solved by the ILP model, the ILP formulation is passed to an ILP solver and the optimal solution is produced.

Bottom-up solution of HTG by subgraph replacement with virtual nodes

With or without flattening at the lower levels, most large problems that are represented by an HTG cannot be totally converted to a DAG. In order to use the subgraphs solutions in the solution process of the enclosing subgraph, the following procedure is applied. The HTG is traversed bottom up and the subgraphs that can be solved without any modification, i.e. they only contain simple tasks or their subgraphs can be flattened, are solved applying a DAG algorithm. The selection of the solution algorithm is done based on characteristics of the graph. If the graph is small an ILP solver is used, if the graph is large a heuristic is applied. For the graphs that contain composite tasks that we have a solution for their subgraph the following transformation is performed. The composite task is replaced by a number of virtual tasks that are equal to the number of the used resources in the solution found. Each virtual task has an execution time of the combined work of the assigned tasks on the associated resource. All incoming and outgoing dependency arcs of the composite resource are replicated and the outer DAG is solved in a similar manner. This process continues iteratively until the top HTG layer is solved. For example, in Figure 4, pseudo task 5_R0 is a task that has an execution duration of 124 and should be executed in processor R0 and pseudo task 5_R2 is a task that has an execution duration of 56 and should be executed on processor R2. Theoretically, a pseudo task 5_R1 could have been created, but since it contains no tasks on the subgraph's schedule, it can be neglected. It is important to note that by specifying the schedule start time of the pseudo task that contains the root node of the subgraph, all other pseudo tasks are relatively positioned based on the solution of the subgraph. If for example pseudo task 5_R0 is scheduled to start at time point 100 then pseudo task 5_R2 should start at 130 and should finish at 186. If any of the virtual tasks is delayed to start compared to the offset of the subgraph solution, a new makespan for the subgraph solution should be calculated.

The virtual task that contains the root node of the subgraph (5_R0 in the example, since node 5_1 is scheduled on processor R0) should inherit the incoming edges of the nodes that were immediate predecessors (node 3 and node 4) of the composite task that was removed. Likewise the pseudo task that contains the sink node of the subgraph (5_R0 in the example, since node 5_7 is scheduled on processor R0) should inherit the outgoing edges to the nodes that were immediate successors (node 6) of the composite task that was removed.

The graph shown in Figure 4 is solved using the mathematical module using execution information included in Table 3. The information about the resource that each pseudo task should be assigned is implicitly included in the model and the only variable about the subgraph that remains to be decided is the start time of the pseudo task containing the root node of the subgraph. A natural extension is to calculate the total execution time of the assigned tasks of each virtual task for all the available resource and constraints that only one resource should be assigned at a virtual task and that no two virtual tasks that belong to a virtual task group should be assigned at the same resource. This will provide the required flexibility to exchange the assigned work to a resource when the surrounding tasks information is available during the solution of the outer subgraph.

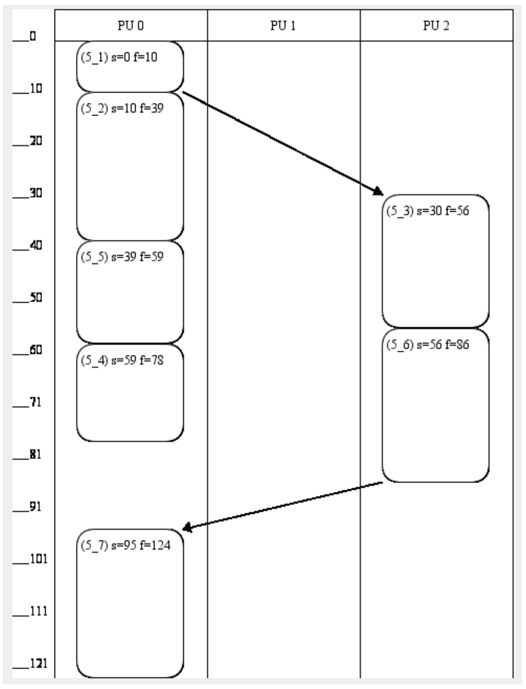


Figure 3: Schedule of subgraph

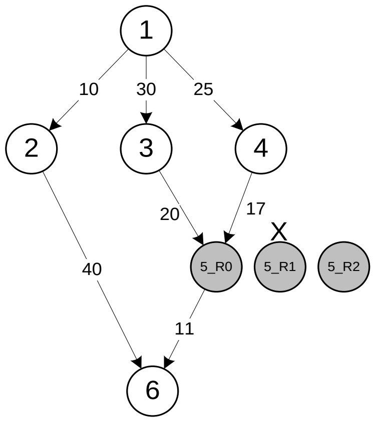


Figure 4: HTG with Composite Task 5 Replaced by Pseudo Tasks 5_R0, 5R1 and 5R2. Task 5_R1 is immediately dropped since no tasks of the subgraph are scheduled in processor R1.

A special case exists, when the subgraph represents the body of nested recurring events (loops). In this case, the subgraph solution represents a recurring task schedule that will be executed for each loop iteration and thus it is not possible to determine the exact makespan of the virtual tasks in the general case. In this case, for all the resources that are used in the solution of the subgraph, it is only possible to estimate the execution time of the assigned tasks before the execution of the virtual task. For all the other tasks that are assigned at the processor and scheduled after the virtual task, their start time can only be determined in relation to the finish time of the recurring virtual task.

Table 3. Execution Times for Tasks and Pseudo Tasks of the HTG

	Tasks						
Resources	1	2	3	4	5_R0	5_R2	6
R0	15	16	38	19	124	X	17
R1	22	31	40	20	X	X	25
R2	37	42	51	30	X	56	22

Our current approach is to split the outer DAG scheduling problem into two sub problems, the “before” sub problem that determine tasks scheduling of all resources before the recurring event and the “after” subproblem for all the tasks after the execution of the virtual tasks. This implies that tasks that depend on the virtual tasks will belong to “after” sub problem, tasks that the virtual tasks depend on will belong to the “before” sub problem and independent tasks can be assigned and scheduled before, after or in parallel to the virtual tasks using resources that are not used by the virtual tasks. The above process may lead to a possibly suboptimal solution. On the other hand, this approach guarantees that no prolonged wait time for tasks that require results generated during the execution of the virtual tasks will occur, thus preventing execution blocking on all the other resources. The side effect is that no deterministic execution makespan of the subgraph can be determined if the number of recurring events is not algebraically determined. For our application problem, the ALMA toolchain provides feedback from a platform simulator and thus the real execution time of the recurring task can be determined, irrelevant to the actual iterations number. In our use case, during the subsequent applications of the described optimization algorithm the estimation of the actual execution time of the recurring event will be used, allowing the algorithm to improve resource utilization, as the whole DAG can be scheduled as in the case where the subgraph does not include loop nests.

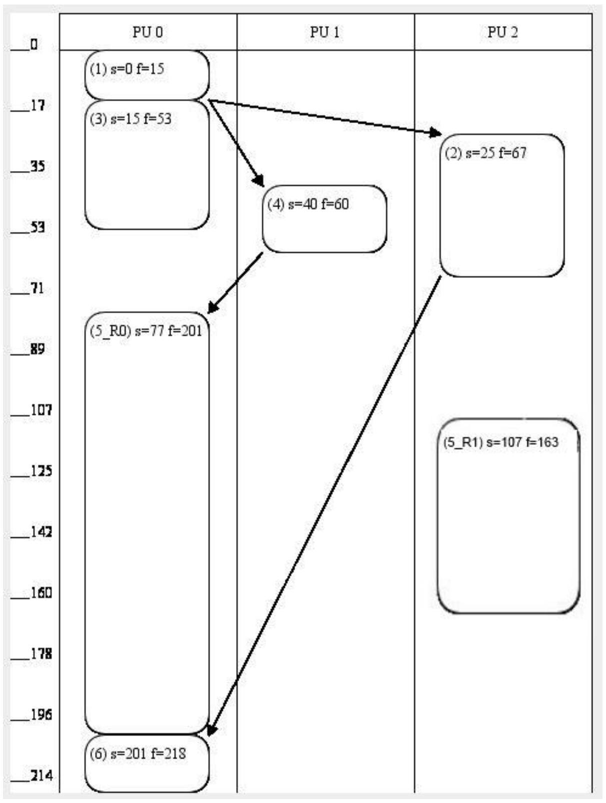


Figure 5: Schedule of the graph including pseudo tasks

ILP model modifications

A set of modification to the ILP model that was previously described are needed in order to recursively solve each subgraph until the original HTG is fully scheduled. The main modification that is necessary is the replacement of each composite task with a number of pseudo tasks, equal to the number of the resources used in the solution of the corresponding subgraph. These pseudo tasks have predefined values for their corresponding binary variables y_{tp} since the assigned resource that each one of them will be scheduled to is known. For each pseudo task group, the value of variable x_t, which denotes the start time of pseudo task *t*, has to be determined only for the pseudo task that contains the root node of the subgraph. The values of variables x_t for the remaining pseudo tasks in a group are determined based on the offset exposed in the solution of the subgraph. Regarding the constraints, only the pseudo task that contains the root node of the subgraph and the pseudo task that contains the sink node of the subgraph have to be included in the set of tasks that participate in the generation of the second group of constraints. Pseudo tasks that neither contain the root nor the sink node of the subgraph only have to participate to the third group of constraints in order to prevent simultaneous execution of tasks by the same resource.

This is required as no edges exist between the pseudo tasks and all the other tasks in the DAG, making them look independent. It should be noted that pseudo tasks share the same set of independent tasks with the composite task that they replace. Consequently, for each pseudo task, constraints belonging to the third group have to be generated with respect to all other tasks that have no direct or indirect path to the composite task that they replace.

The final schedule of the graph of Figure 4 is presented in Figure 5. The HTG is finally scheduled, having a makespan of 218 time units. The proposed algorithms have been applied to code sources that are represented by HTGs with up to 15 layers, 200 to 500 composite tasks and 1000 to 2000 leaf tasks that included recurring tasks and managed to produce parallel solutions in less than 1h using a typical PC.

Conclusions and future work

In this paper, a generic algorithmic approach to assign and schedule hierarchical task graphs to heterogeneous resources is presented. The proposed approach is currently applied to a parallelizing compiler tool chain for automatic parallelization of sequential SciLab programs to embedded multicore architectures but can be used in other application areas. We are currently working in supporting multiple solutions selection for the composite tasks during the bottom up solution process as well as the application of meta-heuristic local search optimization techniques for post processing of the generated solutions. We also plan the inclusion of a more detailed modeling of both the “communication” model between tasks as well as a more detailed modeling of the processing resources.

Acknowledgement: This work is co-funded by the European Union under the 7th Framework Program under grant agreement ICT-287733**,** project “Architecture oriented parallelization for high performance embedded Multicore systems using scilAb (ALMA)”.

References

1. Sinnen O., Task Scheduling for Parallel Systems. John Wiley & Sons, 2007.

2. Canon L.-C., E. Jeannot, R. Sakellariou, and W. Zheng, Comparative Evaluation Of The Robustness Of DAG Scheduling Heuristics, in Grid Computing, S. Gorlatch, P. Fragopoulou, and T. Priol, Eds. Springer US, pp. 73–84, 2008.

3. Hwang J.-J., Y.-C. Chow, F. D. Anger, and C.-Y. Lee, Scheduling Precedence Graphs in Systems with Interprocessor Communication Times, SIAM Journal on Computing, vol. 18, no. 2, pp. 244-257, Apr. 1989.

4. Topcuoglu H., S. Hariri, and M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260 –274, Mar. 2002.

5. Bittencourt L. F., R. Sakellariou, and E. R. M. Madeira, DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm, in 2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 27 – 34, 2010.

6. Kwok Y.-K. and I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 5, pp. 506-521, 1996.

7. Gerasoulis A. and T. Yang, A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors, Journal of Parallel and Distributed Computing, vol. 16, no. 4, pp. 276-291, Dec. 1992.

8. Kim S. J. and J. C. Browne, A general approach to mapping of parallel computation upon multiprocessor architectures, in International Conference on Parallel Processing, vol. 3, pp. 1-8, 1988.

9. Davare A., J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, Classification, Customization, and Characterization: Using MILP for Task Allocation and Scheduling, Technical Report Identifier: EECS-2006-166, EECS Department, University of California at Berkeley, California, Dec. 2006.

10. Valouxis C., C. Gogos, P. Alefragis, G. Goulas, N. Voros and E. Housos, DAG Scheduling using Integer Programming in heterogeneous parallel execution environments, in Proceeding of the Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2013), pp. 392-401, Gent, Belgium, 26th - 30th August 2013.

11. Davidovic, T., Liberti, L., Maculan, N., & Mladenovic, N., Mathematical Programming-Based Approach to Scheduling of Communicating Tasks, Les Cahiers du GERAD, 2004.

12. Tompkins, M. F., Optimization techniques for task allocation and scheduling in distributed multi-agent operations, Doctoral dissertation, Massachusetts Institute of Technology, 2003.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Feature-based tuning of single-stage simulated annealing for examination timetabling

Michele Battistutta · Andrea Schaerf · Tommaso Urli

Abstract We propose a single-stage Simulated Annealing procedure for the Examination Timetabling problem (as formulated in the 2nd International Timetabling Competition). Over our approach, we perform a statistically principled experimental analysis, in order to understand the effect of parameters and to devise a feature-based parameter tuning strategy. The outcome of this work (which is still ongoing) is that this rather straightforward search method, if properly tuned, is able to compete with all the state-of-the-art specialized solvers.

Keywords Examination Timetabling · Simulated Annealing · Parameter Tuning

1 Introduction

We consider the Examination Timetabling problem in the version used in the 2nd International Timetabling Competition (ITC2007 [16], Track 1). For this problem (see [15] for the detailed formulation), we propose a single-stage Simulated Annealing (SA) procedure, along the lines of our previous work on the other two tracks of ITC2007 [2, 3, 6].

We perform an experimental analysis of our solver on the 12 public instances released for the ITC2007, which are, up to now, the only available ones. Specifically, we propose a parameter tuning procedure to obtain a good configuration of the solver for the general case. The tuning procedure works in two steps. In the first step, we identify the most important parameters and we fix the value for all the other ones. In the second one, we develop, through

M. Battistutta, A. Schaerf, and T. Urli
DIEGM, University of Udine
Via delle Scienze 206
E-mail: {michele.battistutta,schaerf,tommaso.urli}@uniud.it

a regression model, a linear function that correlates the value of the most important parameters to the features of the instances.

The outcome of this work (which is still ongoing) is that this rather straightforward search method, properly tuned with a statistically-principled procedure, is able to compete with all state-of-the-art specialized solvers, producing also the best results for a few instances.

2 Search method

Our search method is based on local search. To this regard, we use the following features:

Search Space: The search space is composed by all the assignments of exams to periods and rooms. States that violate hard constraints (e.g., precedences and conflicts) are included in the search space, and they are penalized in the cost function with a high weight (called w_H).

Neighborhood Relation: The neighborhood relation is composed by the union of two basic moves: *1)* Reschedule a single exam to a new period and/or new room *2)* Swap both period and room of two exams. The random selection of the candidate neighbor is performed in two steps: First select the neighborhood (Reschedule or Swap) according to a non-uniform distribution that selects Swap with probability sr and Reschedule with probability $1 - sr$ (where sr stands for swap rate, and is a parameter of the method). Second, perform a uniform selection of the specific move within the corresponding neighborhood.

Stop Criterion: The stop criterion is based on the total number of iterations, so as to have approximately a constant running time independently of the other parameters of the method.

Initial Solution: The initial solution is totally random, and is obtained by assigning a random period and a random room to each exam.

We employ a Simulated Annealing method that uses a cutoff-based non-geometric cooling scheme[11], that speeds up the search in the initial phase. Specifically, the temperature is decreased (multiplying it by the cooling rate α) when the first of the following two conditions holds: *a)* the allotted number of iterations (n_S) has been expired or *b)* the allotted number of *accepted* moves (n_A) have been performed.

The stopping condition of the method is based on the total number of iterations it_{max}, rather than explicitly on the final temperature. For the sake of comparability, it_{max} is set to a fixed value such that the running time is approximately the one prescribed by the ITC2007 benchmarking tool (324s on our machine). The resulting value is $it_{max} = 5 \times 10^8$.

The final temperature t_{min} is passed to the solver and it is used along with it_{max} and the cooling rate α to compute the number of neighbors sampled at each temperature (n_S). More precisely, we pass to it the ratio $tr = t_0/t_{min}$ between the initial and the final one.

$$n_S = it_{max} \Bigg/ \left(\frac{-\log(tr)}{\log \alpha} \right) \tag{1}$$

Actually, the final temperature t_{min} is different for each run due to cut-offs, but we consider the one that is reached in case of a standard cut-off-free execution.

Similarly, instead of using directly the parameter n_A, we replace it with its ratio $\rho = n_S/n_A$ with the neighbors sampled n_S.

The use of the ratios (tr and ρ) instead of absolute values prevents from including meaningless configurations in the analysis, such as those in which the final temperature is greater than the initial one.

Summarizing, the search procedure relies on the following six parameters:

– starting temperature (t_0),
– temperature range (tr),
– ratio between neighbors accepted and neighbors sampled at each temperature (ρ),
– cooling rate (α),
– hard constraints weight (w_H),
– swap rate (sr)

that have to be tuned as explained in the following section.

3 Experimental analysis

In order to tune the parameters of our method, we have carried out a statistically principled experimental analysis over the 12 available instances from the ITC2007 competition.

3.1 Preliminary tuning

For the tuning phase, without resorting to any "premature commitment" [10], we have assigned meaningful ranges to the six parameters described in Section 2, and we have sampled 100 alternative configurations from the *Hammersley point set* [9] based upon such ranges. The Hammersley point set *is scalable*, both with respect to the number of sampled points, and to the dimensions of the sampling space. Moreover, the sampled points exhibit *low discrepancy*, i.e., they are space-filling, despite being random-like, and are thus particularly indicated for parameter tuning applications. The sampled configurations were then tuned through an automated *F-Race(RSD)* [4] with confidence 95 (p-value < 0.05). The considered parameters are summarized in Table 1, together with the values identified by the tuning.

All the experiments were generated and executed automatically using the tool JSON2RUN [18] on an Ubuntu Linux 13.04 machine with 16 Intel® Xeon®

Parameter	**Symbol**	**Tuned value**
Starting temperature	t_0	900
Temperature range	tr	1000
Cooling rate	α	0.99
Neighbors sampled / neighbors accepted ratio	ρ	0.08
Hard constraints weight	w_H	30
Swap rate	sr	0.8
Final temperature	t_{min}	0.9 (derived)
Neighbors sampled at each temperature	n_S	727467 (derived)
Neighbors accepted at each temperature	n_A	58197 (derived)

Table 1 Parameters identified by automatic tuning through *F-Race(RSD)*.

CPU E5-2660 (2.20 GHz) physical cores, hyper-threaded to 32 virtual cores. A single virtual core has been dedicated to each experiment.

3.2 Feature-based tuning

The results attained by our tuned algorithm were good on some instances, but not on all of them. We therefore decided to investigate the origin of this effect, by looking closely at the instances that failed. The outcome of the analysis was that the overall best parameter configuration (see Table 1) was actually sub-optimal for the failing instances, and yielded violations of the hard constraints. A subsequent *F-Race(RSD)* limited to those instances, revealed that, while for most parameters the winning values found in the preliminary race were also good for the failing instances, t_0 and w_H had to be tuned differently. Since *F-Race(RSD)* is based on ranks, and not directly on the obtained costs, the failing instances were only seen as instances with low statistical significance.

We thus ran an exploratory set of experiments on all the instances, and with 100 repetitions for each experiment, by varying t_0 and w_H together. The experiments revealed that, because of the geometric-like temperature update scheme, which is based on α and ρ, the time spent at high temperature is very short, and thus setting a high value for t_0 is always a reasonably safe choice. We thus set $t_0 = 1000$, and ran another set of experiments to study the effect of w_H, which appeared to be much more relevant.

This second set of experiments revealed a recurrent correlation between the value of w_H and the distributions of costs, which is depicted in Figure 1 for one specific instance (no. 4).

As it is visible from the plot, when approaching low values of w_H, the cost increases very steeply because of the increase in the hard constraint violations. This is expected, as a low w_H makes it more likely to choose a neighboring solution with hard constraint violations. On the other hand, as w_H gets farther from the danger zone, the number of solutions with hard constraint violations decreases, but the costs related to soft constraints increases, because the search procedure is not able to exploit the possibility to cross the feasibility boundary.

Ideally, the goal of a good tuning would be to find the parameter configuration that yields the best cost related to soft constraints, while also minimizing

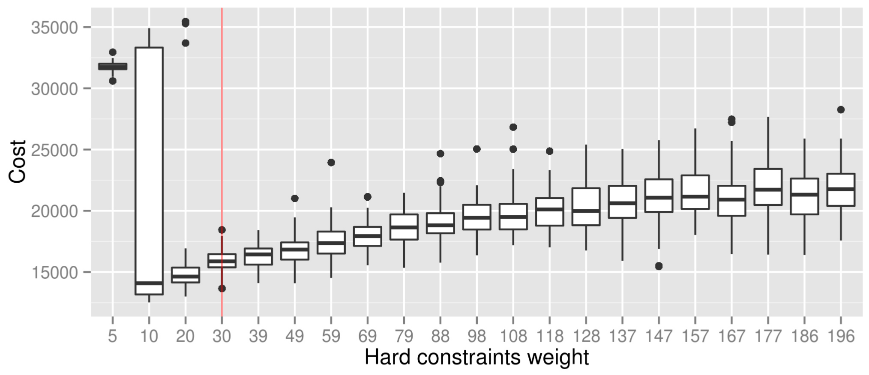


Fig. 1 Correlation between w_H and cost distribution (in logarithmic scale) on instance 3.

the number of violations of the hard constraints. However, since the ideal w_H can differ significantly from instance to instance, we decided to drop the choice of single-point tuning, i.e., one parameter configuration for all instances, and explored a way to compute the ideal w_H "on-the-fly", based on the features of the instance at hand.

3.2.1 Per-instance tuning

First, we observed that a robust strategy to reduce the number of hard constraint violations without increasing too much the cost related to soft constraints, consisted in choosing, for each instance, the value of w_H that minimized the 95^{th} percentile, of the cost distributions (highlighted by the vertical red line in Figure 1).

3.2.2 Feature-based parameter regression

Once an ideal value for w_H was identified for each instance, we looked at the instance features to see whether it was possible predict this value dynamically based on them.

We have considered the set of features described in [14], and a number of additional ones. The features that turned out more useful in our case are summarized in Table 2.

Given the feature values, we built a linear regression model in `R` [17] based on the 100 repetitions of the experiments with varying w_H. The model selection procedure works as follows. First, a model without any variable is generated, this model generates a constant w_H for all the instances and, as expected, has a bad approximation of the ideal w_H. Then, we add one feature at a time, always choosing the one that, if added to the model, minimizes the *Akaike information criterion* (AIC) [12], which is known to have good performances for prediction. The procedure stops when adding one more feature would not lead to a model with a better approximation.

I	Exams	Students	Periods	Rooms	Two in a row	Two in a day	Period spread	Frontload (Exams)	Frontload (Periods)
1	607	7891	54	7	7	5	5	100	30
2	870	12743	40	49	15	5	1	250	30
3	934	16439	36	48	15	10	4	200	20
4	273	5045	21	1	9	5	2	50	10
5	1018	9253	42	3	40	15	5	250	30
6	242	7909	16	8	20	5	20	25	30
7	1096	14676	80	15	25	5	10	250	30
8	598	7718	80	8	150	0	15	250	30
9	169	655	25	3	25	10	5	100	10
10	214	1577	32	48	50	0	20	100	10
11	934	16349	26	40	10	50	4	400	20
12	78	1653	12	50	35	10	5	25	5

Table 2 Instance features for the ITC2007 benchmark instances.

Component	**Symbol**	**Coefficient**	**(Cumulative)** R^2
Intercept	—	18.356988	—
Period Spread Penalty	**PS**	2.311492	0.26
Frontload Periods	$\mathbf{Fl_P}$	-1.7743	0.509
Periods	**P**	1.322	0.824
Two In a Day	**TiD**	1.005	0.894
Students	**S**	-0.0027	0.943
Rooms	**R**	0.292	0.975

Table 3 Coefficients and correlation of the linear predictor for w_H.

Table 3 shows, in order, the features added to the model, with their coefficient and cumulative coefficient of correlation (R^2) which measures its prediction quality. Therefore, the linear model corresponds to computing the following formula

$$\begin{aligned}\mathbf{w_H} = {} & 18.356988 + 2.311492 \times \mathbf{PS} - 1.7743 \times \mathbf{Fl_P} \\ & +1.322 \times \mathbf{P} + 1.005 \times \mathbf{TiD} - 0.0027 \times \mathbf{S} + 0.292 \times \mathbf{R},\end{aligned}$$

which can thus be used to obtain a per-instance tuning of the w_H, which should ideally generalize also to instances outside of the training set.

We have validated the effectiveness of this tuning for our approach over the 12 instances of the ITC2007 competition. The results of the comparison are described in the next section.

4 Comparison of results

Table 4 reports the comparison with the 5 finalists of ITC2007. Specifically, we add our solver as a further competitor, and rerun the competition adjudication by applying the ranking procedure on 10 runs for each of the 6 solvers. According to Table 4 our solver has the lowest sum of ranks (8.58), and thus would have won the competition if submitted at that time.

Table 5 shows our average results (for 100 runs), in comparison with subsequent results available from the literature. We include in Table 5 only those

I	Müller	Gogos	Atsuna *et al*	De Smet	Pillay	Us
1	15.5	25.5	45.5	35.5	55.5	5.5
2	14.8	35.5	48.8	25.5	52.2	6.2
3	28.7	20.5	34.9	53	36.1	9.8
4	30.1	31.95	38.55	48.5	28.4	5.5
5	15.3	35.5	45.5	25.5	55.5	5.7
6	20.1	28.95	35.6	45.9	46.95	5.5
7	15.4	35.5	46.5	25.5	54.5	5.6
8	14.7	25.5	35.5	55.5	45.5	6.3
8	15	31.4	43.6	33.5	53.4	6.1
10	35.9	54.5	21.8	9.5	40.9	20.4
11	41.4	22.5	45	45	16.5	12.6
12	24.8	50	9.7	50	34.7	13.8
avg	22.64	33.11	37.58	37.74	43.34	**8.58**

Table 4 Comparison with the competition finalists.

I	McCollum *et al* [14]		Bykov & Petrovic [5]		Hamilton-Bryce [8]		Alzaqebah [1]		Us	
	$\bar{f}$	F%	$\bar{f}$	F%	$\bar{f}$	F%	$\bar{f}$	F%	$\bar{f}$	F%
1	4799	100	4008	100	5469	100	5517	100	**4004**	100
2	425	100	404	100	450	100	538	100	**399**	100
3	9251	100	**8012**	100	10444	100	10325	100	9033	98
4	15821	100	**13312**	100	20241	100	16589	100	15132	100
5	3072	100	**2582**	100	3185	100	3632	100	2876	100
6	25935	100	**25448**	100	26150	100	26275	100	25912	100
7	4185	100	3893	100	4568	100	4592	100	**3747**	100
8	7599	100	**6944**	100	8081	100	8328	100	7711	100
9	1071	100	**949**	100	1061	100	—	—	994	100
10	14552	100	**12985**	100	15294	100	—	—	14956	96
11	29358	100	**25194**	100	44820	100	—	—	28773	89
12	5699	100	**5181**	100	5464	100	—	—	5648	98

Table 5 Comparison of available results.

results that are compliant with ITC2007 rules, in terms of timeout. The column F% reports the percentage of feasible solutions obtained. Average costs are computed on feasible solutions only.

Table 5 shows that our results are outperformed by the ones of Bykov and Petrovic [5] in 9 out of 12 instances, and they are superior on the 3 remaining ones. With respect to all the other researchers, our results are globally superior.

We acknowledge that our solver does not find feasible solutions more often than the other solvers. This is inherent to the choice of using a single-stage approach that does not solve the feasibility in advance, but rather tries to optimize the objective function and to satisfy the hard constraints all at once.

5 Conclusions

Our solver turned out to have complementary characteristics with respect to previous research. Specifically, it is able to improve the costs on the easier instances (in terms of hard constraints), but it is not always able to find a feasible solution for the hard ones.

Admittedly, the results of Bykov and Petrovic [5], that rely on the Kempe-chain neighborhood, are superior. Nevertheless, we consider these results, which are still preliminary, quite encouraging for further improvements.

Given that w_H turned out to be the most critical parameter, current work involves the use of a *strategic oscillation* approach [7] that adaptively changes the value of w_H, depending on the current number of violations. This seems to be a promising approach to improve our performances on the harder instances.

Regarding the experimental analysis, one of the main obstacles to our study was the scarcity of instances to use for training the linear regression model. In order for the model to attain better generalization properties, i.e., to work well on instances outside of the training set, the size of the training set should ideally be much larger. To deal with this aspect, we are working on an instance generator, in the spirit of the one developed by Lopes and Smith-Miles [13], that will be able to create realistic instances with diverse features.

References

1. M Alzaqebah and S Abdullah. An adaptive artificial bee colony and late-acceptance hill-climbing algorithm for examination timetabling. *Journal of Scheduling*, pages 1–14, 2013.
2. Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. A simulated annealing approach to the curriculum-based course timetabling problem. In *Proc. of the 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA-13)*, 2013.
3. Ruggero Bellio, Luca Di Gaspero, and Andrea Schaerf. Design and statistical analysis of a hybrid local search algorithm for course timetabling. *Journal of Scheduling*, 15(1):49–61, 2012.
4. Mauro Birattari, Z. Yuan, P. Balaprakash, and Thomas Stützle. *F-Race and iterated F-race: An overview.* Springer, Berlin, 2010.
5. Yuri Bykov and Sanja Petrovic. An initial study of a novel step counting hill climbing heuristic applied to timetabling problems. In *Proc. of the 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA-13)*, pages 691–693, 2013.
6. Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research*, 39:1615–1624, 2012.
7. Fred Glover and Manuel Laguna. *Tabu search.* Kluwer Academic Publishers, 1997.
8. R Hamilton-Bryce, P McMullan, and B McCollum. Directing selection within an extended great deluge optimization algorithm. In *Proc. of the 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA-13)*, pages 499–508, 2013.
9. John Michael Hammersley, David Christopher Handscomb, and George Weiss. Monte Carlo methods. *Physics today*, 18:55, 1965.

10. Holger H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.
11. D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989.
12. Roger Koenker. *Quantile regression.* Cambridge University Press, Cambridge, 2005.
13. Leo Lopes and Kate Smith-Miles. Pitfalls in instance generation for Udine timetabling. In *Learning and Intelligent Optimization (LION4)*, pages 299–302. Springer, 2010.
14. B McCollum, PJ McMullan, AJ Parkes, EK Burke, and S Abdullah. An extended great deluge approach to the examination timetabling problem. In *Proc. of the 4th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA-09)*, pages 424–434, 2009.
15. Barry McCollum, Paul McMullan, Edmund 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 (UK), September 2007.
16. Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.
17. R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2008.
18. Tommaso Urli. json2run: a tool for experiment design & analysis. *CoRR*, abs/1305.1112, 2013.

A Simulation Scenario Based Mixed Integer Programming Approach to Airline Reserve Crew Scheduling Under Uncertainty

Christopher Bayliss · Geert De Maere · Jason Atkin · Marc Paelinck

Abstract Airlines operate in an uncertain environment for many reasons, for example due to the effects of weather, traffic or crew unavailability (due to delay or sickness). This work focuses on airline reserve crew scheduling under crew absence and journey time uncertainty for an airline operating a single hub and spoke network. Reserve crew can be used to cover absent crew or delayed connecting crew. A fixed number of reserve crew are available for scheduling and each requires a daily standby duty start time. Given an airline's crew schedule and aircraft routings we propose a Mixed Integer Programming approach to scheduling the airline's reserve crew. A simulation of the airline's operations with stochastic journey time and crew absence inputs and without reserve crew is used to generate disruption scenarios for the $MIPSSM$ formulation (Mixed Integer Programming Simulation Scenario Model). Each disruption scenario corresponds to a record of all of the disruptions in a simulation for which reserve crew use would have been beneficial. For each disruption in a disruption scenario there is a record of all reserve crew that could have been used to solve or reduce the disruption. This information forms the input to the $MIPSSM$ formulation, which has the objective of finding the reserve schedule that minimises the overall level of disruption over a set of scenarios. Additionally, modifications of the $MIPSSM$ are explored, and a heuristic solution approach and a reserve use policy derived from the $MIPSSM$ are introduced. A heuristic based on the proposed Mixed Integer Programming Simulation Scenario Model or $MIPSSM$ outperforms a range of alternative

Christopher Bayliss · Geert De Maere · Jason Atkin
ASAP, University of Nottingham, UK
E-mail: cwb,gdm,jaa@cs.nott.ac.uk

Marc Paelinck
KLM Decision Support, Information services department
KLM Royal Dutch Airlines
KLM Headquarters, The Netherlands
E-mail: Marc.paelinck@klm.com

approaches. The heuristic solution approach suggests that including the right disruption scenarios is as important as ensuring that enough disruption scenarios are added to the $MIPSSM$.

Keywords Airline Reserve Crew Scheduling · Simulation · Mixed Integer Programming

1 Introduction

An airlines primary product is its schedule, due to operating costs airlines maximise the utilisation of resources (crew and aircraft) resulting in schedules with little slack. This makes each resource a critical component of an airlines network and if a component is missing all flights related to that component may be disrupted. Crew can be absent or delayed on connecting flights, in such circumstances airlines may call on reserve crew. This work focusses on reserve crew scheduling, using simulation generated disruption scenarios added to a Mixed Integer Programming model to schedule reserve crew.
A Mixed Integer Programming Simulation Scenario Model ($MIPSSM$) has been developed which will use information from repeat simulations of an airline network where reserve crew are not available. Then reserve crew are to be scheduled in such a way that the level of delay and cancellation that would have occurred in the original simulations (disruption scenarios) is minimised. Simulation (Section 3.3) is used to generate the set of input disruption scenarios for the $MIPSSM$. A disruption scenario corresponds to the set of disrupted flights in a single run of the airline simulation, where a single run corresponds to executing the airlines schedule in the considered time horizon from start to finish once. For each disruption in a disruption scenario there is a record of all of the reserve crew start times (discretised according to scheduled departure times) which, if scheduled, would allow the corresponding reserve crew to be used to solve completely, or reduce, the given disruption. In the $MIPSSM$ there are 2 types of variables, X the reserve crew schedule and y the reserve use decisions within each disruption scenario that are feasible with respect to X. Reserves can only be used if they are scheduled. Solving the $MIPSSM$ in an appropriate solver finds the reserve crew schedule X and reserve use decisions y that minimises delay and cancellations in the set of disruption scenarios used to form the constraints and objective of the $MIPSSM$.
The remainder of the paper is structured as follows. Section 2 outlines closely related work. Section 3 introduces the simulation used to generate disruption scenarios, how disruption scenarios are derived from the simulation and presents the formulation of the $MIPSSM$. Section 4 covers modifications and variants of the basic $MIPSSM$ formulation. Section 5 gives experimental results. Section 6 concludes the paper with a summary of the main findings. Section 7 discusses future work.

2 Related work

The $MIPSSM$ has similarities to Recoverable Robustness [4]. In [4] Liebchen provides a framework for timetabling problems with the objective that the schedule must be feasible in each of a limited set of disruption scenarios given limited availability of recovery from disruptions. The approach reduces to strict robustness (feasible in all outcomes without recovery actions) if the feature of limited available recovery is removed. The similarity between Recoverable robustness and the $MIPSSM$ lies in the idea of solving a scheduling problem over a limited number of realistic disruption scenarios. The $MIPSSM$ is influenced by stochastic programming, which optimises over a set of explicit independent possible outcomes as opposed to optimising over the expected outcome, which may not even correspond to a possible outcome.
In [7], Bailey et al. present an airline reserve crew scheduling model that takes training days and bidline conflicts into account. Such conflicts arise when crew bid for rosters which overlap with recurring training and this leads to open time (flights without scheduled crew) which have to be covered with reserve crew. In [6], Shebalov tackles the robust airline crew pairing problem using the concept of move-up crews. Move-up crews refers to crews who can swap pairings in the event of delay (the available crew can adopt the delayed crew's pairing). Their objective is to maximise move-up crews. Shebalov measures the robustness of schedules/quality of the scheduled move-up crews in computational experiments in terms of the number of deadheads (crew transported as passengers to the origin of their next flight leg), reserve crew used, number of uncovered flight legs and the cost of crew schedule. For the interested reader other work carried out previously on the problem of airline reserve crew scheduling includes [2,3,5].

3 Deriving and formulating the $MIPSSM$

This section starts by introducing the notation, it then introduces the delay cancellation measure function (Section 3.2), which converts delays into a quantity with units of cancellations. This approach means the $MIPSSM$ remains a single objective problem. The cancellation measure function is used in the disruption scenario generating simulation (Section 3.5) to find the cancellation measures associated with all possible reserve crew start times that if scheduled could be used to solve or reduce a given disruption in the given disruption scenario. Section 3.3 gives details of the single hub airline simulation used for disruption scenario generation and (in Section 5) experimental validation of reserve crew schedules derived from the $MIPSSM$ as well as other methods. Section 3.5 defines what is meant by a disruption scenario and how the information it stores is collected from simulation. Section 3.6 defines the notation used in the $MIPSSM$ formulation. Section 3.7 presents and explains the $MIPSSM$ in terms of its objective and constraints.

3.1 Schedule notation

D_h :	Scheduled departure time of flight h
C_h :	Crew team number scheduled to flight h
A_h :	Aircraft number scheduled to flight h
cd_h :	Crew related delay at departure h that occurs in disruption scenario generating simulation
$rd_{h,l}$:	Delay when reserve crew with start time index l used to cover disrupted crew of flight h
td_h :	Total delay at departure h
$crewSize_h$:	number of crew in crew team scheduled to flight h
$ceta_h$:	Estimated time of arrival of crew scheduled to flight h
$aeta_h$:	Estimated time of arrival of aircraft scheduled to flight h
CT :	Cancellation threshold over which delayed flights are cancelled
MS :	Minimum sit or minimum rest time required by crew between consecutive flights within a duty shift
TT :	Minimum turn/ground time required by aircraft between consecutive flights
$\lvert P_n \rvert$:	Length of crew pairing n in terms of hub departures
$P_{n,m}$:	Departure number of the m^{th} hub departure of crew pairing n

3.2 Cancellation measure of a delay

To retain the simplicity of a single objective problem Equation 1 converts delay into a measure of cancellation. The simulation cancels flights with a delay over the cancellation threshold so the maximum cancellation measure of a delay is 1. cm_h is the cancellation measure of flight h, td_h (Equation 2) is the total delay of flight h, cd_h (Equation 3) is the delay of flight h due to crew over and above delay due to the aircraft, i.e. the delay which could be absorbed by using reserve crew. Equation 4 gives the delay due to waiting for reserve crew with start time index l (start time=D_l as reserve start times are discretised according to scheduled departure times) to begin their duty shift, counting only delay over and above delay due to the aircraft assigned to the same flight.

$$cm_h = \left(\frac{td_h - cd_h}{CT}\right)^n \tag{1}$$

$$td_h = \max(0,\ \max(aeta_h + TT,\ ceta_h + MS) - D_h) \tag{2}$$

$$cd_h = \max(0,\ ceta_h + MS - \max(D_h,\ aeta_h + TT)) \tag{3}$$

$$rd_{h,l} = \max(0,\ D_l - \max(D_h,\ aeta_h + TT)) \tag{4}$$

A decision maker choice is required for the delay exponent n of the cancellation measure function. Choosing higher values for $n > 1$ corresponds to giving lower weight to delays below the cancellation threshold. Using the delay cancellation measure function means that the objective measures of using reserve crew teams to cover delayed connecting crew and using reserve crew to cover absent crew are both in the same units, that of cancellations. In the following $n = 2$ is used.

3.3 Simulation

The simulation of a single hub airline is used without reserve crew to generate disruption scenarios which contain information on the possible benefit of using reserve crew scheduled at specified times in response to the given disruption. These disruption scenarios form the input for the $MIPSSM$ formulation (Section 3.7).
Simulation takes as input the airline's scheduled flights, the crew and aircraft scheduled to each of those flights. The simulation's stochastic inputs are journey times and crew absence, each of which have corresponding statistical distributions derived from real data. Crew and aircraft were scheduled using first in first out scheduling. In the crew schedule 30% of crew connections at the hub involve a change of aircraft. The scheduled journey times correspond to a 0.6 probability of early arrival.
A single run of the simulation proceeds by considering each scheduled departure in departure time order. If a departure corresponds to the start of a crew duty then the number of crew absent is instantiated from the cumulative statistical distribution of possible numbers of absent crew. If reserve crew are not available then the flight has to be cancelled. At this point in the simulation, information on the possible benefit of scheduling reserves at different start times is collected (Section 3.5). If reserve crew are available (as is the case in the validation simulation used in Section 5 to validate reserve crew schedules created using the methods proposed herein) they are considered for use in earliest start time order. If a departure is delayed by more than the delay threshold (15 minutes) all combinations of single crew and aircraft swaps are considered. Swaps are only considered feasible if the swap can take place without invoking additional delay on the flights affected by the swap, the crew must be able to complete each other's duties without violating maximum working hours and it must be possible to undo the swap in the overnight break (same overnight station).
In the disruption scenario generation simulation, after the consideration of swap recovery actions, if the delay is still above the delay threshold, information is collected for the given disruption on the possible benefit of scheduling reserve crew at different possible start times (Section 3.5). In the validation simulation, after the consideration of swap recovery actions, possible combinations of reserve crew are considered for replacing delayed connecting crew. If after delay recovery the delay is above the cancellation threshold (180 minutes) the flight is cancelled.

3.4 Disruption scenario notation

W :	Number of disruption scenarios
W_i :	Number of disruptions in scenario i
$N_{i,j}$:	The number of reserve crew required to cover disruption j in scenario i
$CM_{i,j}$:	Cancellation measure contribution when no reserves are used to cover disruption j in scenario i
$N_{i,j}$:	Number of crew required to cover disruption j in scenario i
$F_{i,j}$:	Set of feasible reserve instances for disruption j in scenario i
$F_{i,j,k}$:	k^{th} instance of a reserve feasible to cover disruption j in scenario i
$F^V_{i,j,k}$:	k^{th} reserve use variable index feasible for disruption j in scenario i
$F^U_{i,j,k}$:	k^{th} index of reserve use variable first used at disruption j in scenario i which can subsequently be used to absorb crew related delay propagated to a following flight
$F^{CM}_{i,j,k}$:	Cancellation measure that occurs as a result of using the k^{th} feasible reserve for disruption j in scenario i
$F^{RD}_{i,j,k}$:	reserve delay corresponding to feasible reserve use instance k feasible for disruption j in scenario i
$G_{i,j}$:	Set of feasible reserve instances corresponding to reserve crew first used to absorb delay on a preceding flight that also have the knock-on effect of preventing or reducing delay disruption j in scenario i
$G_{i,j,k}$:	k^{th} instance of a reserve feasible corresponding to a reserve first used to absorb crew delay on a preceding flight that also has the knock-on effect of reducing delay disruption j in scenario i
$G^V_{i,j,k}$:	k^{th} reserve use variable index corresponding to a reserve first used to absorb delay on a preceding flight that has the knock-on effect of reducing delay disruption j in scenario i
$G^{CM}_{i,j,k}$:	Cancellation measure corresponding to the k^{th} feasible reserve use instance first used to absorb delay on a preceding flight that also has the knock-on effect of reducing delay disruption j in scenario i
$R_{i,k}$:	Set of feasible reserve use variable instances corresponding to reserve k in scenario i
$R_{i,k,l}$:	Reserve use variable index corresponding to the l^{th} reserve use variable corresponding to reserve k in scenario i

3.5 Simulation derived scenarios

Simulation is used to derive disruption scenarios that are used as input for the $MIPSSM$. This section explains how simulation is used to derive the information for disruption scenarios. A given disruption scenario i corresponds to a single run of the simulation.
In disruption scenario i, a disruption j is a flight which has a delay over the delay threshold after the consideration of swap recovery or has to be cancelled due to crew absence. Such disrupted flights have a positive cancellation measure, where $CM_{i,j}$ denotes the cancellation measure of disruption j in disruption scenario i.

In a given run of the simulation, when a disruption occurs with a positive cancellation measure, data is collected regarding all of the possible feasible reserve start times that, could be used to reduce the disruption. For each such beneficial reserve start time, feasible reserve use instances are generated. A feasible reserve use instance corresponds to a possible scheduled reserve crew duty start time and subsequent use to cover a given crew disrupted flight in a given scenario. The number generated is equal to the number of reserve crew required to cover the given disruption, which is the number of crew absent in the event of a crew absence disruption or the size of the crew team assigned to flight h ($crewSize_h$) in the event of a delay. For each feasible reserve use instance (b) there is a corresponding cancellation measure (b^{CM}) that replaces the cancellation measure ($CM_{i,j}$) of the disruption if the reserve is used, a unique reserve use variable index (b^V), a unique knock on effect reserve use variable index (b^U) (if applicable) and a reserve delay (b^{RD}). Let $F_{i,j}$ denote the set of feasible reserve use instances corresponding to possible reserve start times that could be used to solve or reduce disruption j of disruption scenario i.

For the specific case of delay disruptions it is also possible that if there was crew delay on the preceding flight then the delay on the current flight might possibly be prevented or reduced by reserve crew used to absorb the initial delay. For this purpose the set $G_{i,j}$ is introduced and denotes the set of feasible reserve use instances corresponding to reserves used to cover crew related delay propagated from a previous previous flight. These feasible reserve use instances only apply if the corresponding feasible reserve use instances are used to cover the root crew related delay. G accounts for reserve crew that can have the effect of absorbing knock on crew delays. Algorithms 1 and 2 outline the procedure of collecting information for the disruption scenarios from the single hub airline simulation.

Algorithm 1 Pseudocode for deriving disruption scenario information for a crew absence disruption occurring at simulation run i departure k

```
1: During simulation run $i$ the $k^{th}$ scheduled flight is disrupted resulting in the $j^{th}$ disruption for which reserve crew use could potentially be beneficial
2: if Crew absence disruption then
3:     Create new disruption ($j$) for scenario ($i$), store the size of the disruption if not absorbed by utilising reserve crew, i.e. The number of flights cancelled=size of pairing ($|P_{C_k}|$) and store the number of crew absent ($N_{i,j}$)
4:     for each hub departure ($m$) in the crew absence disrupted pairing do
5:         for each reserve duty start time ($l$) feasible to cover the absence disrupted pairing at the $m_{th}$ hub departure of the disrupted pairing do
6:             Add $N_{i,j}$ new feasible reserve use instances to $F_{i,j}$, each with a unique variable number index ($V$), compute the associated cancellation measure ($CM$), Add the generated feasible reserve use instances to $R_{i,l}$ for the constraints regarding reserves only being used once per scenario.
7:         end for
8:     end for
9:     $j = j + 1$
10: end if
```

Algorithm 1 is used in the simulation when a crew absence occurs. The number of reserves required to cover this disruption is the number of absent crew (line 3). The cancellation measure of the absence disruption $(CM_{i,j})$ is the number of hub departures in the disrupted crew pairing that would have to be cancelled if reserves are unavailable to cover the absent crew (line 3), with no delay cancellation measure contribution. The algorithm then considers each possible reserve start time (line 5) which can be used to cover absent crew at each hub departure in the disrupted crew pairing (line 4). If reserve start time l is feasible, $N_{i,j}$ new instances of feasible reserve use instances are created with unique reserve use variable indices and cancellation measures equal to the number of flights that have to be cancelled before crew absence is covered at the m^{th} hub departure in the disrupted crew pairing plus a delay cancellation measure contribution from any delay caused by the reserve start time (lines 6). $cm = m - 1 + \left(\frac{rd_{f,l}}{CT}\right)^n$ is the equation for the cancellation measure associated with reserve crew with start time index l being used to cover crew absence disrupted pairing at the m_{th} flight in the crew pairing, where $f = P_{C_k,m}$ is the departure number of the flight the reserve crew are used to cover the absence disrupted crew pairing. The newly generated instances of feasible reserve use are also stored from a reserve perspective (R) (line 6), which is useful later on when creating constraints for feasible reserve use in the $MIPSSM$ formulation.

Algorithm 2 Pseudocode for deriving disruption scenario information for a crew delay disruption occurring at simulation run i departure k

```
1: During simulation run $i$ the $k^{th}$ scheduled flight is disrupted resulting in the $j^{th}$ disruption for which reserve crew use could potentially be beneficial
2: if crew delay disruption then
3:     Store the number of delayed connecting crew that need to be replaced ($N_{i,j} = crewSize_k$) and the cancellation measure of the delay $CM_{i,j} = \left(\frac{td_k}{CT}\right)^n$
4:     for Each reserve start time index $l$ that if scheduled could feasibly reduce the crew related delay of departure $k$ do
5:         Generate $N_{i,j}$ reserve use instances with unique reserve use variable indices ($V$), store the corresponding cancellation measure that applies if the reserves with start time $D_l$ are used, add the reserve use instances to $R_{i,l}$
6:     end for
7:     if Current crew delay is crew delay propagated from the crew's previous flight $q$, disruption $o$ then
8:         for $l = 1$ to $|F_{i,o}|$ do
9:             Create new reserve use instance, with unique reserve use variable index ($V$) and store in $G_{i,j}$, store the corresponding cancellation measure that applies if the root crew delay is absorbed using the reserve associated with the reserve use instance $F_{i,o,l}$
10:            $F^U_{i,o,l} = G^V_{i,j,a}$, where $a = |G_{i,j}|$
11:        end for
12:    end if
13:    $j = j + 1$
14: end if
```

Algorithm 2 is used in the simulation when a crew related delay occurs, the number of reserves required to cover this disruption is the number of crew in the delayed crew team (line 3). The cancellation measure of the delay disruption if reserve crew are not available to cover the delayed crew is also computed (line 3). The algorithm then considers each feasible reserve start time (line 4) used to cover the delay, and for each generates $N_{i,j}$ new feasible reserve use instances with unique reserve use variable indices and cancellation measures calculated using Equation 1 with Equation 4 added to the numerator. Lines 7 to 12 of Algorithm 2 apply if the given delay originated from a crew delay in the scheduled crew's previous flight. In this case it's possible that reserve use instances generated for that previous flight may have the effect of preventing delay propagating to the given delayed flight. For such feasible reserve use instances (line 8) U denotes new unique reserve use variable index for the reserve first used earlier, used to reduce the knock-on delay. For the current disruption j the set G stores the same newly generated reserve use variable index and a cancellation measure (line 9) that depends on the amount of delay that would have propagated if the reserve use instance feasible to cover the root crew delay is utilised. The $MIPSSM$ has constraints ensuring that the beneficial knock on effects can only apply if the reserve is actually used to absorb the root crew delay that propagated in the simulation. After the disruption scenarios have been created they can be used to create the constraints and objective of the $MIPSSM$.

3.6 $MIPSSM$ notation

X :	Reserve crew schedule
x_k :	Number of reserves with start time index k
Y :	Set of reserve use variables
y_m :	Reserve use instance variable m
$\delta_{i,j}$:	Binary variable describing whether or not disruption j in scenario i is left uncovered (1) or covered (0) by reserve crew
$\gamma_{i,j}$:	Real valued variable which takes on the cancellation measure of disruption j in scenario i given the reserve recovery decision made by the model
Z :	Variable that takes on a value equal to the cancellation measure total of the scenario with the maximum cancellation measure
TR :	Total reserve crew available for scheduling
ND :	Total flights in reserve crew scheduling time horizon
Rt_q :	Reserve use policy, the minimum threshold number of reserve crew remaining for using a team of reserve crew to cover a delayed connecting crew to be considered acceptable
obs_q :	Number of times reserve teams are used to cover delayed connecting crew at flight q in reserve use policy derivation
$simRpts$:	Number of repeat simulations used to derive a reserve use policy for a given reserve crew schedule

The reserve schedule X specifies the number of reserves which are scheduled to begin duties at a given time index k. $\gamma_{i,j}$ is a real valued variable, which equals the cancellation measure of disruption j in scenario i given the reserve use decisions (y). Each y variable corresponds to an individual reserve with a given start time index being used to cover a given disruption.

3.7 Mixed Integer programming formulation

Minimise:

$$\sum_{i=1}^{W}\sum_{j=1}^{W_i}\gamma_{i,j} \tag{5}$$

s.t.

$$\sum_{k=1}^{|F_{i,j}|} y_{F^V_{i,j,k}} + \sum_{k=1}^{|G_{i,j}|} y_{G^V_{i,j,k}} + \delta_{i,j}N_{i,j} = N_{i,j},\ \forall i \in 1..W,\ \forall j \in 1..W_i \tag{6}$$

$$\sum_{i=1}^{ND} x_i = TR \tag{7}$$

$$\sum_{l=1}^{|R_{i,k}|} y_{R^V_{i,k,l}} \leq x_k,\ \forall k \in 1..ND,\ \forall i \in 1..W \tag{8}$$

$$y_{R^U_{i,k,l}} \leq y_{R^V_{i,k,l}},\ \forall l \in R_{i,k} | \exists y_{R^U_{i,k,l}},\ \forall i \in 1..W,\ \forall k \in 1..ND \tag{9}$$

$$\delta_{i,j}CM_{i,j} \leq \gamma_{i,j}, \forall i \in 1..W,\ \forall j \in 1..W_i \tag{10}$$

$$y_{F^V_{i,j,k}} F^{CM}_{i,j,k} \leq \gamma_{i,j}, \forall i \in 1..W,\ \forall j \in 1..W_i,\ \forall k \in F_{i,j} \tag{11}$$

$$y_{G^V_{i,j,k}} G^{CM}_{i,j,k} \leq \gamma_{i,j}, \forall i \in 1..W,\ \forall j \in 1..W_i,\ \forall k \in G_{i,j} \tag{12}$$

$$y_m \in \{0,1\},\ \forall m \in Y \tag{13}$$

$$\delta_{i,j} \in \{0,1\},\ \forall i \in 1..W,\ \forall j \in 1..W_i \tag{14}$$

$$X_k \in \{0, 1...maxCA_i - 1, maxCA_i\},\ \forall k \in 1..ND \tag{15}$$

Objective 5 minimises the sum of all cancellation measures over all disruptions in all the scenarios included in the model. Constraint 6 ensures that disruptions are only considered covered if the required number of reserves are used for the given disruption. Constraint 6 forces $\delta_{i,j}$ to 1 when no reserve recovery can be applied to disruption j in scenario i and to 0 otherwise. Constraint 6 means that it is acceptable to cover a crew delayed departure with a combination of reserves used now and reserves used to cover a preceding crew delay that propagated, which may be useful if some of the reserves used to cover the root delay are not feasible to cover the following flight. Constraint 7 ensures that no more than the total number of reserves available (TR) are scheduled. Constraint 8 ensures that in each disruption scenario the number of reserves used with the same start time index does not exceed the number of reserves which are scheduled to that start time index. Constraint 9 ensures that disruptions can only be absorbed by reserves which were first used to absorb delay on the preceding flight if the reserve is used to cover that preceding flight. Constraints (10 to 12) ensure that the cancellation measure associated with a given disruption is the maximum of that associated with the recovery actions used for the given disruption. If no reserves are used for a given disruption that disruption gets the cancellation measure CM_{ij}, the same as occurred in the simulation in which the disruption occurred. If reserves are used the cancellation measure corresponds to the reserve used for that disruption that invokes the largest cancellation measure (as the flight can't take off before all the crew are present). Constraints 13 to 15 are integrality constraints.

4 Variants and Modifications

This section firstly considers 2 alternative formulations of the basic $MIPSSM$ formulation given in equations 5 to 15. Then a scenario selection heuristic designed to address the question of whether the types of scenarios or the number of scenarios included in the formulation has the greatest effect on solution quality. The final part in this section introduces an approach for deriving an optimal reserve use policy for a given reserve schedule, by repeated solving of the $MIPSSM$ for a single disruption scenario and fixed reserve schedule and learning the circumstances in which reserve use is beneficial in the long run.

4.1 Alternative objectives for the $MIPSSM$

Several alternative objectives are suggested in this section.

$MiniMax1$

The objective of minimising the sum of cancellations measures over all disruption scenarios included in the model (Objective 5) could be replaced with the alternative objective $MiniMax1$ of minimising the sum of cancellation measures of the disruption scenario with the largest sum of cancellation measures. This is a minimax objective function and can be implemented by replacing

Objective 5 with Objective 16 and adding Constraint 17. Information on implementing minimax objectives in linear programs can be found in [8]. This approach will have the effect of finding a reserve crew schedule that minimises the extent of the worst case scenario as opposed to minimising the average cancellation measure. In Table 1 the *probability of delay over 30 minutes* performance measure is most relevant to the $MiniMax1$ formulation.

$$\text{min: } Z \tag{16}$$

$$\sum_{j=1}^{W_i} \gamma_{i,j} \leq Z, \, \forall i \in 1..W \tag{17}$$

$MiniMax2$

Instead of minimising the total cancellation measure of the disruption scenario with the largest cancellation measure, the same principle can be applied to individual scenarios with the alternative Objective $MiniMax2$. I.e. find the reserve crew schedule that minimises the single largest disruption. To implement this approach replace Constraint set 17 with Constraint set 18. In Table 1 there is no performance measure which is directly relevant to the $MiniMax2$ formulation because in the reserve crew schedule validation simulation the worst single disruption is a cancellation and these will inevitably occur in each method. However in the $MiniMax2$ formulation the worst single disruption is leaving an absence disruption uncovered which would result in all flights on the absent crew's line of flight being cancelled.

$$\gamma_{i,j} \leq Z, \, \forall i \in 1..W, \, \forall j \in 1..W_i \tag{18}$$

4.2 Alternative solution approach

Scenario Selection Heuristic

The basic $MIPSSM$ and the two alternative formulations $MiniMax1$ and $MiniMax2$ are solved over a given set of disruption scenarios in a linear programming solver (CPLEX in this case). Although CPLEX yields optimal solutions, the solutions are only optimal for the set of disruption scenarios considered in the model. This section introduces a scenario selection heuristic (SSH) to address the issue of the choice of scenarios included in the $MIPSSM$. The solution time increases sharply as the number of disruption scenarios increases, which provides another motivation for considering a scenario selection heuristic solution approach, which includes the right scenarios rather than ensuring that enough disruption scenarios are included in the model. The following heuristic for solving the model defined by Equations 5 to 15 is based on adding one disruption scenario to the model at a time and stopping only when a new disruption scenario cannot be found for which the incumbent solution (overall reserve schedule) performs worse than in the worst disruption scenario already added to the model. The heuristic is analogous to column generation in which the master problem and pricing problem are solved iteratively. Repetition ceases when a specified iteration limit ($itLim$) is reached (in which case the whole algorithm terminates and returns a final solution), or, alternatively, when no new scenario with a sub problem ($subObj$) objective value can be found that is larger than the scenario already in the master problem with the largest objective contribution ($\max_j(masterObj_j)$), after $rptLim$ attempts. In summary this scenario selection approach focusses on finding a reserve schedule that can cope with a wide variety of difficult scenarios as opposed to a random set of scenarios representing the average outcome. An outline of the scenario selection heuristic is given in Algorithm 3.

Algorithm 3 Psuedocode for the scenario selection heuristic

```
1: $newScenarioFound = true$
2: $its = 0$
3: while $newScenarioFound \wedge its \leq itLim$ do
4:     $newScenarioFound = false$
5:     $rpts = 0$
6:     while $\neg newScenarioFound \wedge rpts < rptLim$ do
7:         Run simulation to generate disruption scenario $newScenario$
8:         Solve new scenario subproblem
9:         if $subObj > \max_j(masterObj_j)$ then
10:            $newScenarioFound$ =true
11:            add new scenario to the master problem
12:        else
13:            $rpts = rpts + 1$
14:        end if
15:        if $newScenarioFound$ then
16:            resolve master problem
17:        end if
18:    end while
19:    $its = its + 1$
20: end while
21: return solution
```

4.3 Optimal reserve use policy derivation

The simulation (Section 3.3) when used to test reserve schedules has a default policy of using reserve crew whenever this is immediately beneficial. The default policy also uses reserve crew in earliest start time order, so as to leave the largest amount of unused reserve crew capacity available for subsequent disruptions. The $MIPSSM$ approach uses reserve crew in each disruption scenario in an optimal way based on full knowledge of future disruptions. Knowledge of future disruptions is not available in the simulation, if a scenario which was included in the $MIPSSM$ formulation is repeated in the validation simulation, reserves might not necessarily be used in the same optimal way.

In this section an algorithm for deriving an optimal reserve use policy is described. The policy is based on reserve use decisions in response to delayed crew, where a team of reserve crew could be constructed and used to absorb the delay. The policy consists of threshold minimum numbers of reserve crew remaining for each departure for which using reserve teams to absorb crew related delay is deemed globally beneficial. The threshold numbers of reserves remaining are derived from the simulation by solving for the reserve use variables of the $MIPSSM$ model for the given reserve schedule over the disruptions that occur in the given run of the simulation and then averaging the number of reserves remaining at times when the $MIPSSM$ model uses reserve crew to cover for delayed crew.
The default policy is used for reserve crew use in response to crew absence since the penalty for not replacing absent crew with reserves is far too high (cancellation) to consider a crew absence reserve holding policy, and the cost of using teams of reserves to cover delayed crew is too high if this leaves too few reserve crew to cover subsequent absences. In general using teams of reserve crew to cover delayed connecting crew is expensive as it solves a smaller disruption (a delay compared to a cancellation) using more reserves than are usually required to cover absent crew.

5 Experimental results

The reserve crew scheduling approaches $MIPSSM$ of Section 3.7, $MiniMax1$ and $MiniMax2$ of Section 4.1 and SSH of Section 4.2 are tested and compared to one another. IBM CPLEX Optimization Studio version 12.5 with Concert technology is used as the MIP solver, on a desktop computer with a 2.79GHz Core i7 processor. These methods are also compared to a range of alternative methods for reserve crew scheduling (described below).

5.1 methods

Probabilistic reserve crew scheduling under uncertainty

The probabilistic approach ($Prob$) to reserve crew scheduling is an application of the work by the same authors in [1]. Given knowledge of the probabilities of crew absence for each flight in an airline's schedule, the probabilistic model evaluates the effect a given reserve crew schedule has on the probabilities of cancellations due to crew absence. The solution space of reserve crew schedules is then searched to find the reserve crew schedule that minimises the probabilities of cancellations due to crew absence. It was found that constructive heuristics provide near optimal solutions when solving the model put forward in [1]. The work in [1] has been extended to account for different numbers of crew being absent from each crew pairing in an airline schedule. Moreover the constraint that reserve crew are only feasible for disruptions if their duty start time is no later than the scheduled departure time of the disrupted flight has been relaxed so that some reserve delay is permitted, just as in the $MIPSSM$. Reserve delays in the probabilistic approach are accounted for using the delay cancellation measure function (Equation 1).

Area Under the Graph

The Area Under the graph ($Area$) method is based on running a number of simulations and recording the cumulative demand for reserve crew with respect to time in the form of a bar chart (in terms of the cancellation measure that could be avoided if reserve crew were available). Reserve crew are then scheduled at equal area intervals under the reserve demand graph over the whole scheduling time horizon. The $Area$ approach is based on a simulation without reserve crew to find reserve demand independent of the effects of a reserve crew schedule.

Uniform start rate

The Uniform Start Rate method (USR) schedules reserve crew at equal time intervals.

Zeros

The Zeros method ($Zeros$) schedules all reserve crew to begin standby duties at the first departure of the first day.

5.2 Experiment design

The methods stated in Section 5.1 are each solved for a synthetic airline schedule, synthetic in that the schedule is designed to increase the chance of delays due to delayed connecting crew. Other than this the schedule is fully detailed in terms of crew pairings and aircraft routings. Journey time uncertainty is modelled by statistical distributions based on real data, crew absence uncertainty is modelled as each individual scheduled crew member having a 1% chance of being absent and missing their entire crew pairing. All teams of crew consist of 4 individuals with identical rank (primarily aimed at cabin crew, but extending also to technical crew). The schedule is based on a 3 day single hub airline schedule with 243 flight legs a day with half being from the hub station. The schedule uses 148 teams of crew scheduled and 37 aircraft (single fleet). The schedule was generated using a first in first out approach with stochastic parameters controlling the rate of crew aircraft changes (0.3) and the 60^{th} percentile journey time from each destination's cumulative journey time distribution. These parameters influence the likelihood of delay propagation and the occurrence of delayed connecting crew. The following experiments investigate the effect of the number of reserve crew available for scheduling for each solution approach, and for the $MIPSSM$ based approaches the effect of the number of input disruption scenarios on solution time and solution quality.

5.3 Investigating the effect of varying the number of reserve available for scheduling

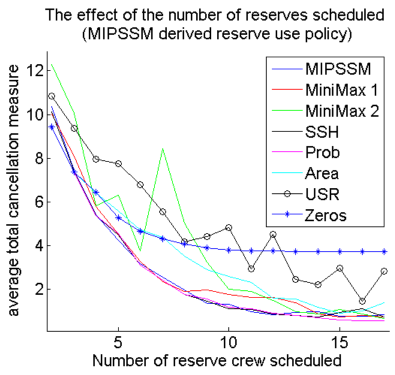


Fig. 1 The effect of the number of reserves which are scheduled on the solution quality of different solution approaches

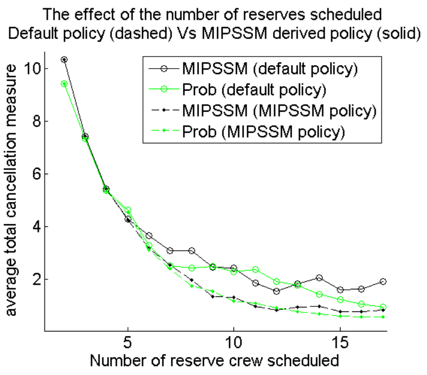


Fig. 2 The effect of the $MIPSSM$ derived reserve use policy

The results in Figure 1 show the effect on the average cancellation measure of varying the number of reserve crew available for scheduling, using 20000 repeat validation simulation tests for each reserve crew schedule for each solution approach. The $MIPSSM$ based approaches are restricted to 50 input disruption scenarios and a maximum of 1 hour to find a solution.
Figure 1 shows how the various reserve crew scheduling approaches compare for different numbers of reserve crew available for scheduling.
The $MIPSSM$, SSH and $Prob$ obtain the lowest average cancellation measures for all numbers of reserve crew available for scheduling of those tested. The $Prob$ model gives a smooth curve of average cancellation measures, whereas $MIPSSM$ and SSH have small fluctuations in average cancellation measure as the number of reserve crew available for scheduling changes. This fluctuation can in part be attributed to the limited number of disruption scenarios used as input for these methods. The $MiniMax1$ modification generally lead to higher average cancellation measures especially when between 9 and 12 reserve crew were available for scheduling. $MiniMax2$ gave the unexpected result that scheduling more reserve crew can lead to a higher average cancellation measure. This fluctuating behaviour of the $MiniMax2$ modification was also observed to a lesser extent in the other methods based on the $MIPSSM$ and can be explained by the fact that the objective of the $MiniMax2$ modification is to suppress the single largest delay or cancellation disruption that can

Method name	**Average cancellation measure**	**Average delay**	**Probability of delay > 30mins**	**Cancellation rate**	**Reserve Utilisation rate**	**solution time /mins**
$NoRes$	15.009	11.147	0.00682	0.03925	-	-
$MIPSSM$	1.159	12.180	0.00898	0.00140	0.674	28.688
$MiniMax1$	1.246	12.393	0.00938	0.00154	0.666	7.060
$MiniMax2$	1.724	13.874	0.01114	0.00171	0.656	2.259
SSH	1.066	11.870	0.00871	0.00141	0.667	2.871
$Prob$	1.077	11.518	0.00818	0.00166	0.690	0.443
$Area$	2.399	14.001	0.01130	0.00353	0.589	0.060
USR	2.925	14.970	0.01336	0.00438	0.555	<0.001
$zeros$	3.756	11.167	0.00725	0.00902	0.571	<0.001

Table 1 Performance measure averages from 20 repeats

occur and is not to minimise the average cancellation measure. This fluctuation is due to the resultant schedules being designed for worst case disruptions as opposed to the average outcomes. The $Area$ under the graph approach lead to average cancellation measures similar to those from the $MiniMax2$ modification without the fluctuations. The USR approach lead to the highest average cancellation measures when 10 or fewer reserve crew are available for scheduling, for more than 10 reserve crew the $zeros$ approach gave the highest cancellation measures. The $zeros$ approach also gave the best results when fewer than 4 reserve crew were available for scheduling, this is because most crew absences are realised at the start of the first day, so scheduling reserve crew at that time prevents cancellations due to crew absence from the outset. The difference between the various solution approaches is clearest when there are around 10 to 12 reserve crew available for scheduling, which also appears to be the most sensible number of reserve crew to schedule (due to diminishing returns). Between 10 and 12 reserve crew, Figure 1 shows that the best performing solution approach was the SSH. 10 to 12 reserves for the given problem instance is approximately proportionate to the number of reserve crew scheduled in reality.
Figure 2 shows the effect of using the $MIPSSM$ derived reserve use policy described in Section 4.3 compared to the default policy of using reserve crew as demand occurs. Using the $MIPSSM$ derived policy had the effect of reducing the average cancellation measure.

5.4 Other performance measure and solution stability comparison of methods

Table 1 gives average performance measures when each method is applied to the same problem instance 20 times, for the $MIPSSM$ approaches the simulation generated scenarios differ in each of the 20 repeats as they start with a different random seed. The results show that on average the $MIPSSM$ performs best on cancellation rate, however the $MIPSSM$ is also the slowest method with average solution times of an hour. The average cancellation measure can be interpreted as the number of cancellations expected in each three

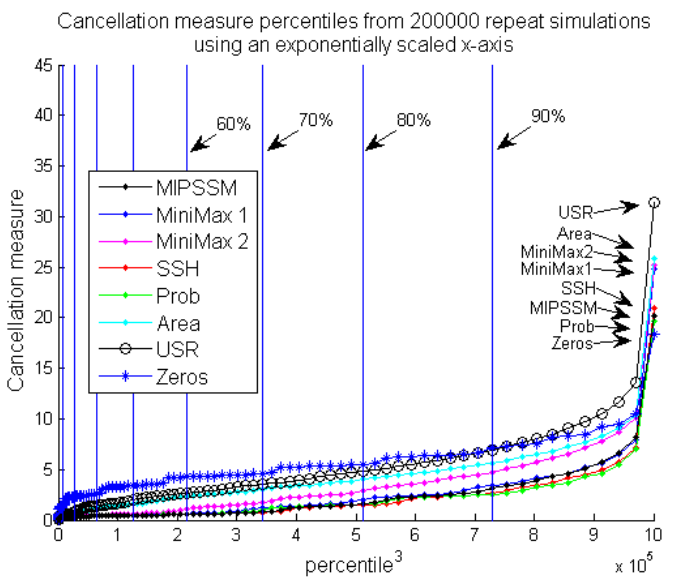


Fig. 3 Percentile cancellation measures

day simulation, but this also includes delays which have been converted to a cancellation measure using Equation 1 of Section 3.2. In terms of all round performance the SSH is a highly efficient approach with the lowest cancellation measure and also, a low average delay, the SSH is also much faster than the $MIPSSM$ with an average solution time of just under 3 minutes. The solution time of the SSH is a result of the termination criteria being satisfied before more than 10 disruption scenarios are added to the master problem. The $Prob$ approach has the second highest average cancellation measure, good average delay performance and a solution time much quicker the those of the $MIPSSM$ based approaches.

The results in Table 1 suggest there is merit in both the probabilistic and $MIPSSM$ approaches to scheduling airline reserve crew under uncertainty. Table 1 also includes performance measures when no reserve crew are scheduled at all as a point of reference. Counter to expectation the probability of delay over 30 minutes is lower without reserve crew, as is the average delay, however this can be attributed to the high cancellation rate, since cancelled flights do not count as delays. The Objectives $MiniMax1$ and $MiniMax2$ are aimed at minimising worst case scenarios, however if the probability of delay over 30 minutes is treated as a measure of worst case scenarios, it does not support this. Reserve utilisation rates are also given in Table 1 and are loosely correlated with the cancellation measures.

Figure 3 displays cancellation measure percentiles, the 100^{th} gives the worst cancellation measure from each approach, and this is the most appropriate validation criteria for the $MiniMax2$ modification. The $MiniMax2$ modification does not have the lowest cancellation measure for the 100^{th} percentile,

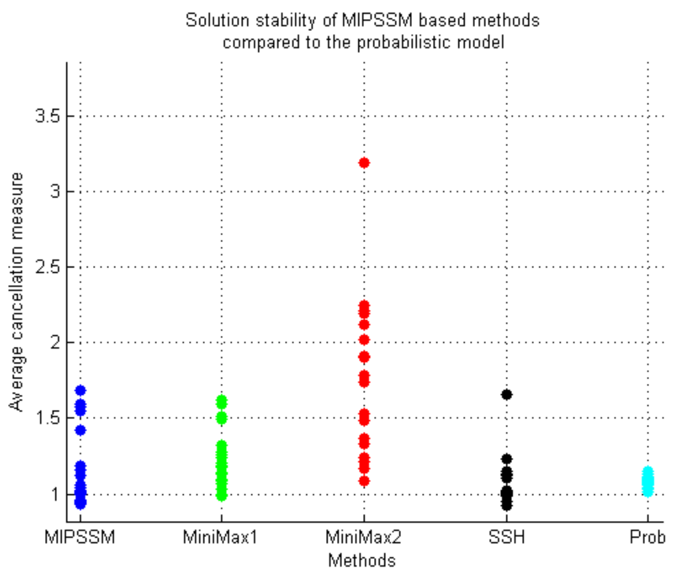


Fig. 4 Solution stability of $MIPSSM$ based methods compared to $Prob$

so it appears that this modification does not achieve its objective. Figure 3 shows the spread of cancellation measures corresponding to each method over the 20 repeats of each method, with each being tested in 20000 repeat validation simulations. Figure 3 demonstrates that for each given percentile the ordering of the methods supports the results given in Table 1 except for the $zeros$ approach which has the lowest worst case cancellation measure. This result suggests that the worst scenario is, for a very large number of crew to be absent at the start of each day, which is precisely the situation the $zeros$ approach can cope with. Figure 4 shows that the $MIPSSM$ based methods have a solution stability issue. Each point on Figure 4 represents a solution to the given method starting from a different random seed in the simulation used to generate the set of disruption scenarios over which the method is solved. Figure 4 shows that the $MIPSSM$ based methods have the potential to give solutions of higher quality that the Probabilistic method, but this depends on the selection of disruption scenarios which used as input for the given $MIPSSM$ based method.

6 Conclusion

In conclusion, a simulation based mixed integer programming approach to airline reserve crew scheduling has been introduced. The main idea is to schedule reserve crew using information from repeat simulations of an airline network where reserve crew are not available, and then scheduling reserve crew in a hindsight fashion in such a way that had they been available, the level of

delay and cancellation that was related to disrupted crew would have been minimised. The $MIPSSM$ formulation also took potential knock-on delays into account.
Another feature of the general approach is that it does not depend on the details of the simulation of the airline (i.e. the simulation is almost a black box). In the example problems the simulation had the capability to recovery from delays using airline resource swaps, such swaps were applied before disruption scenario data was derived. In effect the approach would work with any airline schedule simulator, provided the assumption that the use of reserve crew is a last resort recovery action is valid.
The SSH approach showed that the individual scenarios included in the model is at least as important as the number of scenarios, as this heuristic scenario selection approach yielded solutions of higher quality on average compared to the $MIPSSM$ approach, with only a fraction of the input disruption scenarios. The Probabilistic model ($Prob$) represented an entirely different approach to the $MIPSSM$ and gave comparable results, suggesting both approaches have their own merits. In general it was found that the $MIPSSM$, SSH and $Prob$ approaches gave results that were very similar on average, however the $MIPSSM$ based approaches had lower solution stability from one run to the next due to the stochastic nature of these approaches, but significantly outperformed the $Prob$ approach in some cases.

7 Future work

The $MIPSSM$ based approaches rely on stochastic inputs, this is both the greatest strength and weakness of these approaches. Future work includes investigating how to increase solution stability by improving the process of selecting which disruption scenarios to include in the solution phase of the $MIPSSM$ based approaches.

References

1. Christopher Bayliss, Geert De Maere, Jason Atkin, and Marc Paelinck. Probabilistic Airline Reserve Crew Scheduling Model. In Daniel Delling and Leo Liberti, editors, *12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 25 of *OpenAccess Series in Informatics (OASIcs)*, pages 132–143, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
2. Alexandre Boissy. Crew reserve sizing. In *Agifors syposium.*
3. Jeffrey E. Dillon and Spyros Kontogiorgis. Us airways optimises the scheduling of reserve flight crews. *Interfaces*, 29(5):pp123, September/October 1999.
4. Christian Liebchen, Marco Lubbecke, Rolf H. Mohring, and Sebastian Stiller. Recoverable robustness. Technical report, August 2007.
5. Marc Paelinck. KLM cabin crew reserve duty optimisation. In *Agifors proceedings*, 2001.
6. Sergey Shebalov and Diego Klabjan. Robust airline crew pairing: Moveup crews. *Transportation science*, 2006.
7. Milind G. Sohoni, Ellis L. Johnson, and T. Glennn Bailey. Operational airlines reserve crew scheduling. *Journal of Scheduling*, 2006.
8. H. Paul. Williams. *Model building in mathematical programming.* Wiley, 2002.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A linear mixed-integer model for realistic examination timetabling problems

Lisa Katharina Bergmann · Kathrin Fischer · Sebastian Zurheide

Abstract An examination timetable has to satisfy a vast variety of requirements to be not only feasible, but also to be convenient to all parties involved. Many different aspects, as e. g. spreading of exams for students' convenience or fixing exams to certain days or rooms for teachers' convenience, have been discussed in the literature. However, there are no model formulations which take all aspects relevant for this work into account.

Therefore, in this work a new linear mixed-integer programming model for the exam timetabling problem is presented. The model uses a penalty-based goal programming approach to assure the construction of timetables which fulfill important requirements made by teachers, students and administrators. Based on this model, feasible solutions are derived by a standard solver and subsequently are further improved by a tabu-search procedure. The trade-off between different criteria is shown and some very promising results of the approach for a real-world data set are presented.

Keywords Examination timetabling · University timetabling · Linear programming · Mixed-integer programming

L. K. Bergmann
Institute for Operations Research and Information Systems
Hamburg University of Technology
Schwarzenbergstr. 95 D
21073 Hamburg
Germany
Tel.: +49-40-428784534
E-mail: katharina.bergmann@tuhh.de

K. Fischer
E-mail: kathrin.fischer@tuhh.de

S. Zurheide
E-mail: zurheide@tuhh.de

1 Introduction to exam timetabling at universities

The exam timetabling problem deals with the assignment of exams to rooms and time slots such that there is never more than one exam per student at a time. At universities, this combinatorial problem arises at the end of each term or semester. Usually there is a predefined time-span within each term, in which all exams have to be scheduled. For the assignment of exams to dates this overall examination time-span is split into several time slots (or exam periods) of equal length. The decision when and where to schedule an exam is quite challenging for the planner, and therefore tools which support the timetabling process are valuable.

One might assume that the exam timetable can be derived easily from the course timetable, as teachers might use the last session of their course for the respective examination. However, it is usually required that students do not have to take more than one exam per day, but as they can do more than one class per day, the course timetable cannot be used for the examinations. Moreover, there are many additional conditions to be fulfilled by an exam timetable, as will be explained below.

The course timetable is mainly determined by the curriculum of the respective degree program and is based on the assumption that every student participates in the courses scheduled in a term, and that he/she also passes the corresponding exams in the same term. In reality, students often take classes and exams in a different order. For example, due to lectures without compulsory attendance, illness during the examination time-span or failing of exams, the participation in courses and in the related exams is often rather independent from each other and hence the order in which exams are passed often does not follow the curriculum anymore. In addition, at German universities exams are usually offered in every term, even if the corresponding course is only taught every other term. Hence, about half of the exams that have to be scheduled are so-called resit exams which belong to courses that have not been taught in the current term. Moreover, it is usually required that the seating during an exam is more spacious than during a lecture (and thus a bigger room is required for the exam). Last but not least, there are usually many different electives between which students can choose, enhancing the danger of exam overlaps. All this makes the task of exam timetabling a very complex combinatorial problem.

A new approach for modeling the timetabling problem which is based on students' enrollments is presented in this work. Apart from the very basic requirements, e. g. that every student can only attend one exam per period, the linear mixed-integer model which is developed below contains several features in order to meet realistic demands of teachers and students, like the consideration of the work-load associated with an exam, the actual time-span between two exams or the distinction between exams from courses from the current term and resit exams. Additionally the changing availability of rooms, the need of splitting large exams over several rooms, the booking of external rooms for very specific exams and related to this also the preassignment of exams to dates or rooms is enabled. In order to implement all these requirements the model is based on a penalty based goal programming approach where deviations from soft requirements are penalized and hard requirements are integrated as constraints.

2 Literature review on examination timetabling

Several surveys on the topic state that the primary objective of exam timetable planning is to set up a conflict free schedule for every student (Carter (1986), Carter and Laporte (1996), Qu et al (2009)), i. e. a schedule in which no student has to take more than one exam at a time.

Resulting from this requirement, the most fundamental case of the exam timetabling problem is basically a graph coloring problem where each exam is represented by a node. If at least one student is enrolled for two exams, the corresponding nodes are connected by an undirected arc. The weights on the arcs equal the number of students enrolled in the two exams. The objective is then to find the minimal node coloring, where adjacent nodes do not have the same color, i. e. a schedule where no student is required to take more than one exam at a time. The colors can be identified with the available time slots. As the NP-hard coloring problem can be mapped polynomially onto the examination timetabling problem, the latter is also NP-hard (Garey and Johnson, 1979, pp. 13-14).

In the following a distinction between first and second order conflicts has to be made. A first order conflict arises if two exams are scheduled in the same period and there is at least one student enrolled in both exams. These conflicts have to be avoided by all means. Second order conflicts arise from exams that are "only" scheduled too close to each other, but not in the same period, and if there is at least one student enrolled for both exams.

In addition to the main requirement of being first order conflict-free, some additional hard constraints should be fulfilled by any timetable (Qu et al, 2009):

- Every exam has to be scheduled exactly once and
- Available capacities (rooms, invigilators, time) must not be exceeded.

If it is not possible to find a solution that satisfies the capacity requirements or the requirement of being conflict free, these constraints can be relaxed by adding dummy capacities or by adjusting the objective such that the number of first order conflicts is minimized (Carter and Laporte, 1996). However, this can lead to timetables that cannot be implemented in reality.

Depending on the individual demands of the university, there can be secondary objectives and requirements that can be included in the model as soft constraints. Table 1 lists some of the requirements that can be found in the literature and are often included in model formulations for realistic examination timetabling problems (more potential requirements can be found in Qu et al

Table 1 Overview on selected realistic examination timetabling requirements and authors considering them

Minimize the number of second order exam conflicts (i. e. spread the exams as much as possible over the entire examination time-span)	Müller (2013), Eley (2007), Burke and Newall (2004), Di Gaspero and Schaerf (2001), Carter et al (1994), Arani et al (1988), Laporte and Desroches (1984)
Preassign an exam to a specific time slot (or exclude this time slot)	Müller (2013), Carter et al (1994), Laporte and Desroches (1984)
Preassign an exam to a specific room or room type (or exclude this room/room type)	Müller (2013), Laporte and Desroches (1984)
Enable the consideration of preferred or required room types for some exams, e. g. rooms with large tables or computers	Carter et al (1994), Laporte and Desroches (1984)
Some rooms may not be available during the entire examination time-span	Müller (2013), Di Gaspero and Schaerf (2001), Laporte and Desroches (1984)
If big rooms are scarce it is possible to assign more than one room to an exam	Eley (2007)
Additionally the maximum number of rooms that an exam may be split into is limited	Müller (2013), Laporte and Desroches (1984)
Schedule more than one exam in a room (in the same time slot) if the number of available rooms is scarce	Eley (2007), Di Gaspero and Schaerf (2001), Laporte and Desroches (1984)
Penalize the scheduling of exams in certain time slots, e. g. avoid exams in the last day of the examination time-span	Müller (2013)
Enable the consideration of preferred time slots for some exams	Di Gaspero and Schaerf (2001), Carter et al (1994)

(2009)). Next to each aspect a few authors who consider these requirements in their articles are given.

The approaches by Müller (2013), Carter et al (1994) and Laporte and Desroches (1984) aim to set up realistic examination timetabling systems and consider many of the above named requirements, though none of them gives a complete mathematical formulation taking all aspects relevant for this work into account. Müller (2013) presents several benchmark data sets and an algorithm that resolves second order conflicts in several phases. Laporte and Desroches (1984) introduce an automatic timetabling procedure which includes the respective requirements from Table 1. Carter et al (1994) propose a scheduling system that is based on the article by Laporte and Desroches (1984) and implemented at the University of Toronto and at Carleton University. Carter et al (1996) also base their work on the article by Laporte and Desroches (1984) and carry out several experiments with algorithms that determine the optimal length of the examination time-span. These tests are conducted on unconstrained problems to determine the best strategy, which is then used to solve several real-life constrained problems. Carter et al (1996) do not explicitly state which requirements were considered for the constrained problems, therefore their work has not been included in Table 1.

The focus of the other publications listed above is primarily on the solution approach, and not on the formulation of a mathematical model. Eley (2007) uses different ant algorithms to solve the examination timetabling problem, Burke and Newall (2004) approach the problem through adaption of heuristic orderings, Di Gaspero and Schaerf (2001) present several variants of tabu-search based solution algorithms and Arani et al (1988) (and Arani and Lotfi (1989)) propose a three phase approach using a Lagrangian relaxation.

A common approach, which is also used by some of the authors mentioned above, to mathematically formulate the objectives and constraints listed in Table 1 is in the form of the quadratic assignment problem where the variables represent the assignment of exams to periods and sometimes also to rooms (Eley (2007), Burke and Newall (2004), Di Gaspero and Schaerf (2001), Arani and Lotfi (1989), Laporte and Desroches (1984)). For exams i and j being scheduled in period t and $t+a$, the corresponding constraints of the type "at most one exam per student within a certain time-span" can be formulated as quadratic constraints by the expression $c_{ij}x_{it}x_{j(t+a)} = 0$, with c_{ij} being the number of students that are enrolled for both exams i and j and x_{it} ($x_{j(t-a)}$) being equal to one if exam i (j) is scheduled in period t ($t+a$) and zero otherwise. Independent from the programming approach, the use of the conflict matrix c_{ij} can often be found in the literature, e. g. in Eley (2007), Burke and Newall (2004), Carter et al (1996) and Laporte and Desroches (1984).

For the topic of course timetabling, linear models are presented in the literature (e. g. Schimmelpfeng and Helber (2007), Van den Broek et al (2007), Dimopoulou and Miliotis (2001)). Schimmelpfeng and Helber (2007) model a course timetabling problem as a linear assignment problem. Using elements from goal programming, their model penalizes conflicting dates for courses and violations of room and teaching capacities. Instead of directly minimizing the number of students affected by a bad schedule, the model balances the students' work-load by minimizing the sum of penalties that apply in case of a bad schedule. Dimopoulou and Miliotis (2001) use a linear model to solve the course timetabling problem at a Greek university. Based on the solution of this model, an infeasible starting solution for the examination timetabling problem is generated, then modified into a feasible one by an algorithm that resolves the first order conflicts, and afterwards improved by rescheduling of exams.

Various other publications on the topic of examination timetabling can be found. An extensive literature review on examination timetabling is presented by Qu et al (2009). However, most publications focus on solution methods and use quadratic models whereas in this work, a linear approach is presented, to enable the use of standard solution approaches.

3 A new linear mixed-integer model for exam timetabling

Examination procedures may vary, e. g. depending on the educational institution, on the number of students, degree programs (timing, courses, choices), teachers, available facilities, technical support of the planner, and many more aspects. However, the following requirements and assumptions should match a typical German university.

The linear mixed-integer model that is introduced below considers all of the previously listed requirements, with a few exceptions regarding the room allocation. Experience from different German universities shows, that usually only one exam per room and period is scheduled. This is mainly due to the fact that German universities tend to have only very few large rooms, but enough small rooms, and hence it is usually possible to assign different exams to different (smaller) rooms.

Also the distinction of room types, apart from their capacity, is not explicitly included, but the model enables the exclusion of specific rooms for some exams (and therefore also the preassignment of a room to an exam). The model also includes a few additional requirements which are typical for German universities and were not taken into account by any of the above mentioned authors: If second order conflicts cannot be avoided, it is important to consider whether an exam belongs to a course from the current term, or to the previous term. Exams for courses from the current term are considered to be more important, as universities want to encourage the students to follow the curriculum as closely as possible. Hence, second order conflicts are to be avoided especially if both exams belong to courses from the current term.

This is done by using a goal programming based approach, such that the model penalizes three things: the occurrence of second order conflicts, the splitting of an exam over several rooms and the scheduling of exams in undesirable periods. The approach takes into account the number of students involved in second order conflicts, and the exams' work-loads. A similar approach for course timetabling can be found in Schimmelpfeng and Helber (2007).

3.1 General conditions and assumptions

It is assumed that students enroll for the exams they want to take in an examination period before the examination timetable is created, and that there are no exams without enrollments. In addition to the given enrollments, the examination time-span is split into periods and the total number of available periods within the overall examination time-span is predetermined. However, the number of available periods is varied in the computational study presented below to examine the effect of different examination time-spans. The periods are of equal duration, which is given in hours. E. g. an exam day that starts at 8 a.m. and ends at 6 p.m. can be represented by a period length of 10 hours and a total number of periods that equals exactly the number of days in the examination time-span (i. e. one period per day). Alternatively it is possible to represent each day by two periods, each being 4 hours long, or by any other number of periods per day. It only has to be made sure that the periods are of equal length, and of course there will be no feasible solution, if the duration of any exam exceeds the period duration. Hence, the exams' durations also have to be given in hours. This also allows to schedule more than one exam in the same room and period, if the sum of the exams' durations does not exceed the duration of a period. I. e. these exams are scheduled *subsequently* in the same room and within the same period. The exact schedule and order of exams in a room can easily be set up manually.

For the students it is very important that the work-loads associated with the exams are taken into account, especially when second order conflicts cannot be avoided. Therefore, in addition to the enrollments and duration of each exam, the ECTS points (European Credit Transfer and Accumulation System) for every exam have to be considered by the model.

In realistic timetabling situations, the room availabilities may not be the same for every period. E. g. conferences or other events might take place during part of the examination time-span, and hence certain rooms are not available at certain times. Moreover, it is necessary to enable the model to predetermine the period for an exam or exclude some periods for certain exams. The same has to be possible for the room allocation: e. g. for a very large examination, an external room might be booked, but other smaller exams should not take place in that room. To allow a reasonable assignment of rooms and to limit the number of invigilators needed, the maximum number of rooms into which an exam may be split has to be limited.

Furthermore the model allows to specify a "room allocation enrollment limit". If this is done, only exams that exceed the limit have to be scheduled with room. Smaller exams only get a period assigned. As pointed out above, at many German universities, small rooms (e. g. for seminars or group work) are available in abundance, such that the manual assignment of appropriate rooms to small exams is easily possible. This can lead to a considerable reduction of model size.

Finally it should be possible to avoid scheduling of exams in certain periods if this is feasible. E. g. students as well as teachers usually prefer not to have any exams in the very first or last periods of the overall examination time-span which usually starts right after the end of the term. If also weekends can be used for examinations, their use should also be avoided whenever possible.

3.2 Sets and parameters used in the model

In the following, the sets, indices and parameters which are needed to include the stated requirements and assumptions in the model are introduced. I is the set of all exams, and the indices for exams are i and j. Exams may be scheduled in a room r out of all rooms R. The total number of available exam periods is given by P with p being the index for periods such that $p \in \{1..P\}$.

To balance the students' work-load, they should not be required to take more than one exam within a certain number of periods. The length (in periods) of this time-span is defined by A, indexed by a, $a \in \{1..A\}$. If an exam is scheduled in period p, a penalty is applied if another exam is scheduled in a period $p+1$ to $p+A$, if there is at least one student enrolled in both exams.

To enable the distinction between exams from the current term and resit exams from the previous term, several sets of ordered tuples are introduced. Each tuple contains two exams that have a conflict potential, i. e. there is at least one student enrolled in two exams, i and j. These tuples of conflicting exams are generated in advance, based on the students' enrollment information. In the following sections and chapters, conflicting exams will be denoted by i' and i''.

The set I^T contains all exam pairs that have a conflict potential, i. e. all exam pairs where at least one student enrolled for both exams. I^{TC} contains only tuples of exams that both belong to courses from the current semester, while I^{TP} contains only tuples where at least one exam belongs to a course from the previous semester, such that $I^T = I^{TC} \cup I^{TP}$. All sets of tuples are indexed by (i', i'').

The parameter E_i gives the number of students that are enrolled for an exam $i \in I$ while the number of students that are enrolled for two exams i' and i'' (thus with conflict potential) is given by the parameter $C_{(i',i'')}$.

To reduce the problem size, the original set of all exams I is also complemented by two subsets: I^Z and I^W. I^Z is the set of all "big" exams with E^Z or more enrollments. Only exams in this set have to be scheduled with a room. As E^Z is a parameter of the model, it is of course possible to set its value to 1; then all exams will be scheduled with a room. Analogously I^W is the set of all exams with special time requirements. If no time requirement exists, the exam can be scheduled in any period. But if e. g. a teacher is absent during specific periods, his or her exam should not be scheduled in these periods (see also O_{ip} below).

The individual work-load or severity for each exam is represented by the parameter S_i. It takes a value between 1 and 5, $S_i \in \{1..5\}$. E. g. if i is a very easy exam, with a low work-load (and thus few ECTS points), S_i equals 1, but if i is a very difficult exam, with high work-load (and many ECTS-points), S_i equals 5. The difficulty is combined with $C_{(i',i'')}$ to determine the parameter $B_{(i',i'')}$, which gives the "badness" that occurs if exams i' and i'' are scheduled too close to one another. It is obtained by multiplying $C_{(i',i'')}$ with the exams' severities. Based on discussions and a university internal survey with students it is assumed that they prefer taking a difficult exam first and then an easier one, instead of the other way around. Hence, the "badness" value of a tuple is doubled if i'' has a higher work-load than i', i. e. $S_{i'} > S_{i''} \Rightarrow B_{(i',i'')} = 2C_{(i',i'')}S_{i'}S_{i''}$. Therefore, the elements of B are not symmetric (unlike $C_{(i',i'')}$).

The duration of the individual periods is given in hours and denoted by H. If H is set to a relatively large number (e. g. 8 hours), a high occupancy of each room can only be achieved by allowing the scheduling of more than one exam in a room. Therefore, the duration of each exam needs to be considered, to make sure that several exams can take place consecutively in the same room. This individual duration D_i gives the hours needed for each exam i, including time for preparation and follow up.

K_{rp} is a $(m \times P)$ matrix, with m being the total number of available rooms, and P the total number of available periods. The elements of K_{rp} are positive integer or zero and give the number of seats that are available in room r during period p. If a room is not available in a specific period, the value for the corresponding K_{rp} is set to zero. The time requirements for every exam are specified by the binary parameter O_{ip}. It equals 1 if exam i may be scheduled in period p and 0 otherwise. If a special time requirement for an exam i exists ($\sum_{p\in\{1..P\}} O_{ip} < P$), the exam will be added to the set of exams with time requirements, I^W, as mentioned above. Similar to this Q_{ir} is a binary parameter which specifies the room requirements for every exam: It equals 1 if exam i may be scheduled in room r and 0 otherwise. The parameter F limits the number of rooms into which an exam may be split.

There are four penalty factors to enable a weighting of the different terms of the objective function. (Note that the superscripted number is an index to distinguish between the different parameters and not a mathematical exponent.) N_a^1 gives the penalty that applies if conflicting exams are scheduled within $A+1$ periods, with $a \in \{1..A\}$. It enables the model to penalize second order conflicts with respect to the actual distance of time, a, of the corresponding exams (similar to Eley (2007) or Carter et al (1996)). N^2 determines the impact of second order conflicts of two exams from current courses compared to conflicts with an exam from a previous term course. N^3 defines the level of the penalty for splitting an exam into several rooms. Finally, N_p^4 penalizes the scheduling of exams in undesirable periods, e. g. all periods representing a Saturday. These factors can of course also be used to favor certain schedules (instead of penalizing them) if the values are set appropriately (i. e. if the penalty values are set between 0 and 1).

3.3 Decision and deviational variables

The timetabling model presented below comprises two types of decision variables. There are two classical decision variables, y_{irp} and x_{ip}, and two so called deviational variables, $u_{(i',i'')a}$ and v_{ip}.

The variable y_{irp} gives the information in which room and period an exam is scheduled. Hence, it is equal to 1 if exam i is scheduled in room r in period p, and otherwise it is zero. The second decision variable, x_{ip}, only gives the information in which period an exam i is scheduled. x_{ip} is equal to 1 if exam i is scheduled in period p and equal to 0 otherwise. Due to this separate allocation of periods and rooms it is possible to ensure that every exam is scheduled exactly once, but to allow several rooms to be assigned to one exam.

The first deviational variable $u_{(i',i'')a}$ is binary and indicates whether a student has to sit two exams i' and i'' within $A+1$ periods or not. If exam i' is scheduled in period p and exam i'' is scheduled in period $p+a$ then $u_{(i',i'')a}$

takes the value 1. If there is no student enrolled in both exams or there are more than A periods in between the exams, $u_{(i',i'')a}$ takes the value 0. The index a contains the information how many periods there are in between the periods in which the two exams are scheduled.

The number of additional rooms that are assigned to exam i scheduled in period p is given by v_{ip}. E. g. if exam i is scheduled in period p and assigned to three rooms, then $v_{ip} = 2$. On the other hand, if $v_{ip} = 0$, exam i is either not scheduled in period p or it is assigned to just one room. These two decision variables are thus called positive deviational variables (Jones and Tamiz, 2010, pp. 4-5, 20-22).

3.4 Model formulation

To provide a better overview of the notation used, the following lists contain all sets, indices, parameters, decision and deviational variables.

Sets:

I	Set of all exams
I^W	Set of exams with time specification
I^Z	Set of exams with E^Z or more enrollments
I^T	Set of ordered tuples of all exam pairs with conflict potential, $I^T = I^{TC} \cup I^{TP}$
I^{TC}	Set of ordered tuples of exams with conflict potential that belong to courses from the current term
I^{TP}	Set of ordered tuples of exams with conflict potential, where at least one exam is a resit exam
R	Set of rooms

Indices:

1–4	Naming indices for penalty factors
a	Index for "time lags" between two exams, $a \in \{1..A\}$
i, j	Indices for exams, $i, j \in I, I^Z$ or I^W
(i', i'')	Tuple of indices for exam pairs with conflict potential, $(i', i'') \in I^T, I^{TC}$ or I^{TP}
p	Index for periods, $p \in \{1..P\}$
r	Index for rooms, $r \in R$

Parameters:

A	Number of consecutive periods in which no student should have to write more than one exam
$B_{(i',i'')}$	Badness for exams of tuple (i', i'') being scheduled too close
D_i	Duration of exam i
E_i	Number of students enrolled for exam i

E^Z	Limit of enrollments, such that all exams with E^Z or more enrollments have to be scheduled with room
F	Limit on rooms, an exam may be split into
H	Length of each period in hours
K_{rp}	Seating capacity of room r in period p
N_a^1	Penalty factor that applies if exams of a tuple are scheduled within $a+1$ periods
N^2	Additional penalty factor that only applies for tuples $(i', i'') \in I^{TC}$
N^3	Penalty factor that applies if an exam is split into several rooms
N_p^4	Penalty factor that depends on the period in which an exam is scheduled
O_{ip}	Time specification that indicates if institutes/teachers prefer exam i to be scheduled in period p or not
P	Total number of available periods, being H hours long each
Q_{ir}	Room specification that indicates if exam i may be scheduled in room r or not
S_i	Work-load (severity) of exam i

Decision variables:

x_{ip}	Equal to 1 if exam i is scheduled in period p, and 0 otherwise
y_{irp}	Equal to 1 if exam i is scheduled in room r in period p, and 0 otherwise

Deviational variables:

$u_{(i',i'')a}$	Equal to 1 if exams of tuple (i', i'') are scheduled within $a+1$ periods, and 0 otherwise
v_{ip}	Equal to the number of additional rooms occupied by exam i in period p, and 0 otherwise

The following model formally states a realistic exam timetabling problem, taking all above mentioned aspects into account, and using the notation presented.

Objective function:

$$\min \sum_{a \in \{1..A\}} N_a^1 \left(\sum_{(i',i'') \in I^{TC}} N^2 B_{(i',i'')} u_{(i',i'')a} + \sum_{(i',i'') \in I^{TP}} B_{(i',i'')} u_{(i',i'')a} \right) + \sum_{i \in I} \sum_{p \in \{1..P\}} N^3 v_{ip} + \sum_{i \in I} \sum_{p \in \{1..P\}} N_p^4 S_i x_{ip} \tag{1}$$

Subject to:

$$\sum_{p \in \{1..P\}} x_{ip} = 1 \qquad \forall i \in I \tag{2}$$

$$x_{ip} \le O_{ip} \qquad \forall i \in I^W, \forall p \in \{1..P\} \tag{3}$$

$$x_{i'p} + x_{i''p} \le 1 \qquad \forall (i', i'') \in I^T, \forall p \in \{1..P\} \tag{4}$$

$$x_{i'p} + x_{i''(p+a)} - u_{(i',i'')a} \le 1 \qquad \forall (i', i'') \in I^T, \forall p \in \{1..P\}, \forall a \in \{1..A\} \tag{5}$$

$$\sum_{p \in \{1..P\}} y_{irp} \le Q_{ir} \qquad \forall i \in I^Z, \forall r \in R \tag{6}$$

$$\sum_{r \in R} y_{irp} - v_{ip} = x_{ip} \qquad \forall i \in I^Z, \forall p \in \{1..P\} \tag{7}$$

$$\sum_{r \in R} y_{irp} \le F \qquad \forall i \in I^Z, \forall p \in \{1..P\} \tag{8}$$

$$\sum_{r \in R} K_{rp} y_{irp} \ge E_i x_{ip} \qquad \forall i \in I^Z, \forall p \in \{1..P\} \tag{9}$$

$$\sum_{i \in I^Z} D_i y_{irp} \le H \qquad \forall r \in R, \forall p \in \{1..P\} \tag{10}$$

$$y_{irp} \le x_{ip} \qquad \forall i \in I^Z, \forall r \in R, \forall p \in \{1..P\} \tag{11}$$

$$x_{ip} = 0 \qquad \forall p \in \{P+1..P+A\}, \forall i \in I \tag{12}$$

$$x_{ip} \in \{0,1\} \qquad \forall i \in I, \forall p \in \{1..P\} \tag{13}$$

$$y_{irp} \in \{0,1\} \qquad \forall i \in I^Z, \forall r \in R, \forall p \in \{1..P\} \tag{14}$$

$$u_{(i',i'')a} \in \{0,1\} \qquad \forall (i', i'') \in I^T, \forall a \in \{1..A\} \tag{15}$$

$$v_{ip} \ge 0 \qquad \forall i \in I^Z, \forall p \in \{1..P\}. \tag{16}$$

Constraints (2) ensure that every exam is scheduled exactly once. Equations (3) take the time requirements of institutes and teachers into account. They also allow to predetermine the period p in which a certain exam i has to be scheduled, by setting O_{ip} to 1 for only this specific period p.

As represented in the constraints (4), no student can take two exams at the same time, hence, first order conflicts cannot occur in a feasible solution. Constraints (5) relate to exams being scheduled within $A + 1$ periods. As both equations only apply to exams that have a conflict potential, they only have to hold for the tuples of the set I^T. In (5) the first set of deviational variables, $u_{(i',i'')a}$, is used: Whenever there is at least one student enrolled in both exams i' and i'' and these exams are scheduled within $A + 1$ consecutive periods, $u_{(i',i'')a}$ takes the value 1.

The next six groups of constraints relate to exams that have to be scheduled with a room, as they have E^Z or more participants; these are all exams $i \in I^Z$. Constraints (6) determine the room specifications for each exam, similar to the time specification in (3). Only if Q_{ir} is equal to 1, the exam i may be scheduled in room r. This allows to predetermine the location(s) for exams, release rooms only for selected exams and to exclude specific exams from a certain room.

In some cases it might be necessary to split up the participants of one exam over several rooms. Equations (7) enable this, using the deviational variables v_{ip}, which equal the additional number of rooms needed for an exam. Hence their values should be minimized, as it is preferred not to split up exams. The maximum number of rooms in which an exam might take place is determined in constraints (8). Restrictions (9) make sure that the rooms which are assigned to an exam have enough seats for all enrolled students.

Constraints (10) allow several exams to be scheduled in the same room and period, as long as the sum of their durations does not exceed one period. Constraints (11) restrict the assignment of rooms to the period in which the corresponding exam is scheduled. Finally, constraints (12) to (16) declare the decision and deviational variables' domains.

The objective function (1) aims to find a schedule that keeps the work-load for each student balanced, but also meets the university's resources. This is done by minimizing the deviations from the following three targets: First, every student should take at most one exam (e. g. exam i') within $A + 1$ periods. For every additional exam within this time range (e. g. exam i''), the deviational variable $u_{(i',i'')a}$ takes the value 1. The index a indicates within how many periods these two exams are scheduled and is one of three factors that determine the influence of a second order conflict on the objective function value. If the values of N_a^1 are inverse proportional to a it is ensured that the closer the two corresponding exams are scheduled, the larger the contribution of a conflict to the objective function value will be.

The second influencing factor is the so-called badness $B_{(i',i'')}$, a combination of the number of affected students and the work-load of each exam, as was explained in subsection 3.2. The last influencing factor is the information whether the tuple (i', i'') contains an exam from the previous semester or not. $B_{(i',i'')}$ is multiplied by N^2 if the tuple (i', i'') contains only exams from courses from the current semester.

Second, every exam should be scheduled in only one room. If, however, the students enrolled for an exam do not fit into the biggest room available, it is possible to split the exam over several rooms. For an exam i, scheduled in period p, v_{ip} indicates how many additional rooms are required. The factor N^3 enables an adequate weighting of this target and penalizes the use of additional rooms.

The last target considered by the objective function (1) is the general choice of favorable periods or, more explicitly, the choice of inconvenient periods. Depending on the values of N_p^4, scheduling exams e. g. at the beginning and at the end of the overall exam period is penalized. Especially difficult exams should not be scheduled right in the first periods to grant the students enough time for preparation. For this reason, the penalty is multiplied by the work-load S_i of the corresponding exam.

For a better understanding of the objective function, a small example of an exam schedule is presented in the following. Table 2 shows a timetable with

Table 2 Example schedule

	Periods (p)				
	1	2	3	4	5
Room 1	i_D	i_E			i_F
Room 2	i_D		i_G	i_H	
Example Student	i_D	i_E			i_F
N_p^4	50	30	1	40	80

five periods, two rooms and five exams (named i_D to i_H). Furthermore, the enrollments of an example student and the values of the parameter N_p^4 can be found in this table.

The example student is enrolled for the exams i_D, i_E and i_F. It is assumed that 20 other students have the same exam conflicts. The degrees of difficulty for these three exams are as follows: $S_D = 3$, $S_E = 5$ and $S_F = 1$. The exams i_D and i_E are from the current semester, and the exam i_F belongs to a course from the previous semester. The values of the remaining parameters are $A = 3$, $N_1^1 = 100$, $N_2^1 = 10$, $N_3^1 = 1$, $N^2 = 2$ and $N^3 = 1000$.

The badness of the exam tuples needs to be determined, based on the formula presented in subsection 3.2, i. e. $B_{(i',i'')} = [2]\, C_{(i',i'')} S_{i'} S_{i''}$. Hence, $B_{(D,E)}$ is equal to $2 \cdot 20 \cdot 3 \cdot 5 = 600$, $B_{(D,F)}$ is equal to $20 \cdot 3 \cdot 1 = 60$ and $B_{(E,F)}$ is equal to $20 \cdot 5 \cdot 1 = 100$. Here, the first exam tuple has a high value for the parameter $B_{(D,E)}$, because the conflicting exams have a high work-load, and the exam with the lower work-load is scheduled first. The following calculation shows how the objective function value for this small example is computed:

$$\begin{array}{cccccccccc}
N_1^1 & & N^2 & & B_{(D,E)} & & u_{(D,E)1} & & & \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & & \\
(100 & \cdot & 2 & \cdot & 600 & \cdot & 1 & + & & \\
 & & & & & & & & & \\
N_3^1 & & B_{(E,F)} & & u_{(E,F)3} & & N^3 & & v_{D1} & \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\
1 & \cdot & 100 & \cdot & 1) & + & (1000 & \cdot & 1) & + \\
 & & & & & & & & & \\
N_1^4 & & S_D & & x_{D1} & & & & & \\
\downarrow & & \downarrow & & \downarrow & & & & & \\
(50 & \cdot & 3 & \cdot & 1 & + & & & & \\
 & & & & & & & & & \\
N_2^4 & & S_E & & x_{E2} & & & & & \\
\downarrow & & \downarrow & & \downarrow & & & & & \\
30 & \cdot & 5 & \cdot & 1 & + & \ldots) & & &
\end{array}$$

In the first part of the calculation, the facts that the exams i_D and i_E are planned without a period inbetween ($a = 1$) and that there is only a gap of two periods between the exams i_E and i_F ($a = 3$) are penalized. The time gap between the exams i_D and i_F is big enough so that it is not penalized. The first exam conflict (i_D, i_E) accounts for 120 000 units of penalty costs, because the exams are scheduled in consecutive periods, both exams are from the current semester and the badness value for this tuple is very high. The conflict between the exams i_E and i_F results only in penalty costs of 100, because the exam i_F is for a course from the previous semester, there is a time gap of two days between the exams and the badness value is low.

The second part of the calculation represents the penalties for splitting the exam i_D over two rooms. Each additional room is penalized with a penalty value of 1000, hence the contribution to the objective function value is 1000 in this case. The last part of the objective function adds penalty costs for each exam depending on the scheduled period. For example, penalty costs of 150 are added for scheduling the exam i_D in the first period. The other exams are penalized accordingly.

4 Experimental results

The easiest way to derive a feasible solution for the model presented above is using a standard solver, e. g. Gurobi Solver or IBM ILOG CPLEX Optimization Studio. For the test case given below, Gurobi constructed feasible solutions for different numbers of available periods. Due to the size of the test case, which has been chosen to meet the size of realistic instances, and the complexity of the problem these timetables were not very good (in terms of the objective function values), even after two days of run-time. Therefore the solutions found by the solver were further improved by a tabu-search based heuristic.

As student numbers are relevant for the respective contribution to the objective function, it shows that second order conflicts have a major impact on the objective function value. Therefore, the heuristic focuses on resolving these conflicts based on a feasible start solution which is constructed by Gurobi and handed over in the beginning.

To improve the provided timetable there are four possible moves that can be performed by the heuristic. The first move tries to move a single exam to a new period, the second tries to exchange the periods of two exams and the third move tries to exchange the periods of three exams. These moves are only performed if they improve the timetable. Following the idea of tabu-search the fourth move may also deteriorate the schedule, by exchanging all the exams of two periods. By this the heuristic can escape from local optima.

As a test case, the original examination data from the Hamburg University of Technology (TUHH) of the winter term 2012/2013 is used. In that term there were 243 exams to be scheduled, and a total of 23 317 enrollments that yield 7132 tuples of exams with conflict potential. The durations of the examinations and the capacities of the rooms are based on the original data of the university. The values for the work-load of each exam were set according to the ECTS points of the corresponding courses. At this university the exams are scheduled in the lecture-free time at the end of each term. The relevant time-span amounted to 39 days (= number of available periods P)

in the corresponding term. It would be advantageous if this time-span could be shortened, to provide students with more time for internships, laboratory work or vacation. It turned out that the minimum number of days for which a feasible solution could be generated is 24. Hence, the number of periods P was varied from 24 to 39 to study the effect of the length of the examination time-span. For all cases the number of consecutive periods, in which no student should have to write more than one exam, was set to $A = 3$.

Values for the penalty parameters were defined as follows: The factors that penalize conflicts according to the actual number of periods in between the two corresponding exams are $N^1_a = 100$ (for $a = 1$), 10 (for $a = 2$) and 1 (for $a = 3$). The penalty that applies for conflicts of exams from the current term is $N^2 = 2$. For each additional room that an exam is split to the penalty is $N^3 = 1000$ and finally, to guide the spreading of exams the last penalty-vector is $N^4_p = \{60, 50, 40, 20, 5, 5, 5, 1, \dots 1, 10, 10, 20, 20, 30, 60, 80, 100, 140, 180, 220, 260\}$ where all values are set to 1 for periods greater 7 and smaller $(P - 11)$.

Feasible solutions for the different time-spans were generated by the Gurobi Solver (version 5.5) on a computer with two 2.27 GHz Intel Xeon quad core processors and 24 GB RAM. The model for 39 periods consists of 1 140 612 rows, 237 496 columns and 3 888 846 non-zeros in the coefficient matrix. As the resulting solutions after two days of optimization time still showed a huge integrality gap, they were afterwards improved by the tabu-search based heuristic.

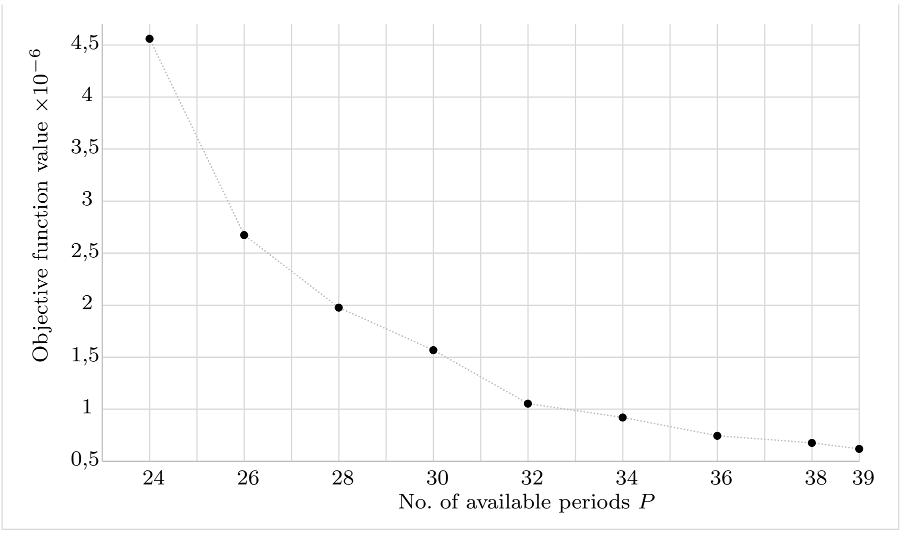


Fig. 1 Improved solutions for different numbers of available periods P

Figure 1 shows the resulting objective function values for different numbers of available periods P. The downward trend meets the expectation that the incurred penalties decrease with an increase of the total number of available

Table 3 Number of second-order conflicts for selected improved timetables

Number of available periods P	24 periods		28 periods		39 periods	
Total number of conflicts	762		640		379	
- affecting 10 or less students	602	(79%)	512	(80%)	327	(86%)
- affecting only one student	215	(28%)	196	(31%)	126	(33%)

periods, as exams with conflict potential can more easily be scheduled with a greater distance of time. In fact a closer look at the solutions shows that (e. g. due to the chosen values of the penalty factors) the main contribution to the objective function value results from second order conflicts: For the improved 24-periods-solution there are 762 second order exam conflicts (out of the 7132 exam pairs with conflict potential), but only seven exams (out of 243) had to be split into several rooms (actually exactly two rooms per exam). For the 28-periods-solution the improved timetable shows 640 second order conflicts and 11 exams were split into two rooms. For the 39-periods-solution the number of second order conflicts reduces to 379, and also 11 exams were split into two rooms.

For these three solutions, the number of conflicts can also be found in Table 3, which shows that the occurring second order conflicts mostly affect very few students. Moreover, a closer look at the solutions revealed that only very few of these conflicts involve two exams from courses from the current term.

The histogram in Figure 2 shows for the initial and improved 39-periods solution the number of conflicts that result from exams being scheduled too closely to each other. The horizontal axis gives the range in which the penalty costs induced by the conflicts lie, while the vertical axis gives the number of secondary conflicts per range. The total number of conflicts is reduced from initially 532 to 379. The figure shows that especially the number of conflicts with large penalties, i. e. the number of conflicts that either affect many students or concern difficult exams (or both), is significantly reduced by the heuristic procedure.

The original examination timetable, as it was (manually) generated and executed at the TUHH in the winter term 2012/2013 is not displayed in Figure 1. With respect to the model formulation in subsection 3.4, the original timetable was infeasible: There were 55 first order conflicts, i. e. there were 55 exam pairs scheduled on the same day, although there were students enrolled in both corresponding exams. Additionally, there were a few minor room size mismatches, i. e. some exams were scheduled in too small rooms. In practice this is not a problem, as there are usually a few students who do not attend every exam they are enrolled for, such that a slightly smaller room suffices. Ignoring all infeasibilities, the executed solution results in an objective function value of 736 882, but, due to the modifications that were necessary to resolve the infeasibilities, it is not advisable to compare this timetable to those constructed by the approach suggested in this work.

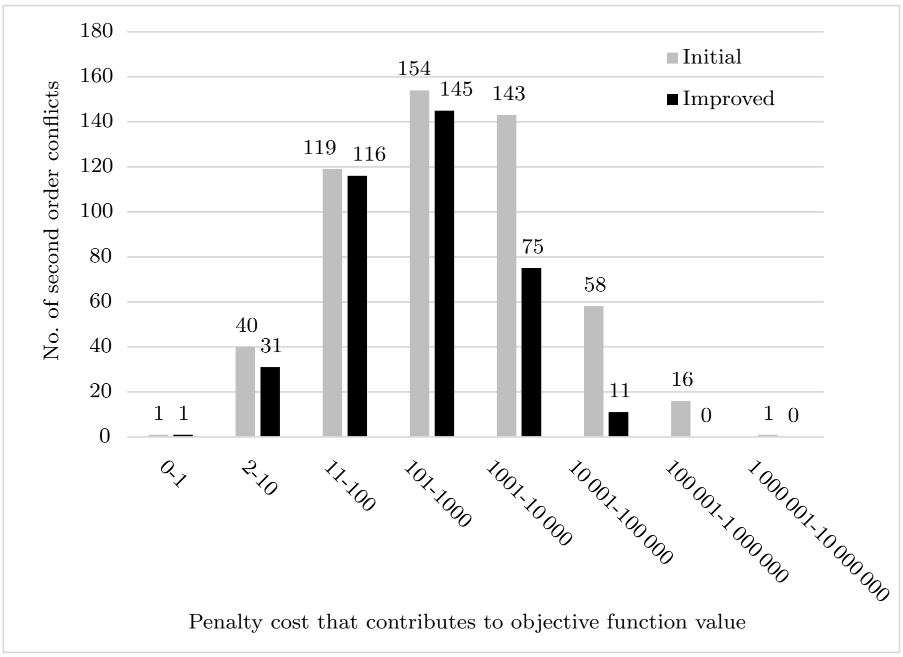


Fig. 2 Number of conflicts for initial and improved 39-periods-timetables

The results show that a compromise has to be found between the length of the examination period and the number of second order conflicts, as there is a trade-off between the two. Of course, which compromise is best may differ and depends largely on the preferences of the decision makers at the respective university. E. g. from Figure 1 it can be concluded, that in the specific case under study the shortening of the examination time-span by one week would lead to a still acceptable level of conflicts, which is only slightly higher than the one for the 39-period solution; so this might be a good compromise for this situation.

5 Summary and outlook

In this work, a linear mixed-integer model for the examination timetabling problem is presented. This model includes not only the fundamental constraints that have to be fulfilled, but also possibilities to set certain exams in specific periods or rooms. Furthermore, conflicts of exams are weighted based on the work-load, the time-span between exams and the term in which the respective course is taught. Based on students' enrollments and the resulting conflicts the model is aimed at finding the feasible solution which minimizes these conflicts, avoids the splitting of exams over rooms and the assignment of exams to inconvenient periods.

Several feasible timetables that were constructed by the Gurobi solver and improved by a tabu-search based heuristic demonstrate that the approach presented in this work can contribute substantially to the overall contentment of students and university teachers, as it is able to reduce the number of conflicts considerably compared to solutions which are constructed manually. The generated timetables could be used in practice at the TUHH, if the exam scheduling could be based on the students' enrollments. However, due to a different organizational approach this is currently not the case.

Concerning the constraints and targets of the model, there are still many issues that might be included in the future, e. g. the introduction of room types to satisfy special requirements of some exams (e. g. if computer workstations or tables for drawing are needed) or the assignment of exact start times for exams (not only the period) and different treatment of exams from compulsory and elective courses. Additionally, further analyses on the penalty parameters of the model or with respect to solution strategies might be carried out. An exact solution procedure like column generation might be able to solve realistic instances to optimality. These aspects are left for future research.

References

Arani T, Lotfi V (1989) A Three Phased Approach To Final Exam Scheduling. In: IIE Transactions, vol 21 (1), Taylor & Francis, pp 86–96

Arani T, Karwan M, Lotfi V (1988) A Lagrangian relaxation approach to solve the second phase of the exam scheduling problem. In: European Journal of Operational Research, vol 34, Elsevier Science B.V., pp 372–383

Van den Broek J, Hurkens C, Woeginger G (2007) Timetabling Problems at the TU Eindhoven. In: Burke EK, Rudová H (eds) PATAT 2006. LNCS, vol 3867, Springer, pp 210–227

Burke EK, Newall JP (2004) Solving examination timetabling problems through adaption of heuristic orderings. In: Annals of Operational Research, vol 129, Kluwer Academic Publishers, pp 107–134

Carter MW (1986) A Survey of Practical Applications of Examination Timetabling Algorithms. In: OR Practice, vol 34, Elsevier Science B.V., pp 193–202

Carter MW, Laporte G (1996) Recent Developments in Practical Examination Timetabling. In: Burke EK, Ross EK (eds) PATAT 1995. LNCS, vol 1153, Springer, pp 1–21

Carter MW, Laporte G, Chinneck JW (1994) A General Examination Scheduling System. In: Interfaces 24, pp 109–120

Carter MW, Laporte G, Lee SY (1996) Examination Timetabling: Algorithmic Strategies and Applications. In: Journal of the Operations Research Society 47, pp 373–383

Di Gaspero L, Schaerf A (2001) Tabu Search Techniques for Examination Timetabling. In: Burke EK, Erben W (eds) PATAT 2000. LNCS, vol 2079, Springer, pp 104–117

Dimopoulou M, Miliotis P (2001) Implementation of a university course and timetabling system. In: European Journal of Operational Research, vol 130, Elsevier Science B.V., pp 202–213

Eley M (2007) Ant Algorithms for the Examination Timetabling Problem. In: Burke EK, Rudová H (eds) PATAT 2006. LNCS, vol 3867, Springer, pp 364–382

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

Jones D, Tamiz M (2010) Practical Goal Programming, 1st edn. Springer

Laporte G, Desroches S (1984) Examination Timetabling by Computer. In: Computers and Operations Research, vol 11, No. 4, pp 351–360

Müller T (2013) Real-life examination timetabling. In: Proceedings of the 6th Multidisciplinary International Scheduling Conference

Qu R, Burke EK, McCollum B, Merlot LTG, Lee S (2009) A survey of search methodologies and automated system development for examination timetabling. In: Journal of Scheduling, vol 12, Springer, Heidelberg, pp 55–89

Schimmelpfeng K, Helber S (2007) Application of a real-world university-course timetabling model solved by integer programming. In: OR Spectrum, vol 29, Springer, pp 783–803

A Multi-Stage IP-Based Heuristic for Class Timetabling and Trainer Rostering

Oliver Czibula · Hanyu Gu · Aaron Russell · Yakov Zinder

Abstract We consider an academic timetabling and rostering problem involving periodic retraining of large numbers of employees at an Australian electricity distributor. This problem is different from traditional high-school and university timetabling problems studied in the literature in several aspects. We propose a three-stage heuristic consisting of timetable generation, timetable improvement, and trainer rostering. Large-scale integer linear programming (ILP) models for both the timetabling and the rostering components are proposed, and several unique operational constraints are discussed. We show that this solution approach is more flexible regarding constraints and objectives, and is able to generate solutions of superior quality to the existing software system in use at the organisation.

Keywords Timetabling · Rostering · Integer Programming · Heuristic

O. Czibula
School of Mathematical Sciences
University of Technology, Sydney
Tel.: +61-2-95142281
Fax: +61-2-95142260
E-mail: oliver.czibula@student.uts.edu.au

H. Gu
School of Mathematical Sciences
University of Technology, Sydney
Tel.: +61-2-95142281
Fax: +61-2-95142260
E-mail: hanyu.gu@uts.edu.au

A. Russell
48-50 Holker Street
Silverwater, NSW 2128
Tel.: +61-2-87451569
Fax: +61-2-96486859
E-mail: ajrussell@ausgrid.com.au

Y. Zinder
School of Mathematical Sciences
University of Technology, Sydney
Tel.: +61-2-95142279
Fax: +61-2-95142260
E-mail: yakov.zinder@uts.edu.au

1 Introduction

Ausgrid, Australia's largest electricity distributor, is responsible for building, repairing, and maintaining all the electrical substations, voltage transformers, and overhead and underground cables that supply electricity to homes, businesses, and industries within their operational area of 22,275km^2. The voltages on Ausgrid's electricity network range from 230V to 132kV, and there is an extreme risk of electrocution if works are not performed carefully and with strict safeguards in place. In addition to electrocution, other hazards Ausgrid workers face include falling from heights, having objects dropped on them from high above, working in confined spaces, and working in the presence of hazardous materials such as toxic gas, asbestos, or other harmful substances. Having such a hazardous working environment and supplying such a vital utility to the population, it is among Ausgrid's highest priorities to deliver safety and technical training promptly and efficiently to all people, including Ausgrid employees, contractors, and third parties, working on or near the electricity network, as required by Australian industry law.

Most training delivered by Ausgrid has a limited validity period, after which it is considered lapsed and no longer valid. Most courses have a validity period of 12 months from the date of successful completion. Others can last 3 or 5 years, and a few have indefinite validity periods, and validity periods are subject to change as the industry legislation related to training is occasionally revised. If a worker does not successfully complete the required training again before it expires, they will not be permitted to work on or near the electricity network until they do successfully complete the training.

Ausgrid delivers many different training courses, and each course is composed of one or more modules. All students enrolled in a course must complete all modules together. Each module has a duration, and a maximum number of students that it can have (some modules are better suited to be taught in large groups, whereas others require more individual attention from the trainer, hence they should be run in smaller groups). The modules of a course can be run in any order, however they must be run back-to-back (Ausgrid does not permit gaps between the modules of a course). The only exceptions are lunch time, which is fixed at 12:00 to 12:30, and after-hours for courses that go for longer than a day. Courses may not start at arbitrary times. If a course has a total duration of half a day or less, it may start first thing in the morning, or right after lunch. Otherwise, if a course goes for longer than half a day, it may only start first thing in the morning. Each course can be run an arbitrary number of times, sometimes several times a day, and each individual run is known as a course instance.

Ausgrid's operational area can be divided into a number of disjoint and congruent geographical regions, where each region contains one or more training facilities, which we refer to as locations. Each location contains one or more rooms, which come in various sizes and may contain certain equipment necessary for certain modules. Some rooms have a built-in divider wall, which allows the room to be split into two, separate, smaller rooms. The modules of courses are run in rooms. Each room has a list of compatible modules, and a maximum number of students it can accommodate. Since both modules and rooms have a limitation on the number of students, a module-room pair has a maximum number of students given by the minimum of these two values. While different modules of a course can be assigned to different rooms, it is important that all the modules of a course are assigned to a single location.

Modules are taught by trainers, each of whom has a location to which they are assigned by default. Trainers may, however, travel to other locations with their company vehicle when required. All trainers work a standard work day from 07:30 to 16:00 with a half-hour lunch break from 12:00 to 12:30. In addition to their training requirements, trainers are also required to perform administrative tasks such as general paperwork and course development, therefore each trainer has a target training workload utilisation depending on which type of trainer they are. Trainers may be unavailable due to planned annual leave, or planned or unplanned sick leave.

Certain groups of modules may require shared, mobile resources. For example, there may be a number of different fire fighting modules, which belong to different courses and which all require a large piece of fire fighting equipment of which Ausgrid may only own a limited number. These pieces of fire fighting equipment can be relocated from place to place, however pack-up, transportation, and unpacking time must be considered. The total number of these fire fighting modules that can run at any given time in any given place is limited by the quantity of the required equipment present.

Because training is delivered not only to Ausgrid employees, but also contractors and third parties over whom Ausgrid has little influence, Ausgrid does not currently schedule individual participants into classes in advance, as is the case in most high school and university timetabling. Instead, Ausgrid schedules classes to run in times and places where people are expected to need training, and those people book themselves, or are booked by their supervisors, into a suitable class around their existing duties. Since the courses have a known validity period, Ausgrid is able to cross-reference the training records with the current staff details to get a fairly accurate breakdown of how many people will require certain types of training in particular locations at particular times. We call this the demand, which is characterised by the number of participants expected for a course in a given region and window of time. Ausgrid must schedule at least enough courses of each type in each region across the planning horizon to cater for the expected demand.

The robustness of the training plan is of paramount importance to Ausgrid. Since people are expected to book themselves into classes when needed, a good timetable should exhibit certain characteristics that maximise the likelihood that people will be able to find a class at a suitable time and place. For example, it is desirable for courses to be distributed uniformly throughout the planning horizon. Moreover, a timetable should exhibit robustness by minimising the impact of unforeseen events that may affect the running of courses. One undesirable element in a timetable, and source of uncertainty, is a room swap which happens when consecutive modules of the same course instance are assigned to different rooms. We consider it highly desirable if all the modules of a course can be run in a single room allowing the students and trainer to remain in one place, uninterrupted, for the whole duration of the course. Conversely, requiring the students and trainer to pack up and migrate to a nearby room is not only disruptive to the flow of the course, but it also detracts from the robustness of the timetable as the room may be unexpectedly unavailable, jeopardising the whole course if a substitute room cannot be found.

When rostering trainers it is desirable for trainers not to have to travel excessively, and it is desirable not to have to change trainers often throughout a course. Both of these factors play a role in the robustness of the overall solution. Ideally, a whole course instance should be taught by a single trainer where possible. By having more trainers allocated to a course than necessary, the robustness of the timetable is compromised as each new trainer allocated to a course has the possibility of being sick or otherwise unavailable, jeopardising the entire course instance if no substitute can be found. As with swapping rooms, trainer swaps take time and can delay the progress of a course if one trainer is late. Similarly, having trainers travel longer distances more often not only incurs the cost of travel (travel cost is not directly considered in this model), but also increases the likelihood that the trainer will fail to arrive at their class on time due to traffic, roadwork, accident, etc., detracting from the overall robustness of the timetable.

Rooms in certain locations can be rented out when not in use. If these rooms are not required by Ausgrid for an extended period of time, they can be advertised to be leased by third parties generating some revenue for Ausgrid. Currently, due to poor timetable optimisation, not many rooms are available for third-party rental, however if the quantity of revenue generated can be shown to be significant, Ausgrid may expand their room rental program.

Currently Ausgrid uses a software heuristic to generate their training plans on a month-by-month basis. This software is able to rapidly generate a timetable, however does not contain any optimisation functionality. Due to rising electricity prices and resulting government pressure, Ausgrid must minimise their operational costs where possible. Due to changing industry regulations related to safety and technical training, as well as long-term fluctuations of demand, Ausgrid needs a tool to manage and optimise their training plan which is capable of handling these changes. The current software tool is not sufficiently flexible to handle many of the changes that have happened in the recent past.

In this paper we propose a three-stage heuristic procedure consisting of an initial timetable generation stage, an iterative timetable improvement stage, and finally a trainer rostering stage. Integer linear programming (ILP) models are developed for each stage, which can deal with all the practical requirements flexibly. Different algorithms are designed to achieve the balance of solution quality and computing time.

The remainder of the paper is organised as follows: Section 2 gives an outline of the current state of research in the area of academic timetabling. Section 3 describes the three-stage heuristic in detail. Section 4 describes the class timetabling

ILP model, and section 5 describes the trainer rostering ILP model. Section 6 discusses some important details about the implementation of the approach. Section 7 describes our computational experimentation and results. Finally, our conclusions are given in section 8.

2 Literature Review

The literature review in this section gives a brief overview of the types of problems that have been solved in the field of academic timetabling. Ausgrid's timetabling problem is similar in certain ways to these problems and we can get some idea of what approaches are likely to work well and what approaches may not.

Problems in the area of academic timetabling have attracted a great deal of research attention over the last few decades[18]. In many cases the problem has been shown to be NP-Hard[6], often by relating it to the graph colouring problem[11]. For the classroom assignment problem (CAP) — the problem problem of assigning n classes to a set of m classrooms, in such a way that each class is run exactly once and each room can be used at most once per period — Carter proposes in [6] that there are three possible objectives: Feasibility, Satisfiability, and Optimisation. Feasibility asks whether there is any feasible solution given the constraints mentioned before, Satisfiability asks whether there is a feasible solution that puts each class into a satisfactory room, and Optimisation is the objective of minimising some linear cost function. Carter showed that the interval CAP satisfiability for even as little as two time periods, as well as the feasibility of the non-interval CAP, are NP-Complete.

Researchers involved with large-scale timetabling generally do not attempt to find optimal solutions to problems as they cannot be found in practically acceptable time due to the computational complexity. Instead, much of the recent research has been focused on approximation algorithms including metaheuristics[13][5], and decomposition methods such as Lagrangian relaxation[8] and column generation[19][17]. Many different timetabling problems can be expressed as graph colouring problems[14], and there has been some research activity in using graph colouring heuristics to solve timetabling problems[16][15][21][4].

A recent trend has been to develop so-called "hybrid heuristics" that combine certain features of one heuristic with another, with the aim of improving performance by overcoming a weakness in one or both of the heuristics. In [9], attempts were made to improve the convergence rate of SA by implementing the memory characteristics of tabu search (TS) to solve a university course timetabling problem. The annealing rate in SA, given by the cooling function, can have dramatic influence over the performance and success of an SA implementation[22]. It is not uncommon for people to implement complex reheating rules to help the heuristic avoid being trapped in local minima prematurely. A novel hybridisation was presented in [3], where the authors propose a Genetic Program (GP) to optimise the annealing schedule in simulated annealing. For several given problems, they presented the dedicated cooling schedules found by GP that converge fastest, and they also provided the cooling schedule that converges fastest across all problems.

Despite the difficulty in modelling and solving large-scale IP-based models, several researches have had some success. Perhaps the earliest examples are [12] from 1969 and [1] from 1973, where the authors increased tractability by grouping students together and using layouts for each group, and by solving assignment problems between sets of courses and time periods, respectively. A more recent example of IP-based university course and examination timetabling is presented in [7], where the authors augmented an IP with a heuristic improvement stage, and high school timetabling is presented in [2], where the authors were able to solve the IP directly with available computers. A full, integer linear programming formulation was presented in [20], which had, amongst others, 99 teachers, 156 courses, and 181 teaching groups. The model had 35,611 rows, 91,767 binary or integer variables, and 662,824 non-zeroes in the co-efficient matrix. An optimal solution was obtained in just 10 seconds using IBM ILOG CPLEX 9.1.2, or about 2 minutes with Coin-OR Branch and Cut (CoinCbc).

The high school and university timetabling problems discussed in the literature differ from Ausgrid's timetabling problem in many ways. Generally, university and high school timetabling is solved for just one or two weeks, that repeats throughout the semester or year. Ausgrid, on the other hand, cannot solve for short periods that will repeat; demand can fluctuate significantly from week to week or month to month. This, alone, makes the Ausgrid problem size significantly larger and existing solution approaches may not be suitable. In university and high school scheduling, a fixed set of courses must be allocated to a fixed set of rooms, whereas at Ausgrid, we do not know the number of times each course will run a priori. If we choose to run all the instances of a course in large rooms, we may only have to allocate very few instances. On the other hand, allocating the same course in the same time window in smaller rooms may require many more instances. This feature is unusual in the research area of academic timetabling. Courses in high schools and universities can generally start at arbitrary time periods, whereas courses at Ausgrid can start at permissible and, in certain cases, irregular periods. While it may appear that restricting the times at which courses can start would make the problem easier to solve, our analysis of scheduling problems with these constraints suggests this may not be the case and that restricting the set of starting times to irregular periods may actually be one of the sources of difficulty for our problem. Ausgrid timetabling also contains "no-wait" constraints, as modules within a course must run back-to-back; these constraints dramatically increase the problem complexity and are rarely, if ever, seen in high school and university timetabling. While splittable rooms do exist in many universities and high schools, we did not find many papers that mentioned them. Ausgrid has several splittable rooms and, especially in the high volume periods, efficient use of them is important. Another difference arises with the objectives of university and high school timetabling versus Ausgrid timetabling. The large majority of university and high school timetabling models try to maximise student and teacher satisfaction, which is given by how closely the solution meets their preferences. At Ausgrid we aim to provide training to meet the local demands, with minimal complicating factors in the courses (unnecessary swapping of rooms and/or trainers, etc.), and in ways economically beneficial for Ausgrid (for example, maximising the potential to rent out classrooms and auditoriums to third parties to bring extra revenue).

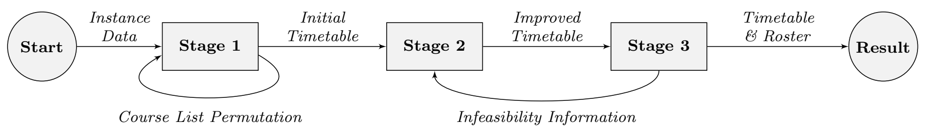


Fig. 1 A high-level view of the three-stage approach.

We believe our research problem is novel, interesting, and relevant to many related industrial applications as well. There are many electricity distributors worldwide as well as other industries that engage in periodic safety and/or technical training. Efficient solutions to timetabling problems like Ausgrid's can be of benefit to many organisations with similarly structured, periodic training requirements.

3 Optimisation Procedure

We have broken the main problem up into two sub-problems: a timetabling problem and a rostering problem. The timetabling problem is concerned only with the movement of shared, mobile resources, how many times each course should be run given the demand, and at what time and place their modules should be run. In the timetabling problem, trainers are considered in a generalised and aggregated way for capacity purposes only. The rostering problem is concerned with allocating individual trainers to individual modules, given a timetable of classes. The solution to both these sub-problems will yield a complete, functional timetable and roster. We have chosen to divide the main problem into these two components to improve the tractability of the problem, as well as the understandability and maintainability of the model.

To have a flexible tool that is able to solve these two complex sub-problems, we have developed two Integer Programming (IP) models that represent each of the two sub-problems. This approach, which is based on rigorous mathematical methods, guarantees, at least in principle, an optimal solution where one exists. IP is flexible in the sense that one can simply add, remove, or substitute constraints to modify the model in various ways; in contrast, problem specific computer algorithms may need more convoluted modifications even for minor changes to the problem. Even after dividing the problem, given Ausgrid's data, the IP models for the two sub-problems still have far too many variables and constraints to solve them in practically acceptable time.

In order to produce a complete, usable timetable and roster in acceptable time, we propose a three stage heuristic approach (see figure 1). The first stage produces an initial feasible timetable, the second stage attempts to improve the timetable as much as possible, and the third stage allocates individual trainers to the timetable.

3.1 First stage: Initial timetable construction

In the first stage, all course instances are placed in an ordered list and then arranged on the timetable one at a time. As more courses are placed on the timetable, rooms become unavailable at certain times, resources and trainers are consumed in particular locations at particular times, and less decisions need to be made for subsequent course instances, therefore the solving subsequent iterations becomes, in general, much easier. The course instances can be considered in arbitrary order, however certain rules may increase the likelihood of successfully finding a feasible timetable at the end of the first stage. For example, some course instances are considered easier to schedule if they have fewer modules and fewer resource requirements. Considering the course instances in a different order will likely yield a different timetable with a different cost, therefore we can generate several initial timetables by applying a different set of rules when constructing the course instance list. Furthermore, if short courses are dotted around the timetable before long courses are considered, then it is much more likely that there will not be a sufficiently long gap to fit a long course instance. Conversely, if the longest courses are considered first, then the shorter ones will have a greater chance of fitting into the remaining gaps.

The individual course instances are assigned one at a time by solving the timetabling IP model. Only those variables and constraints that are related to the course instance being assigned are included in the model. Any rooms unavailable as a result of previously assigned course instances are also omitted from the model during their periods of unavailability. The resulting IP model is much smaller and easier to solve. To the best of our knowledge, even the single course instance problem is quite challenging to solve and we know of no polynomial time algorithm that will produce an optimal solution. We are not currently considered using a heuristic approach for the single instance problem as allocating one instance at a time is already heuristic at best.

To further reduce the size of the IP model, we consider a narrower planning horizon for the course instance we wish to allocate. Knowing course instances should be spread out uniformly in the ideal case, we can estimate when the course instances should run. Since, however, we cannot guarantee that the course can be scheduled at the times we expect, the set of considered time periods should have a buffer at each end to allow some freedom in scheduling (see figure 2). The shorter the buffer, the more control we have over the precise timing of the course, and the less variables will be in the model, but the probability the solver will fail to find a feasible solution will be increased. On the other hand, having longer buffers requires more variables to be included in the model, allows greater freedom for the solver in scheduling the courses, meaning it will be easier for the solver to find a feasible placement given existing allocations. For a course instance c with duration l_c, we considered initial buffers of length l_c meaning that, if we expect the course to start at period τ, then the planning horizon initially contains periods $\tau - l_c$ up to $\tau + 2l_c$. If no feasible solution can be found, we gradually expanded the buffers in both directions until a feasible solution is produced.

When scheduling course instances one-by-one, we need not consider all locations at once either. It is possible to look at the demand information for a course and the state of the current partial solution to decide in which region the next course instance will be placed, and include only those locations which belong to that region in the IP model.

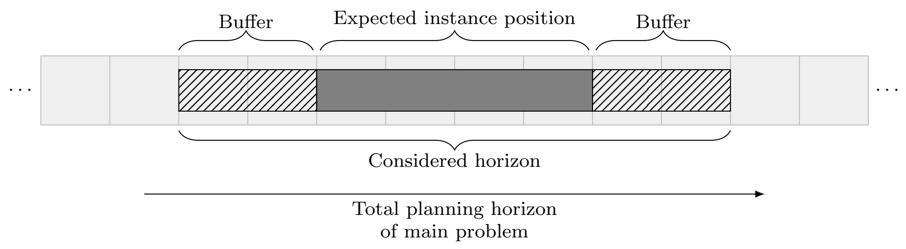

Buffer
Expected instance position
Buffer
...
...
Considered horizon
Total planning horizon of main problem

Fig. 2 A buffer added to either end of the reduced timeline.

The final IP model for each iteration of the first stage becomes substantially smaller and can be solved in just a few seconds. It contains only one course instance with a small number of modules (typically 1 to 5), only a short planning horizon (typically 24 to 144 time periods), only one region with a small number of locations and rooms (typically 1 to 4 locations with 1 to 15 rooms in total).

3.2 Second stage: Timetable improvement

While the first stage can produce an initial timetable relatively quickly, it is unlikely to find the optimal solution to the whole timetabling problem. Poor decisions made at the early stages of the process can have a compounding effect on the remaining allocations, leading to poor quality timetables.

The second stage attempts to improve the timetable, using an iterative LNS heuristic. At the beginning of each iteration, a computer algorithm specifically tailored to the objective being considered scans the current solution and attempts to identify the components which contribute most to the objective function. For example, suppose one course instance contributes a lot to the objective function because it contains several room swaps. In order for this iteration of the improvement stage to remedy this, additional degrees of freedom must be created. A new timetabling model is constructed for this iteration for the location of the course instance, and for the day in which the course instance is currently allocated, optionally with a buffer at either end. All course instances for this location in this reduced planning horizon are removed from the timetable and re-solved simultaneously. A list of previous states must be stored in memory to prevent cycling, much like a Tabu list.

A feasible solution is guaranteed at each iteration, which must be no worse than that of the previous iteration. At iteration $i-1$, most courses were considered as constants in the model, while some were variables that were optimised over the objective function to produce the solution $S^{(i-1)}$. In the subsequent iteration i, a different set of courses are considered constants and a different set are variables, however the model is optimised over the same objective function. Since the solution $S^{(i-1)}$ is feasible (and optimal given the model at iteration $i-1$), $S^{(i-1)}$ must also be feasible at iteration i and will necessarily have the same objective value. However, it is possible that $S^{(i)} \neq S^{(i-1)}$ and then it is possible that the objective value at iteration i will be better than it was at $i - 1$, however it can never be worse.

The features of timetables that stage 2 is currently set up to attempt improvement over are:

– Course instances with excessive room swaps;
– Course instances using rooms larger than necessary when smaller rooms exist;
– Periods (or days) with unbalanced (too many or too few) allocations; and
– Courses with too many instances.

The IP model is again used at each iteration when attempting to improve the timetable. At each iteration, after the scanning algorithm identifies a component of the timetable that may be improved, a new IP model is set up in which the majority of the timetable at the previous iteration are not included as decision variables, only their student capacities, and room and trainer consumption is included as constants in the model. The only decision variables included in the IP model are the ones that pertain to the course instances being improved. Given the course instances that are left as decision variables, the "large neighbourhood" in the large neighbourhood search is the set of all feasible solutions to the IP model.

3.3 Third stage: Rostering

Once the first two stages have completed, the timetable is assumed to be finalised, only requiring individual trainers to be allocated to specific modules. The third and final stage constructs an IP model taking into consideration all the rostering requirements. Given a typical monthly timetable from Ausgrid, the rostering IP model can be solved directly by commercial IP solvers. The first two stages utilise the timetabling IP model to generate solutions, whereas the third stage uses the rostering IP model.

The objective of the rostering IP model is to minimise the flow cost along the networks. The flow cost of each arc on the network is given by a linear combination of the travel cost and trainer swap cost, either or both of which may be zero. While worker pay is often a high cost component in many other problems, we do not consider it in our rostering subproblem as Ausgrid trainers are paid a fixed salary regardless of what courses the do or do not teach, although the case of considering workforce pay may be a consideration in future research.

It is possible, though unlikely, that the timetable produced in the first two stages has no feasible solution with respect to trainers in the third stage. If no feasible solution can be found to the rostering IP model in the third stage, an algorithm analyses the distribution of courses identifies which trainer capacity constraints are violated. For instance, if there are n trainers capable of teaching a particular set of modules, however there are $m > n$ of those modules running at a particular time period, then each of the course instances of those modules should be considered for rescheduling. The algorithm briefly returns to stage two and solves the model with the additional constraint that, at any given time, the total number of times modules of this type can be run must not exceed n.

4 Timetabling Model

4.1 Time Discretisation

In the timetabling model time is discretised into periods of half-hour blocks of time. Periods that are not available for training are not considered, which include the after-hours interval, lunch time and public holidays. Periods are grouped into coarser intervals called days, each of which contain exactly 16 periods starting at 07:30 and ending at 16:00 with a break from 12:00 to 12:30. Many days are grouped together into time windows, which represent longer durations such as a week or a month. For practical purposes, we consider a window to start on the first working day of the calendar month, end on the last working day of the calendar month, and only include working days. This usually ends up being about 20 working days in a calendar month. Finally, periods are grouped together into rental windows. Rental windows can be as short as half a day, but can be as long as one or more days. A rental window represents a set of periods in which a room can be rented out to a third party. Once the room has been rented out, the room becomes unavailable for Ausgrid. Renting out larger rooms brings in more revenue, as does renting them out for longer periods of time.

4.2 Input Data Set-up

Once the raw problem instance data is imported, some pre-processing must first be done. For the purpose of the timetabling model, we fix the number of instances of each course to a practical estimation. In practice, one module can exist in many different courses, for example, a basic first aid module may be a component in several courses. For our model, however, each module must belong to exactly one course instance - these are known as module instances.

Each location may have several rooms, however some rooms may be regarded as identical - they have the same compatible list of modules, the same physical size, the same number of seats, etc. In this case, we do not need to consider individual rooms; instead we can consider room types, where each room type represents a set of rooms in a given location which are functionally identical. Rooms in different locations are not grouped together, and rooms that are in the same location but are part of a compound room set (those rooms with removable dividers), cannot be aggregated into room types; each piece of a compound room forms its own rooms type with only one room.

Compound rooms can be modeled using a set of mutually exclusive room pairs. Suppose room C can be split into two smaller rooms A and B. Rooms A and B can be used simultaneously, however the use of A is mutually exclusive with that of C, as is the the use of B with that of C.

4.3 List of Symbols

Sets of primary objects:

P	The indexed set of periods.
D	The indexed set of days.
Ω	The indexed set of time windows.
Λ	The indexed set of rental windows.
L	The set of locations.
Ξ	The set of regions.
$\hat{M}$	The indexed set of module instances.
C	The set of courses.
R	The set of room types.
T	The set of resources types.
$S^{(i)}$	The i^{th} element of an indexed set S.

Sets of derived objects:

I_c	The indexed set of instances for course $c \in C$.
$B_{c,i}$	The indexed set of modules instances for course $c \in C$ instance $i \in I_c$.
P_d	The indexed set of periods in day $d \in D$.
P_ω	The indexed set of periods in time window $\omega \in \Omega$.
P_λ	The indexed set of periods in rental window $\lambda \in \Lambda$.
$\hat{P}_c$	The set of periods in which course $c \in C$ may start.
L_ξ	The set of locations in region $\xi \in \Xi$.
$\bar{R}$	The set of mutually exclusive room pairs.
$\tilde{R}_l$	The set of room types in location $l \in L$.
R_m	The set of room types suitable for module $m \in \hat{M}$.

Primary decision variables:

$X_{m,r,p}$	1 if module $m \in \hat{M}$ runs in a room of type $r \in R$ starting at period $p \in P$, or 0 otherwise.
$Y_{c,i,p}$	1 if course $c \in C$ instance $i \in I_c$ starts at period $p \in P$, or 0 otherwise.
$\hat{Y}_{c,i,l}$	1 if course $c \in C$ instance $i \in I_c$ runs in location $l \in L$, or 0 otherwise.
$\psi_{t,l,k,p}$	The quantity of resource $t \in T$ moving from location $l \in L$ to location $k \in L$ (l and k may be the same), starting at period $p \in P$.
$\hat{\psi}_{t,l,k,d}$	The quantity of resource $t \in T$ moving from location $l \in L$ to location $k \in L$ (l and k may be the same), overnight at the end of day $d \in D$.
$\phi_{l,d}$	The number of trainers assigned to location $l \in L$ on day $d \in D$.

Auxiliary variables:

$\hat{X}_{m,r,p}$	1 if module $m \in \hat{M}$ runs in a room of type $r \in R$ during period $p \in P$, or 0 otherwise.

$\bar{Y}_{c,i}$	1 if course $c \in C$ instance $i \in I_c$ runs, or 0 otherwise.
$\tilde{Y}_{c,i,\omega,\xi}$	The number of students expected to sit in course $c \in C$ instance $i \in I_c$ during time window $\omega \in \Omega$ in region $\xi \in \Xi$.
$u_{c,\omega,\xi}$	The number of students not accommodated for course $c \in C$ during window $\omega \in \Omega$ in region $\xi \in \Xi$.
t_m	The new room flag for module $m \in \hat{M}$. If the room type for this module is that same type of room as for the previous module, if applicable, then $t_m = 0$, otherwise $t_m = 1$.
$\rho_{r,\lambda}$	The number of rooms of type $r \in R$ occupied during rental window $\lambda \in \Lambda$.
Z_i	The ith goal term in the objective function.

Constants:

σ_t	The quantity available of resource $t \in T$.
$\delta_{t,l,k}$	The time required (in periods) for a unit of resource $t \in T$ to move from location $l \in L$ to location $k \in L$.
$\theta_{l,d}$	The number of trainers normally allocated to location $l \in L$ on day $d \in D$.
$\theta^{\max}$	The maximum number of additional trainers permitted to any location on any given day.
$\theta^{\min}$	The maximum number of subtracted trainers permitted from any location on any given day.
$Q_{r,p}$	The quantity of room type $r \in R$ available at period $p \in P$, or 0 otherwise.
$d_{r,\lambda}$	The expected revenue from renting out a unit of room type $r \in R$ during rental window $\lambda \in \Lambda$.
l_c	The length (in periods) of course $c \in C$.
b_c	The minimum class size (in students) required to justify running an instance of course $c \in C$.
π_c	The length (in periods) of the rolling time window used to compute the minimum and maximum number of times a course $c \in C$ should be run.
π_c^+	The maximum number of times a course $c \in C$ should be run in any given time window of length π_c.
π_c^-	The minimum number of times a course $c \in C$ should be run in any given time window of length π_c.
$s_{c,\omega,\xi}$	The demand (in students) for course $c \in C$ during window $\omega \in \Omega$ in region $\xi \in \Xi$.
α_i	The coefficient of the ith goal in the objective function.

4.4 Core Timetabling Constraints

The following constraints express the core requirements of the timetabling problem, and are likely to appear in many similar course timetabling problems:

$$\hat{X}_{m,r,p} = \sum_{q=0}^{d_m-1} X_{m,r,(p-q)} \qquad \forall m \in \hat{M}, r \in \hat{R}_m, p \in P \quad (1)$$

$$\sum_{m \in \hat{M}} \hat{X}_{m,r,p} \leq Q_{r,p} \qquad \forall r \in R, p \in P \quad (2)$$

$$\sum_{m \in \hat{M}} \hat{X}_{m,\tilde{r}_1,p} + \sum_{m \in \hat{M}} \hat{X}_{m,\tilde{r}_2,p} \leq 1 \qquad \forall \{\tilde{r}_1, \tilde{r}_2\} \in \tilde{R}, p \in P \quad (3)$$

$$\sum_{p \in \hat{P}_c} Y_{c,i,p} \leq 1 \qquad \forall c \in C, i \in I_c \quad (4)$$

$$\sum_{r \in R} \sum_{p \in P} X_{m,r,p} = \bar{Y}_{c,i} \qquad \forall c \in C, i \in I_c, m \in B_{c,i} \quad (5)$$

$$\bar{Y}_{c,i} = \sum_{p \in \hat{P}_c} Y_{c,i,p} \qquad \forall c \in C, i \in I_c \quad (6)$$

The auxiliary variables $\hat{X}_{m,r,p}$ are set up from $X_{m,r,p}$ according to (1). The constraints (2) express the requirement that rooms should not be double-booked, however since identical rooms within a single location are aggregated together, the right-hand-side is given by the quantities of the aggregated rooms. The constraints (3) also express the requirement that splittable rooms should not be double-booked, however since splittable rooms are never aggregated together, the right-hand-side remains 1. The constraints (4) ensures each course instance can start at most once, and (5) ensure that each module of a course is run exactly once if the course is run, or not at all. The expression (6) sets up the $\bar{Y}_{c,i}$ variable, which is a sum over all periods of $Y_{c,i,p}$.

4.5 Characteristic Constraints

The remaining constraints express the operational requirements that are rarely found in traditional timetabling problems.

4.5.1 Module Positioning Constraints

$$\sum_{m \in B_{c,i}} \sum_{r \in R_m} \hat{X}_{m,r,p} = \sum_{q=0}^{l_c-1} Y_{c,i,(p-q)} \qquad \forall c \in C, i \in I_c, p \in P \quad (7)$$

$$\sum_{m \in B_{c,i}} \sum_{r \in \tilde{R}_l} \sum_{p \in P} X_{m,r,p} = |B_{c,i}| \times \hat{Y}_{c,i,l} \qquad \forall c \in C, i \in I_c, l \in L \quad (8)$$

The constraints (7) expresses the requirement that the modules for a course run back-to-back, and (8) expresses the requirement that all the modules for a course must be run in exactly one location.

4.5.2 Capacity Constraints

The following constraints determine the capacity of each course instance in each time window and region based on the values of the X and Y variables:

$$\tilde{Y}_{c,i,\omega,\xi} \leq \sum_{r \in R_m \bigcap R_l} \sum_{p \in \bar{P}_w} (min\{u_m, v_r\} \times X_{m,r,p}) \quad \forall c \in C, i \in I_c, \omega \in \Omega, \xi \in \Xi \tag{9}$$

$$b_c \times \bar{Y}_{c,i} \leq \sum_{\xi in \Xi} \sum_{\omega \in \Omega} \tilde{Y}_{c,i,\omega,\xi} \qquad \forall c \in C, i \in I_c \tag{10}$$

The constraints (9) set up the $\tilde{Y}_{c,i,\omega,\xi}$ variables, and (10) ensure the capacity of a course is at least as great as the minimum allowable class size.

4.6 Trainer Movement Constraints

Trainers are considered in a generalised, aggregated way for capacity purposes only. Nevertheless, we permit the quantity of these generalised trainers to change per location per day to give a coarse representation of trainer movements. Each trainer has a location where they are normally based, however they may be required to travel to other locations. The total quantity of trainers at location $l \in L$ on day $d \in D$, by default, is given by the constant $\theta_{l,d}$ (we have specified a subscript for days so we can subtract trainers who are unavailable, such as trainers on annual leave, etc.).

$$\phi_{l,d} \leq \theta_{l,d} + \theta^{\max} \qquad \forall l \in L, d \in D \tag{11}$$

$$\phi_{l,d} \geq \theta_{l,d} - \theta^{\min} \qquad \forall l \in L, d \in D \tag{12}$$

$$\sum_{l \in L} \phi_{l,d} = \sum_{l \in L} \theta_{l,d} \qquad \forall d \in D \tag{13}$$

$$\sum_{m \in \hat{M}} \sum_{r \in \tilde{R}_l} \hat{X}_{m,r,p} \leq \phi_{l,d} \qquad \forall l \in L, d \in D, p \in P_d \tag{14}$$

The constraints (11) and (12) establish the minimum and maximum number of trainers permitted to be at a given location on a given day, and the constraints (13) ensures that the total number of trainers allocated to each location is equal to the total number of trainers expected to be working company-wide on that day. The constraints (14) express the requirement that, at any given time, the total number of modules run in a location concurrently must not exceed the number of generalised trainers we have chosen to allocate there.

4.7 Resource Movement Constraints

A network formulation can be leveraged to represent the flow of resources between locations across time in a convenient way. Resources, in this context, refer to mobile pieces of equipment that are required for teaching particular modules. One such example mentioned in section 1 is that of a set of different fire fighting modules which require some fire fighting equipment in order to run; in this case we can use a flow network to represent the movement of the fire fighting equipment between the training facilities across time. For each type of resource and for each day, we construct a flow network with the nodes arranged in a rectangular lattice (See figure 3). The horizontal axis represents time, and the vertical axis represents the various locations. Each node represents the end points of a time period at a given location (note that the end of one time period is equivalent to the beginning of the next, consecutive time period). Adjacent nodes are connected by directed arcs horizontally and pointing forward in time, with the flow along those arcs representing the quantity of the resource available at a particular location at a particular time. Nodes are also connected between different locations by directed arcs in such a way that the time interval from the source node to the destination node is given by the time required to move the resource from the source to the destination locations.

We permit any resource to move from any location to any other other location overnight at no cost, therefore the initial condition for each resource network on each day is simply that the sum across all location must equal the quantity of the particular resource in Ausgrid's possession.

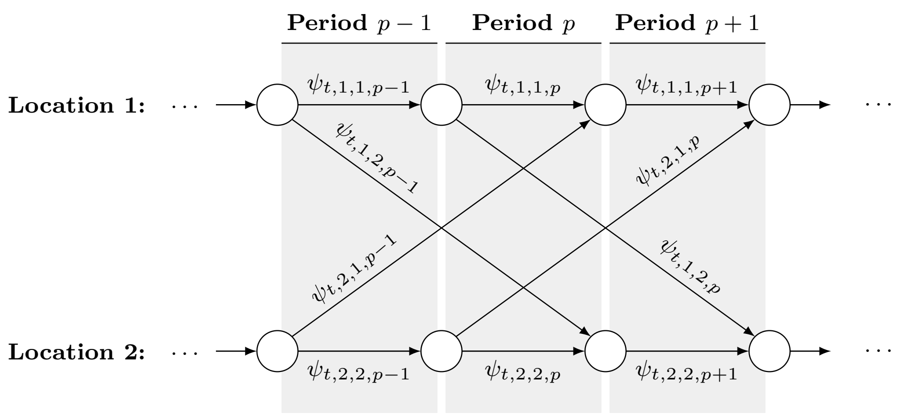


Fig. 3 A sample flow network for some resource t about period p with 2 locations.

If $l = k$, the variables $\psi_{t,l,k,p}$ represents the quantity of resource $t \in T$ available at location $l \in L$ during time period $p \in P$. If $l \neq k$, the variable represents the quantity of the resource moving from location $l \in L$ to location $k \in L$ starting its journey at $p \in P$. Since the transport time of resource $t \in T$ from $l \in L$ to $k \in K$ is given by $\delta_{t,l,k}$, the arc represented by $\psi_{t,l,k,p}$ will be connected to the node that represents the start of period $p + \delta_{t,l,k}$.

The flow balance equations for the resource movement network are expressed as follows:

$$\sum_{l\in L}\sum_{k\in L}\psi_{t,l,k,P_d^{(1)}} = \sigma_t \qquad \forall t\in T, d\in D \tag{15}$$

$$\sum_{k\in L}\psi_{t,k,l,(p-\delta_{t,k,l})} = \sum_{k\in L}\psi_{t,l,k,p} \quad \forall t\in T, l\in L, d\in D, p\in (P_d\setminus P_d^{(1)}) \tag{16}$$

Now that we have constraints that govern the movement resources between locations across time, we ensure that the number of times we run modules is limited by these quantities:

$$\sum_{m\in\hat{M}_t}\sum_{r\in\tilde{R}_l}\hat{X}_{m,r,p} \le \psi_{t,l,l,p} \quad \forall l\in L, t\in T, p\in P \tag{17}$$

4.8 Spreading Constraints

For each course, we have a defined minimum and maximum number of instances that may be run in any arbitrary set of consecutive periods of a predetermined length:

$$\sum_{i\in I_c}\sum_{q=0}^{\pi_c} Y_{c,i,p+q} \ge \pi_c^- \quad \forall c\in C, p\in\hat{P}_c \tag{18}$$

$$\sum_{i\in I_c}\sum_{q=0}^{\pi_c} Y_{c,i,p+q} \le \pi_c^+ \quad \forall c\in C, p\in\hat{P}_c \tag{19}$$

The constraints (18) and (19) establish the minimum and maximum number of instances, respectively, that must be run across all regions for each course. Selection of the π_c and the π_c^- and π_c^+ constants is made given the problem data.

4.9 Objective Function

Being a large-scale industrial problem, there are many potential objectives we can consider. In this paper, we consider three objectives:

– Minimise the number of students not accommodated;
– Maximise the rental revenue;
– Minimise the number of room swaps in the timetable;

The objective function is the weighted linear combination of these three objective. The weight for the first objective, which is to minimise the number of students for whom there are no spots in any classes, has a much higher weight than the weights for the remaining objectives, because it is extremely undesirable if this happens.

The first objective, denoted by Z_1, is to minimise the number of students not accommodated:

$$\sum_{i\in I_c} \tilde{Y}_{c,i,\omega,\xi} + u_{c,\omega,\xi} - o_{c,\omega,\xi} = s_{c,\omega,\xi} \quad \forall c \in C, \omega \in \Omega, \xi \in \Xi \tag{20}$$

$$u_{c,\omega,\xi} \geq 0 \quad \forall c \in C, \omega \in \Omega \tag{21}$$

$$o_{c,\omega,\xi} \geq 0 \quad \forall c \in C, \omega \in \Omega \tag{22}$$

$$\sum_{c\in C}\sum_{\omega\in\Omega}\sum_{\xi\in\Xi} u_{c,\omega,\xi} = Z_1 \tag{23}$$

The second objective, denoted by Z_2, is to maximise the rental revenue:

$$\sum_{m\in\hat{M}} \hat{X}_{m,r,p} \leq \rho_{r,\lambda} \quad \forall r \in R, \lambda \in \Lambda, p \in \hat{P}_\lambda \tag{24}$$

$$Z_2 = \sum_{r\in R}\sum_{\lambda\in\Lambda} [-d_{r,\lambda} \times (\hat{Q}_{r,\lambda} - \rho_{r,\lambda})] \tag{25}$$

where $\hat{Q}_{r,\lambda}$ is the smallest value of $Q_{r,p}$, $\forall p \in \hat{P}_\lambda$ for each $\lambda \in \Lambda$.

The last objective, denoted by Z_3, is to minimise the number of room swaps across all courses:

$$X_{m,r,p} - \sum_{n\in B_{c,i}, m\neq n} \hat{X}_{n,r,(p-1)} \leq t_m \qquad \forall c \in C, i \in I_c, m \in B_{c,i}, r \in R_m, p \in P \tag{26}$$

$$Z_3 = \sum_{m\in\hat{M}} t_m \tag{27}$$

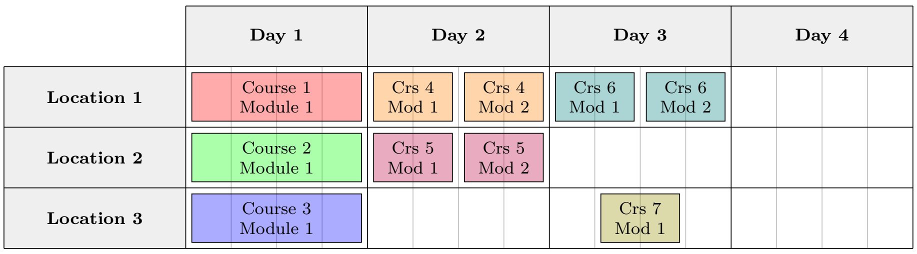
The objective function is a weighted linear sum of the the individual objectives:

$$\text{minimise:} \quad Z = \alpha_1 Z_1 + \alpha_2 Z_2 + \alpha_3 Z_3 \tag{28}$$

with weights $\alpha_1 >> \alpha_2$ and $\alpha_1 >> \alpha_3$.

5 Rostering Model

Given a solution to the class timetabling problem from section 4, the rostering model describes the task of allocating specific trainers to modules to form a complete, usable timetable and roster. A minimum cost network flow approach, together with some side constraints, can be utilised to give a simple, convenient representation of the rostering problem.



	Day 1	Day 2	Day 3	Day 4
Location 1	Course 1 Module 1	Crs 4 Mod 1; Crs 4 Mod 2	Crs 6 Mod 1; Crs 6 Mod 2	
Location 2	Course 2 Module 1	Crs 5 Mod 1; Crs 5 Mod 2		
Location 3	Course 3 Module 1		Crs 7 Mod 1	

Fig. 4 A sample timetable, simplified for viewing in this format, showing 4 days, 3 locations, and 7 courses each with 1 or 2 modules.

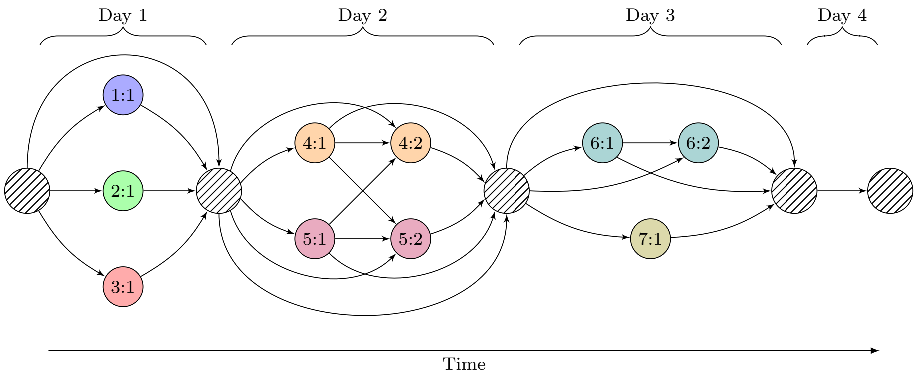


Fig. 5 The flow network corresponding to the sample timetable shown in figure 4. (Home nodes are hatched, and activity nodes are solid)

Given a timetable, a flow network is constructed for each trainer, which is referred to as a trainer allocation network. There are two different types of nodes in the trainer allocation network: home nodes and activity nodes. Home nodes can represent either the trainer's own home, or their usual place of work. Activity nodes represent specific modules that can be taught by the trainer. There are four different types of arcs in the trainer allocation network: commencement arcs, transition arcs, return arcs, and bypass arcs. Commencement arcs are those that originate from the home nodes and end at the activity nodes; they are called commencement arcs because they represent the first module the trainer will teach on a particular day. Transition arcs are those that originate from activity nodes and end at activity nodes; they represent a trainer completing one module and starting another, although trainers do not need to be allocated to modules back-to-back—they may have a gap after teaching one module and before teaching the next. Return arcs are those that originate from activity nodes and end at home nodes; they represent the last module a trainer will teach on a particular day. Bypass arcs are those that originate at home nodes and end at home nodes; they represent a trainer having no allocations on a particular day.

As an example, Figure 4 shows a simplified view of a timetable with 7 courses, with 1, 1, 1, 2, 2, 2, and 1 modules, respectively, and the corresponding flow network is shown in Figure 5.

Costs on the arcs of the network are determined by two factors. The first factor is determined by the distance the trainer needs to travel for the allocation, including travel to the first module taught in a day, travel from the last module taught in a day, and also travel from module to module. The second factor is the trainer swap cost. A trainer swap happens if the arc starts with an activity node which is not the last module of a course instance, but ends with a home node or an activity node from a different course instance.

We introduce the following symbols for the rostering model:

Set	**Description**
D	The indexed set of days.
$\hat{M}$	The indexed set of module instances.
M_d	The set of modules that run within day $d \in D$
T	The set of trainers.
T_m	The set of trainers capable of teach module $m \in \hat{M}$.
$pred(m)$	The set of predecessors of module $m \in \hat{M}$.
$succ(m)$	The set of successors of module $m \in \hat{M}$.

Variable	**Description**
$\bar{\psi}_{\tau,m}$	1 if trainer τ teaches module m as their first module on that day, or 0 otherwise.
$\psi_{\tau,m,n}$	1 if trainer τ teaches module m followed by module n, or 0 otherwise.
$\tilde{\psi}_{\tau,m}$	1 if trainer τ teaches module m as their last module on that day, or 0 otherwise.
$\hat{\psi}_{\tau,d}$	1 if trainer τ doesn't teach any modules on day d, or 0 otherwise.

The flow balance equations for the network are as follows:

$$\hat{\psi}_{\tau,d} + \sum_{m \in M_d} \bar{\psi}_{\tau,m} = 1 \qquad \forall \tau \in \mathrm{T}, d \in D \tag{29}$$

$$\bar{\psi}_{\tau,m} + \sum_{n \in pred(m)} \psi_{\tau,n,m} = \sum_{n \in succ(m)} \psi_{\tau,m,n} + \tilde{\psi}_{\tau,m} \qquad \forall \tau \in \mathrm{T}, m \in \hat{M} \tag{30}$$

where (29) ensures that, at the start of each day, the trainer either teaches one or more modules or does not teach any modules; and (30) conserves flow throughout the day. Since the flow for each day is implicitly conserved by (29) and (30), we do not require any additional equations to balance the flow from day to day.

It is important to note that *any* integral flow is always a feasible line of work for a single trainer, i.e. the network is constructed in such a way the trainer will never be required to be in two places at once, nor will the trainer be required to teach a module they are not capable of teaching.

The individual trainer networks on their own are not sufficient to guarantee a feasible solution to the rostering problem as multiple trainers may be allocated to the same module, or modules may be left with no trainer at all. We introduce some side constraints that integrate the many trainer networks into a single IP model.

$$\bar{\psi}_{\tau,m} + \sum_{n \in pred(m)} \psi_{\tau,n,m} = X_{m,\tau} \qquad \forall m \in \hat{M}, \tau \in \mathrm{T} \tag{31}$$

$$\sum_{\tau \in \mathrm{T}_m} X_{m,\tau} = 1 \qquad \forall m \in \hat{M} \tag{32}$$

where (31) sets up the auxiliary variable $X_{m,t}$, which is 1 if trainer t teaches module m or 0 otherwise, and (32) ensures that every scheduled module is taught be exactly one trainer.

Fairness is important when rostering at Ausgrid, and we wish to avoid, wherever possible, the situation where one trainer is scheduled to train more or less than their peers.

$$U_\tau^- \leq \sum_{m \in \hat{M}} (w_m \times X_{m,\tau}) \leq U_\tau^+ \quad \forall \tau \in \mathrm{T} \tag{33}$$

where U_t^- and U_t^+ are the minimum and maximum number of periods, respectively, that we permit trainer $t \in T$ to teach.

The objective of the rostering problem is to minimise the flow cost all networks simultaneously.

$$\begin{aligned} \min \sum_{\tau \in \mathrm{T}} \sum_{m \in \hat{M}} [c_1(\tau, m) \times \tilde{\psi}_{\tau,m}] + \sum_{\tau \in \mathrm{T}} \sum_{m \in \hat{M}} \sum_{n \in \hat{M}} [c_2(\tau, m, n) \times \psi_{\tau,m,n}] + \\ \sum_{\tau \in \mathrm{T}} \sum_{m \in \hat{M}} [c_3(\tau, m) \times \bar{\psi}_{\tau,m}] + \sum_{\tau \in \mathrm{T}} \sum_{d \in D} [c_4(\tau, d) \times \hat{\psi}_{\tau,d}] \end{aligned} \tag{34}$$

where $c_1(\cdot)$, $c_2(\cdot)$, $c_3(\cdot)$, and $c_4(\cdot)$ give the flow costs of the commencement, transition, return, and bypass arcs, respectively, where the flow costs are characterised by any applicable trainer travel costs and trainer swap costs.

6 Implementation

6.1 Pre-Processing

In order to increase the tractability of our model, we wish to eliminate as many variables and constraints as possible. We can reduce the set of permissible start times for each module in a given course by identifying all the possible times the module can start relative to the start time of the course. Since each of those modules belongs to a particular course instance which does have a set of permissible start times, the modules implicitly inherit a restriction on when they may start.

Suppose a course c instance i has a set of modules $\{m_1, m_2, \ldots, m_{|B_{c,i}|}\}$. Since the order of the modules is unrestricted, there are $B_{c,i}!$ possible permutations we can choose to run the modules. For each permutation, each module has a starting offset—the amount of time, in periods, between the course start time and the module start time. If, for each permutation and for each module, we identify the unique starting offsets, we can apply those offsets (which are in relative time, relative to the time in which the course starts), to each permissible start time of the parent course in order to enumerate the complete set of time periods in which the modules may start.

6.2 Symmetry Breaking

One weakness of the timetabling model is the symmetry present in the solution space. In many cases, objects can be arranged in a variety of ways, where each configuration has no dominance over the rest. Suppose we start with a timetable where course c instance 1 is run on Monday and instance 2 is run on Wednesday. If we keep all practical aspects about the timetable the same, however we now refer to the Monday course as instance 2 and the Wednesday course as instance 1, then there is no difference in terms of solution cost. There are $n!$ ways of indexing the n instances of a single course across an existing timetable.

We may eliminate many symmetric solutions by introducing the following constraints:

$$\sum_{q=0}^{p} Y_{c,i,q} \geq Y_{c,(i+1),p} \quad \forall c \in C, i \in I_c, p \in P_c \tag{35}$$

which ensures that, for any given course, instance i must be run in order to run instance $i+1$, and also that instance i must be run no later than instance $i+1$.

There are many other sources of symmetry in the model, however we will not discuss these in this paper.

7 Computational Experiments

Ausgrid's training department supplied both current and historical data. We worked with their 8 most frequently run courses with module numbers ranging from 1 to 4, and instance numbers ranging from 1 to 26. The planning horizon we considered was 1 month, with a total of 23 working days. Across 5 regions, there were 15 locations with room counts ranging from 1 to 8, and 12 composite rooms. With 8 working hours per day, not including meal breaks, the planning horizon was divided into 368 half-hour time periods, 69 rental windows, and 1 demand window. There were 21 trainers spread across 11 of the 15 locations, and each trainer was qualified to teach between 8 and 14 modules. Two trainers had physical disabilities that prevented them from travelling longer distances, and we enforced this requirement by removing those arcs from the trainer allocation networks that would require them to travel further.

We used IBM ILOG CPLEX 12.5.0.0[10] on an Intel i7-2640M dual-core 2.8Ghz system with 4GB DDR3 RAM, running Windows 7 Professional 64-bit, Service Pack 1. The model was dynamically constructed from data files supplied by Ausgrid using a program we developed in C# 4.0, interacting with CPLEX using the IBM ILOG Concert API. Setting up the timetabling model directly for all courses for the entire month resulted in, on average, over 3 million variables and we were unable to obtain solutions at all. Setting up the timetabling model directly for all courses for a planning horizon of 5 days resulted in, on average, about $600,000$ variables and it took 88 hours to arrive at the optimal solution. Our prior experimentation with smaller test cases indicated that disabling all automatic cut generation and using CPLEX's aggressive probing yielded the fastest solution times.

Prior to investigating a mathematical programming approach to Ausgrid's scheduling problem, a list-based constructive software system was developed to automate the generation of timetables on a month-by-month basis. The software does not perform any optimisation directly, focusing instead on producing a feasible timetable quickly, with a "good" use of resources where possible. The problem being solved in the existing system has some tighter assumptions based on current practice, whereas the model discussed in this paper is more general. The existing software system is written in in C# 4.0 running on the same machine. It is a collection of algorithms that constructs a full timetable including trainer roster for a given month.

We chose the objective weights to be $\alpha_1 = 10^6$, $\alpha_2 = 1$, and $\alpha_3 = 6$, based on empirical testing and inspection of the produced timetables.

Table 6 shows the results for a timetable based on a one-year data set from 2012, generated by the existing software system. From left-to-right, the columns show the month of the year solved, the number of courses placed on the timetable, the partial cost of the timetable contributed by the first, second, and third goals respectively, the weighted linear sum of the three timetabling optimisation goals, the cost of the trainer roster, and the combined cost of the timetable and roster.

	Num. Courses	Z_1	Z_2	Z_3	Timetable	Roster	Total
Jan	47	0	-22.58	6	13.42	1580.61	1594.03
Feb	61	0	-6.30	7	35.70	1791.57	1827.27
Mar	49	0	-10.68	9	43.32	1908.55	1951.87
Apr	38	0	-32.78	5	-2.78	1170.40	1167.62
May	33	0	-26.55	3	-8.55	853.71	845.16
Jun	31	0	-14.27	5	15.73	789.33	805.06
Jul	58	0	-8.73	9	45.27	1607.18	1652.45
Aug	51	0	-9.73	7	32.27	1254.27	1286.54
Sep	44	0	-17.10	3	0.90	1146.64	1147.54
Oct	42	0	-22.10	6	13.90	1141.56	1155.46
Nov	47	0	-11.96	8	36.04	1945.33	1981.37
Dec	34	0	-30.80	4	-6.80	1238.28	1231.48

Table 6 Results for the existing software system.

The existing system is able to produce a solution for one month in between 3.196 and 7.603 seconds, depending on the density of courses in the month. The system was able to produce a feasible solution (where all students are accommodated) for all months in all regions that we tested.

Table 7 shows the results for a timetable based on the same data set, generated by the 3-stage heuristic. From left-to-right, the columns show the month of 2012 that was solved, the number of courses placed on the timetable, the cost of the timetable after initial timetable generation, the partial cost of the improved timetable contributed by the first, second, and third goals respectively, the weighted linear sum of the three timetabling goals after the improvement stage, the cost of the trainer roster, and the combined cost of the timetable and roster.

	Num. Courses	Stage 1	Z_1	Z_2	Z_3	Stage 2	Stage 3	Total
Jan	47	-22.81	0	-66.38	5	-36.38	977.6	941.22
Feb	61	35.57	0	-15.26	6	20.74	1509.75	1530.49
Mar	49	-13.02	0	-66.95	7	-24.95	1465.59	1440.64
Apr	38	-62.96	0	-99.12	4	-75.12	970.52	895.4
May	33	-61.56	0	-93.17	3	-75.17	596.64	521.47
Jun	31	-86.42	0	-125.54	4	-101.54	781.51	679.97
Jul	58	16.67	0	-28.96	6	7.04	1520.18	1527.22
Aug	51	-14.72	0	-66.62	6	-30.62	1241.85	1211.23
Sep	44	-57.59	0	-86.83	3	-68.83	639.32	570.49
Oct	42	-35.26	0	-72.22	4	-48.22	886.2	837.98
Nov	47	-21.83	0	-69.85	6	-33.85	1261.95	1228.1
Dec	34	-70.42	0	-109.23	4	-85.23	710.6	625.37

Table 7 Results for the three-stage heuristic.

For all twelve months, it took the three-stage heuristic in total 1 hour, 42 minutes, and 39 seconds to complete stage 1, 13 hours, 18 minutes, 47 seconds to complete stage two, and 2 hours 35 minutes, 7 seconds to complete stage 3. On average, it takes the three-stage heuristic about 8.6 minutes per month for stage 1, and 12.9 minutes for stage 3, depending on the volume of courses being run. Stage 2 will run until some stopping criterion is met, which we defined as being 2 hours for each timetabled month. There is a slight overrun in time for stage 2 of about 6.6 minutes on average per timetabled month due to the time it takes to complete an iteration.

Both the existing software system and the three-stage heuristic were able to satisfy all demand for each month (Z_1). Compared with the existing system, the three-stage heuristic was able to produce a significantly improved solution with respect to the room rental goal (Z_2), and an improved solution with respect to the room swap goal (Z_3). It should be noted, however, that the existing system does not give the same priority to the room rental objective as is given in our model, as this was not a priority for Ausgrid when the original system was under development. Stage 2 of the three-stage heuristic was able to improve on the initial (stage 1) timetable by, on average, 13.15 units.

This improvement comes at a significant time difference, with the three-stage heuristic taking much longer to produce a solution than the existing software. It should be noted, however, that the time taken by this system is still significantly less than a human constructing a timetable, manually, which generally takes about two uninterrupted working days for a one-month timetable and trainer roster. For longer-term strategic planning purposes, these solution times are acceptable, however for day-to-day timetabling, these solution times are generally not practically acceptable and it is a matter of ongoing research to further improve the process.

8 Conclusion(s)

In this paper we studied an academic timetabling and rostering problem involving periodic retraining of large numbers of employees at an Australian electricity distributor. A three-stage heuristic framework has been presented which consists of an initial timetable generation stage, an iterative timetable improvement stage, and a trainer rostering stage. Integer linear programming (ILP) models were developed for each stage, which can deal with all the practical requirements flexibly. Different algorithms are designed to achieve the balance of solution quality and computation time. The preliminary computational results show that this approach can generate solutions with lower trainer movement and swap costs, lower room swap costs, and increased revenue from room rentals compared with the existing software system in this organisation. More work needs to be done to further reduce the computation time.

References

1. Akkoyunlu, E.: A linear algorithm for computing the optimum university timetable. The Computer Journal **16**(4), 347–350 (1973)
2. Birbas, T., Daskalaki, S., Housos, E.: Course and teacher scheduling in hellenic high schools. In: 4th Balkan Conference on Operational Research, Thessaloniki, Greece (1997)
3. Bölte, A., Thonemann, U.W.: Optimizing simulated annealing schedules with genetic programming. European Journal of Operational Research **92**(2), 402–416 (1996)
4. Burke, E., Elliman, D., Weare, R.: A university timetabling system based on graph colouring and constraint manipulation. Journal of Research on Computing in Education **27**, 1–1 (1994)
5. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research **140**(2), 266–280 (2002)
6. Carter, M.W., Tovey, C.A.: When is the classroom assignment problem hard? Operations Research **40**(1-Supplement-1), S28–S39 (1992)
7. Dimopoulou, M., Miliotis, P.: Implementation of a university course and examination timetabling system. European Journal of Operational Research **130**(1), 202–213 (2001)
8. Fischetti, M., Widmayer, P.: Towards solving very large scale train timetabling problems by lagrangian relaxation. In: 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08), vol. 9. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2008)
9. Gunawan, A., Ng, K., Poh, K.: A hybrid algorithm for the university course timetabling problem. In: Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (2008)
10. IBM: Ibm cplex optimizer (2014). URL http://www.ibm.com/software/commerce/optimization/cplex-optimizer/
11. Karp, R.M.: Reducibility among combinatorial problems. Springer (1972)
12. Lawrie, N.L.: An integer linear programming model of a school timetabling problem. The Computer Journal **12**(4), 307–316 (1969)

13. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum **30**(1), 167–190 (2008)
14. Marx, D.: Graph coloring problems and their applications in scheduling. In: in Proc. John von Neumann PhD Students Conference. Citeseer (2004)
15. Mehta, N.K.: The application of a graph coloring method to an examination scheduling problem. Interfaces **11**(5), 57–65 (1981)
16. Neufeld, G., Tartar, J.: Graph coloring conditions for the existence of solutions to the timetable problem. Communications of the ACM **17**(8), 450–453 (1974)
17. Papoutsis, K., Valouxis, C., Housos, E.: A column generation approach for the timetabling problem of greek high schools. Journal of the Operational Research Society **54**(3), 230–238 (2003)
18. Pillay, N.: A survey of school timetabling research. Annals of Operations Research pp. 1–33 (2013)
19. Qualizza, A., Serafini, P.: A column generation scheme for faculty timetabling. In: Practice and Theory of Automated Timetabling V, pp. 161–173. Springer (2005)
20. Schimmelpfeng, K., Helber, S.: Application of a real-world university-course timetabling model solved by integer programming. OR Spectrum **29**(4), 783–803 (2007)
21. Ülker, Ö., Özcan, E., Korkmaz, E.E.: Linear linkage encoding in grouping problems: applications on graph coloring and timetabling. In: Practice and Theory of Automated Timetabling VI, pp. 347–363. Springer (2007)
22. Yao, X.: A new simulated annealing algorithm. International Journal of Computer Mathematics **56**(3-4), 161–168 (1995)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A model and fast heuristics for the multiple depot bus rescheduling problem

Balázs Dávid · Miklós Krész

Abstract The daily schedule of a transportation company is often disrupted by unforseen events. As a result, a new schedule has to be produced as soon as possible to restore the order. In this paper, we consider the bus rescheduling problem for solving such a scenario. We present a mathematical model for the problem, and also introduce fast solution methods that give efficient solutions with short running time. These methods are tested on different random and real-life instances, and their results are compared to that of the optimal solution of the mathematical model.

Keywords Disruption management · Vehicle scheduling · Heuristic

1 Introduction

Public transportation companies create their daily schedules in advance for a longer planning period. This planning process is carried out by a complex system, an example for which can be seen in [1]. However, several events (the most common of which are vehicle breakdown and lateness) can render the pre-planned schedules infeasible. In most of these cases, companies want to restore the order as soon as possible, and need a new feasbile schedule where all of their tasks are carried out in a feasible manner once again.

Such an unforseen event is called a disruption, and is defined by Clausen et al. in [7] as "an event or a series of events that renders the planned schedules for aircraft, crew, etc. infeasible". Although the above definition was given in a technical paper about airline disruptions, this could be generalized for any

B. Dávid
University of Szeged, Gyula Juhász Faculty of Education
E-mail: davidb@jgypk.u-szeged.hu

M. Krész
University of Szeged, Gyula Juhász Faculty of Education
E-mail: kresz@jgypk.u-szeged.hu

means of transportation. In our paper, we will be dealing specifically with the rescheduling of disruptions arising in public bus transportation.

To our knowledge, there are only a handful of papers that deal with disruptions in bus transportation [12–14]. In practice, the problem is solved by the operators of the company who use their past experience for constructing the new schedule. However, the problem size is large, and many feasible solutions exist for a disruption. Not even the most skillful operator can see all the good possible solutions.

Addressing such a disruption and restoring the order in public transportation should be done as quickly as possible. The reason for this is the fact that if a disruption remains unresolved, it could result in several other disruptions. This is why we need to introduce fast methods for the problem that also give good quality solutions. Instead of solving the problem to optimality, these algorithms should give a number of good quality feasible solutions to the operator as suggestions. Using these suggestions, the operator can make the final decision on how to reschedule the disrupted trips.

In the following sections, we define the bus rescheduling problem (BRP), and give a mathematical model for it. As the size of this mathematical model is large even for smaller instances, we propose two fast algorithms to solve the BRP: a recursive heuristic and a local search method. We analyze the solutions of these algorithms, and compare their results on random instances to the optimal solution of our mathematical model.

2 Disruption management and rescheduling

The structure of the bus rescheduling problem is similar to that of the vehicle scheduling problem (VSP). We are given a set V of vehicles and set T of service trips. Every trip has a departure and arrival time, a starting and ending location, and a set of vehicles that are able to serve the trip. A (t, t') pair of trips are compatible if a vehicle can service both trips with respect to the running time and distance between the arrival location of t and the departure location of t' (such a journey is called a deadhead trip).

The VSP assigns the trips of the given timetable to the vehicles, satisfying certain conditions:

- Every trip in $t \in T$ must be executed exactly once.
- For every vehicle $v \in V$, the trips assigned to v must be compatible with each other.
- The cost of the assignment must be minimal. The cost of the VSP is usually given by two components: a cost proportional to the distance travelled in the solution, and a cost given by the number of buses used.

Furthermore, vehicles can be classified into depots depending on two characteristics: the type of the vehicle (eg. solo or articulated bus), and its starting location at the beginning of the day. In this case, every trip is also assigned a depot-compatibility vector which corresponds to the depots that can feasibly serve it. Moreover, the arising costs can be different from depot to depot.

If the problem has only 1 depot, it is called a single depot vehicle scheduling problem (SDVSP), and can be solved in polynomial time. A formulation for the SDVSP can be seen in [3]. If the number of depots is at least 2, we get a multiple depot vehicle scheduling problem (MDVSP). The MDVSP was introduced by Bodin et al. in [4], and proven to be NP-hard by Bertossi et al. [2]. An overview of different VSP models can be found in [5].

2.1 The bus rescheduling problem

The bus rescheduling problem considers a given daily bus schedule. The schedule consits of several vehicle duties, each such duty corresponding to a unique vehicle. A vehicle duty is a sequence of compatible trips, where compatible means that they can legally be executed one after the other. When a disruption happens in the daily schedule of a company at time s, the current schedule becomes infeasible, and as a result, a number of trips can no longer be executed by their original vehicles.

As an input for the problem, we need to consider the disrupted daily schedule DS, which contains all the vehicles and trips that are still executed according to the original schedule. We also have the set DT of disrupted trips, which contains the trips that cannot be served due to the disruption. The set DT can contain timetabled trips that no longer have their assigned vehicle as a result of the disruption, and it can also contain newly introduced trips that were not part of the original daily schedule. Let $T' \subseteq T$ be the subset of trips that start later than s. The aim once again is to give a feasible solution to the problem by executing the trips $T' \cup DT$ and minimizing the arising costs.

The cost of the problem depends on the restrictions that are taken into consideration. We introduced the following cost components:

- **Operational costs**: This cost is proportional to the distance covered by the given vehicle. If a new vehicle is introduced, it also has a fixed daily cost. This cost can be scaled with a penalty parameter for new vehicles if we want to primarily use our current vehicles in service.
- **Deviation from the original schedule:** If a trip is carried out by a different vehicle in the solution of the BRP than in the original schedule, we introduce an extra penalty. If we take the physical needs of the drivers into consideration, the solution should be as close to the original schedule as possible, and as a result, this cost should be high.
- **Lateness of the trips:** It is possible for a trip to be shifted in time, thus introducing lateness to its starting time. Each minute of lateness should be penalized.
- **Trip cancellation:** We can allow trips to be cancelled, but its cost must always be higher than the actual cost of the trip and its possible deadhead trips, and it must also be higher than the cost introduced to the deviation from the original schedule.

Figure 1 models a typical situation that can arise in the daily practice of a transportation company. A schedule is disrupted, and 2 trips (coloured blue) have to be rescheduled using 3 remaining vehicle duties (part *a)*). We give 2 different solutions: in part *b)*, a task is moved from the first duty to the third one before the insertion of the disrupted trips, while part *c)* gives a solution with trivial insertion into the duties. Note, that different solutions are also possible, for example one where we introduce lateness to one of the trips.

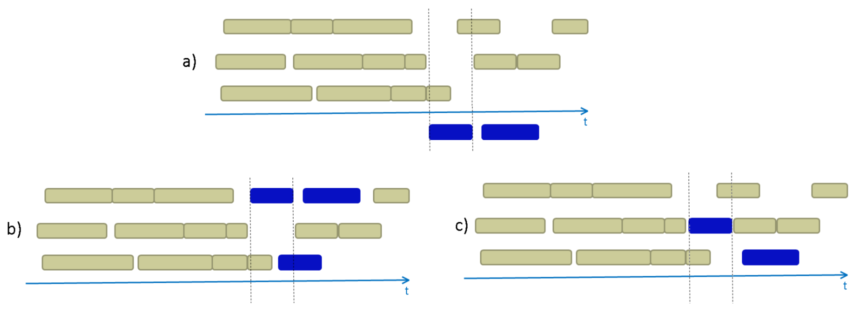


Fig. 1 A typical example for the BRP: there are two disrupted trips in example a) that have to be inserted into the given duties, b) and c) represent possible solutions

2.2 Related work

Literature usually addresses recoveries from disruptions under the field of disruption management. Depending on its effect, there are two main types of disruptions:

– A short term disruption only affects the schedule of the given day, and can be addressed quickly.
– A long term disruption has a more lasting effect, and can affect several days of the companies long term plan.

In this paper, we only deal with short term disruptions, where a few trips of the original daily schedule become infeasible, and must be rescheduled. This is the typical case when a vehicle that is late with regards to its schedule, and would only be able to start some of its service trips with significant lateness, or a vehicle that can not carry out some of its trips due to technical difficulties.

The first research into disruption management was carried out in the airline industry. Clausen et al. give a thorough overview of this field in [6, 7]. The underlying network is somewhat similar to the problem of the BRP. However, the methods used for airline disruptions are computationally intensive, and have a long running time on the significantly larger bus transportation problems. This size difference comes from the smaller instances sizes of the airline industry, and the limited deadheading possibilites of the aircraft.

Disruption management in railway transportation is covered in [10]. These problems have a different structure, which mainly comes from the fixed railway network, and the capacity limit of the tracks. Because of this, railway disruptions are handled in a different manner than in the airline industry.

To our knowledge, bus rescheduling as we defined in the above section was only considered by Li et al. In [13], they propose a quasi-assignment model and an auction algorithm for the problem, while in [14] they introduce a network flow model for the single depot BRP, which they solve using a lagrangean method.

2.3 Mathematical model

In this section we give a multi-commodity network flow model for the BRP described in subsection 2.1, which is similar to the network flow model reported by Li et al. in [14]. While Li et. al presented a model for the single depot BRP that allows trip cancellations, we propose a model for the multiple-depot BRP with trip cancellations and lateness.

Our input is the schedule of the company for a given day, which is disrupted at time point s. Let D be the set of depots, V be the set of vehicles currently in service, and $P = D \cup V$.

Let T' be the set of non-disrupted service trips of the given day that depart after time s, and thus still need to be serviced, and let set DT contain the disrupted trips. Let the set $T = T' \cup DT$ represent all the trips of our problem. Every trip $t \in T$ has a departure time $dt(t)$, arrival time $at(t)$, starting location $sl(t)$ and ending location $el(t)$. The set of depots and vehicles that can execute a trip t is denoted by $g(t)$. Let $T_d \subseteq T$ be the set of trips that can be executed from depot d, and $T_v \subseteq T$ the set of trips that can be carried out by vehicle v.

For every depot $d \in D$, we introduce notations $sl(d)$ and $el(d)$. A depot d is represented by $sl(d)$ when we consider it as the starting location of a vehicle, while we use $el(d)$ when it gives the ending location of the vehicle. Similarly for every vehicle $v \in V$ currently in service we define a starting location $sl(v)$ and ending location $el(v)$. For a vehicle v, $sl(v)$ corresponds to the current geographical location of v at the time of the disruption, and $el(v)$ is the geographical location of its depot. The set of nodes of our network will be the following:

$$N = \{dt(t) \cup at(t) \cup sl(d) \cup el(d) \cup sl(v) \cup el(v) | t \in T, d \in D, v \in V\}.$$

Using the nodes above, we define the different edges of the network. Let

$$J^d = \{(dt(t), at(t)) | t \in T_d\}$$

be the set of trips that can be served by depot d, and let

$$J^v = \{(dt(t), at(t)) | t \in T_v\}$$

be the set of trips that can be executed by vehicle v. Let

$$K^d = \{(at(t), dt(t')) | t, t' \in T_d \textit{ are compatible}\}$$

be the possible deadhead trips of a depot d and

$$K^v = \{(at(t), dt(t'))|t, t' \in T_v \ \textit{are compatible}\}$$

be the possible deadhead trips of a vehicle v.
Let

$$L^d = \{(sl(d), dt(t)), (at(t), el(d))|t \in T_d\}$$

be all the pull-in and pull-out edges of depot d, and let

$$L^v = \{(sl(d), dt(t)), (at(t), el(d))|t \in T_v\}$$

be the pull-in and pull-out edges of vehicle v. The above sets together with circulation edges for every depot and vehicle give us the set of edges of our network:

$$E = \{J^d \cup J^v \cup K^d \cup K^v \cup L^d \cup L^v \cup \{(el(d), sl(d))\} \cup \{(el(v), sl(v))\} \ \textit{for every} \ d \in D, \ v \in V \ \}.$$

With the nodes and edges introduced above, a solution of the vehicle rescheduling problem can be determined by using the network (N, E). We define an integer vector x for every edge of the network. Every $p \in P$ defines a component belonging to edge e, and is denoted by x_e^p. We also introduce a variable w_t for every $t \in T$, which allows the cancellation of t.

The difference between the original schedule and the resulting schedule should also be modeled. For every vehicle v, and trip-edge $e \in J^v$, we introduce a constant q_e. The value of q_e is 0, if the trip corresponding to edge e is carried out by the same vehicle v as in the original schedule, and 1 otherwise. The cost of a trip will depend on this constant, because αq_e will be added to the each such edge, where α is the penalty for deviation from the original schedule.

To allow lateness for trips, we introduce variable z_t, which gives a new departure time for every trip t. This value includes the added lateness, if any. For the trips to remain compatible, a constraint has to be added that examines trip compatibilities with respect to z_t:

$$(z_t + length(t) + deadhead_{t,t'} - z_t') \sum_{p \in P} x^p_{a(t),d(t')} \leq 0, \forall (t, t') \in E, \tag{1}$$

where $length(t)$ gives the running time of service trip t, and $deadhead_{t,t'}$ represents the running time of the deadhead trip between $al(t)$ and $dt(t')$. Constraint (1) is not a linear equation, but such a constraint can be rewritten with the introduction of a large constant M, as seen in [9].

The IP model of the problem can be formalized in the following way:

$$\sum_{p \in g(t)} x^p_{dt(t),at(t)} + w_t = 1, \forall t \in T \tag{2}$$

$$\sum_{e:(sl(d),dt(t)) \in K^d} x_e^d \leq k(d), \forall d \in D \tag{3}$$

$$\sum_{e:(sl(v),dt(t))\in K^v} x_e^v = 1, \forall v \in V \tag{4}$$

$$\sum_{e\in n^+} x_e^p - \sum_{e\in n^-} x_e^p = 0, \forall p \in P, \forall n \in N \tag{5}$$

$$z_t + length(t) + deadhead_{t,t'} - z_t' \leq \sum_{p\in P} 1 - x^p_{a(t),d(t')} M, \forall (t,t') \in E \tag{6}$$

$$z_t \geq start(t), \forall t \in T \tag{7}$$

$$z_t \leq start(t) + L, \forall t \in T \tag{8}$$

$$x_e^p, w_t \in \{0,1\}, \forall e \in E, p \in P, t \in T \tag{9}$$

Due to constraint (2), every trip is either executed exactly once, or cancelled. Constraint (3) gives maximum capacities for the depots of the problem, while vehicles in service are always given duties according to constraint (4). Constraint (5) ensures flow conservation. Constraint (6) is the linear reformulation of constraint (1). Constraints (7) and (8) limit the values of the trip starting times. A trip $t \in T$ will always depart in the $[start(t), start(t) + L]$ time window, where $start(t)$ is the departure time of t and L is the maximum allowed lateness. To solve the problem to optimality, we need to minimize

$$\sum_{e\in E}\sum_{p\in P} c_e^p x_e^p + \sum_{t\in T} \beta(z_t - start(t)) + \sum_{t\in T} \gamma w_t, \tag{10}$$

where β and γ are penalty parameters for lateness and cancelling trips respectively. c_e^p gives the corresponding operational cost of edge x_e^p, along with the possible added penalty of deviation from the original schedule given by α.

2.4 Importance of the model

The size of the above model grows quickly with an increase in the number of compatible trips. As each commodity layer of the model contains the possible connections of the given depot or vehicle, its size will increase significantly with every added commodity. We have to represent every bus currently in service as a new commodity in the model, which results in a significantly high number of vehicle layers. For example, the middle-sized city of Szeged, Hungary has about 2700 trips on a regular workday, which is executed by 108 vehicles (belonging to 4 vehicle types). This would mean a network with over 110 commodities, which is taxing to solve both in memory and running time.

In a real life application, solutions for disruptions are needed in real time, because the resolution of a disruption does not end with the solution of the problem. After solving the BRP, operators still need to communicate with bus drivers about the recent changes, which also takes time.

However, the mathematical model is important because it allows us to compare our results to the optimal solution for the problem. This way it can provide us with a quality control for any heuristic method that we use for the fast solution of the BRP.

3 Applying fast solution heuristics

As we mentioned in subsection 2.4, the size of the model will grow quickly with the increase of the problem size. The number of commodities in the network is given by the number of depots and vehicles, and this will be dominated by vehicle commodities (except for small test-instances).

Exact solution of such a big problem would take a long running time, which is not acceptable for a real life application of the method. Recovering from a disruption needs to be done as quickly as possible, thus fast and efficient solution heuristics are needed.

It might well be that the costwise optimal solution is not be the best one regarding operations planning. However, a solution given by an algorithm can help the operator decide faster about how to resolve the disruption. Integrating fast algorithms into a decision support system that gives multiple suggestions for the solution can speed up the decision of the operators.

In the following sections, we present fast solution methods that provide multiple good quality solutions in a short running time. These methods can easily be integrated into a decision support system for disruption management in public transportation, which recommends possible solutions for the operators depending on their parameter settings. We will be dealing with such a system in one of our future papers.

Li et al. also presented a prototype decision support system in [12] for the single depot BRP. In this system, they were only dealing with the optimal solution for the problem that the operators could change on an interactive interface.

3.1 A recursive search algorithm

One of the algorithms we propose is a recursive search heuristic for the problem. Recursive search seems an ideal method because of the expectations described above. This algorithm is able to find multiple solutions with a short running time. Our method can be seen in Algorithm 1.

The input of the algorithm is the following:

- **vDuties:** The disrupted daily schedule. It includes all available vehicles and the duties assigned to those vehicles that were not disrupted.
- **dTrips:** The set of disrupted trips, which have no assigned vehicle duties.
- **depth:** A non-negative integer parameter that limits the depth of the search tree. Every time trips are removed from a schedule, the value of this parameter is decreased.

Algorithm 1 Recursive search for bus rescheduling.

```
procedure RecSearch(vDuties, dTrips, depth)
    if depth = 0 then
        return 0
    end if
    for i = 1 to Size(dTrips) do
        Trip = dTrips[i]
        for j = 1 to Size(vDuties) do
            nDuties = vDuties
            nTrips = dTrips
            Duty = vDuties[j]
            if Trip and duty are not compatible then
                continue
            end if
            if Trip overlaps with trips in Duty then
                if Possible solution with lateness then
                    Duty' = Insert Trip into Duty with introducing lateness
                    nDuties' = nDuties with Duty' inserted into nDuties[j]
                    nTrips' = nTrips without Trip
                    if Size(nTrips') = 0 then
                        Add(Solutions, nDuties')
                    else
                        RecSearch(nDuties', nTrips', depth)
                    end if
                end if
                tRemoved = overlapping trips from duty
            end if
            Insert Trip into Duty
            Remove Trip from nTrips
            nDuties[j] = Duty
            if Size(nTrips) = 0 then
                Add(Solutions, nDuties)
            else if Size(tRemoved) > 0 then
                Add(nTrips, tRemoved)
                RecSearch(nDuties, nTrips, depth-1)
            else
                RecSearch(nDuties, nTrips, depth)
            end if
        end for
    end for
    return Best solution in Solutions
end procedure
```

The input for the heuristic is the set of feasible vehicle duties, and the set of disrupted trips. Every function call chooses the disrupted trip dt with the earliest departure time, and tries to fit it into every compatible vehicle duty vd. There are three possibilities for every $dt - vd$ pair:

- One or more trips have to be removed from vd. The removed trips are flagged as temporary disrupted trips.
- Trip dt can be inserted into vd, but lateness has to be introduced for some of the trips of vd.
- Trip dt can be inserted into vd without additional modifications.

If the set of disrupted trips is empty after a modification, and there are no temporary disrupted trips, the heuristic has found a feasible solution, which is saved. Otherwise, the recursive function is called with new parameters:

– **vDuties':** The original $vDuties$ is updated with the modified duty.
– **dTrips':** The temporary disrupted trips are inserted into $dTrips$, while dt is removed.
– **depth':** If the size of $dTrips'$ is smaller than the size of $dTrips$, $depth' = depth$. Otherwise, $depth' = depth - 1$.

The algorithm will explore the solution space determined by the trips and the schedules of the problem, examining every possible solution found during its runtime. The depth of this search tree is limited by the parameter $depth$. Further limitations can be introduced into the method to exclude visiting similar configurations multiple times.

These limitations (especially the parameter for the depth) help to keep the running time of the algorithm from exploding. Introduction of this parameter was also based on a practical observation: each level of the recursive search tree corresponds to a vehicle whose original duty is modified. Companies want to keep the number of modified vehicle duties low. Because of the way $depth$ is decreased, its initial value also defines the maximum number of vehicle duties from which the algorithm can remove trips. As we mentioned in Subsection 2.1, altering the original schedule of a driver should have a high cost, so it is unlikely that the optimal schedule will be cut from the search tree by this parameter.

The algorithm terminates after it has traversed the above defined search tree. If it has found at least 1 solution, then the one with the lowest cost is returned as a result.

3.2 A local search algorithm

The other proposed algorithm is a local search method for finding a feasible solution for the BRP. A brief outline of the algorithm can be seen in Algorithm 2.

The input of the algorithm is the following:

– **vDuties:** The disrupted daily schedule. It includes all available vehicles and the duties assigned to those vehicles that were not disrupted.
– **dTrips:** The set of disrupted trips. These are currently not executed by vehicles, and have to be assigned to vehicle duties.
– **tRange:** Gives a time window in which the events are considered. The time window begins at the start time of the disruption, and ends after the ending time of the last disrupted trip.

The initial candidate solution of the algorithm is constructed from the original vehicle schedule. A new vehicle duty is added to the schedule, that contains all the disrupted trips, increasingly ordered by their departure time.

Algorithm 2 Local search for bus rescheduling.

```
 1: procedure LocSearch(vDuties, dTrips, tRange)
 2:     Build infeasible duty dt from dTrips
 3:     Label dt temporary
 4:     Add(dt to vDuties)
 5:     tabuList = list of forbidden transformations
 6:     while notEmpty(dt) & !(terminatingConditions) do
 7:         tmpSchedules = empty container for vehicle schedules
 8:         for i = 1 to Size(vDuties) do
 9:             for j = i+1 to Size(vDuties) do
10:                 for each neighborhood transformation $t$ do
11:                     if t(vDuties[i], vDuties[j]) is not forbidden then
12:                         newSchedule = apply $t$ of vDuties
13:                         Add(newSchedule, tmpSchedules)
14:                     end if
15:                 end for
16:             end for
17:         end for
18:         bSchedule = best schedule from tmpSchedules
19:         tS = transformation that can reverse bSchedule
20:         vDuties = bSchedule
21:         Add(tabuList, tS)
22:     end while
23:     return vDuties
24: end procedure
```

If there is more than one disrupted trip, this new duty is more than likely infeasible, which will make our initial solution also infeasible. This new duty is labelled as a *temporary duty.*

In each iteration the algorithm will examine all (i, j) pairs of the duties. It checks the trips of the duties that are in the given *tRange* time window, and examines the following two neighborhood transformations:

– **1-move:** Moves a trip from duty i to duty j. This transformation is not carried out, if j is a temporary schedule.
– **1-change:** Exchanges a trip from duty i with the corresponding trip(s) from duty j. This transformation is not carried out, if any of the duties is temporary.

All feasible neighbors given by the above transformations are assigned a cost. This cost is computed from the operational cost of the duties and the penalties introduced in subsection 2.1. If the transformation moves a trip from a temporary schedule to another schedule, a high negative penalty is added to decrease the cost, which will make it more likely to be chosen in an early iteration of the local search.

The local search algorithm chooses the neighbor solution with the lowest cost as its new candidate. If any trip t was removed from a duty D in the process, the (t, D) pair is saved on a tabu list. For every (t, D) pair on the tabu list, trip t cannot be moved to duty D with any of the transformations.

The algorithm terminates when at least one of the terminating conditions is met. We use the following terminating conditions:

– Limit for the running time.
– If the difference in quality of the consecutive candidates is always below a given gap for a fixed amount of iterations.

If the algorithm has found at least 1 feasible solution, then the one with the lowest cost is returned by default. The number of solutions returned can be set higher using a parameter, if the user wants to ask for more suggestions.

4 Test results

In this section we provide the test results of the heuristics presented in section 3. To analyze the quality of our results, we will compare them to the optimal solution of the BRP model given in subsection 2.3.

As it was mentioned earlier, the size of the mathematical model can grow quickly with the increase of the instance size. The model we presented in 2.3 represents all possible connection between pairs of trips with a unique deadhead edge for every commodity. Because of the high number of deadhead connections, Kliewer et al. introduced the time-space network in [11] to solve the MDVSP. This model aggregates deadhead edges, resulting in a much smaller problem that is easier to solve. The number of deadhead edges of the BRP model in subsection 2.3 can be decreased in the same way.

In our test instances, we used a time-space network equivalent of our BRP model. We generated our daily schedules by solving a time-space network model on random instances given by a method described in [8]. The disruption time was set to 0, which means that we considered the whole daily schedule in every case. We modelled the scenario when a new trip is introduced to the daily schedule at the beginning of the day. This new trip is our disrupted trip, which is generated as a single short trip by the same random method referenced above.

We tested the methods on different instances with 12, 100, 500 and 800 trips in their original schedule. Several different test cases were generated for each instance. Table 1 shows the average results of 10 randomly generated cases for every instance.

Table 1 Test results of the heuristic methods

Instance	Trips	Depots	Opt.(s)	Rec.(s)	Gap (Rec.)	Loc.(s)	Gap(Loc.)
R1	13	2	1.03	0.02	0 %	0.001	0%
R2	101	4	1.12	0.05	0 %	0.004	0%
R3	501	4	132	0.08	0 %	0.01	0%
R4	801	4	801	0.08	0.04 %	0.05	0%

The number of trips and depots of the BRP can be seen in columns 2 and 3. The running time of the solution of the exact model is given in column 4. The running time of the heuristics and their gap from the optimal solution are represented in columns 5-6 and 7-8 respectively. All tests were carried out on a PC with and Intel Core i5 2.80GHz CPU and 4 GB RAM. The IP model was solved using the COIN-OR Symphony solver.

We could not solve the IP for instances with higher number of trips, because the model itself was too big to be contained in memory. The heuristics provided results for significantly larger input as well. We tested the two heuristic methods on both random and real-life instances, and both of them returned multiple solution suggestions. The biggest real-life instance we used contained 2674 trips, while the biggest random instance had 3500 trips.

The solution time of the heuristics remained fast, and took at most around 15 seconds. We also hand-tailored some of the bigger test instances, in which we knew the best solution from the point of view of operational planning (e.g. trivial insertion of a trip to a duty, or a certain trip has to be moved/delayed to insert the disrupted trip). The algorithms found the desired solutions in every case.

The results of the heuristic algorithms are promising, as they give multiple solutions even for larger instances in a short time, while the number of modified schedules and moved trips stay low. Their good speed and solution quality, and the multiple given solutions make them suitable for a decision support system described in the previous sections.

5 Conclusions and future work

In this paper, we considered the multiple depot BRP, which deals with rescheduling the disrupted daily schedule of a transportation company. This problem is important, as disruptions happen in the daily schedules of every company, and the order of transportation should be restored as soon as possible. We described the restrictions of the problem, and defined a mathematical model based on the arising needs.

Such a problem requires a real-time solution, because the results must be processed by operators and communicated to the bus drivers, which also takes time. As the size of the model is too big to be solved in such short time, we proposed two fast heuristic algorithms to produce results in a couple of seconds. Our tests on randomly generated instances showed that the heuristics give a solution that is close to the optimum. While we could not measure the quality of the algorithms on bigger real-life instances, the running time still remained fast, and both methods gave the expected results for artifical disruption scenarios for these inputs.

Because of their ability to produce multiple good quality solutions in a short time, these algorithms seem suitable for a decision support system that helps the operators of a transportation company in the rescheduling process by giving them possible solution suggestions for the problem. However, there are still questions for future research.

The size of the mathematical model is too big to be contained in memory for bigger instances. To get exact solutions for real-life problems, we can use decomposition methods (e.g. column generation), or heuristic size reduction of the model. Both approaches should be investigated in future works.

Acknowledgements This work was partially supported by the European Union and co-funded by the European Social Fund through project HPC (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0010).

References

1. Békési, J., Brodnik, A., Krész, M., Pas, D.: An integrated framework for bus logistics management: Case studies. Logistik Management **5**(1), 389–411 (2009)
2. Bertossi, A., Carraresi, P., Gallo, G.: On some matching problems arising in vehicle scheduling models. Networks **17**(1), 271–281 (1987)
3. Bodin, L., Golden, B.: Classification in vehicle routing and scheduling. Networks **11**(1), 97–108 (1981)
4. Bodin, L., Golden, B., Assad, A., Ball, M.: Routing and scheduling of vehicles and crews: The state of the art. Computers and Operations Research **10**(1), 63–212 (1983)
5. Bunte, S., Kliewer, N.: An overview on vehicle scheduling models. Journal of Public Transport **1**(4), 299–317 (2009)
6. Clausen, J., Larsen, A., J. Larsen, J., Rezanova, N.J.: Disruption management in the airline industry-concepts, models and methods. Computers & Operations Research **37**(5), 809–821 (2010)
7. Clausen, J., Larsen, A., Larsen, J.: Disruption management in the airline industry - concepts, models and methods. Tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark, DTU (2005)
8. Dávid, B., Krész, M.: Application oriented variable fixing methods for the multiple depot vehicle scheduling problem. Acta Cybernetica **21**(1), 53–73 (2013)
9. Desrochers, M., Lenstra, J., Savelsbergh, M., Soumis, F.: Vehicle routing with time windows: Optimization and approximation. Vehicle routing: Methods and studies **16**, 65–84 (1988)
10. Jespersen-Groth, J., Potthoff, D., Clausen, J., Huisman, D., Kroon, L.G., Maróti, G., , Nielsen, M.N.: Disruption management in passenger railway transportation. Tech. rep., Erasmus University Rotterdam (2007)
11. Kliewer, N., Mellouli, T., Suhl, L.: A time-space network based exact optimization model for multi-depot bus scheduling. European Journal of Operational Research **175**(3), 1616–1627 (2006)
12. Li, J.Q., Borenstein, D., Mirchandani, P.B.: A decision support system for the single-depot vehicle rescheduling problem. Computers and Operations Research **34**(4), 1008–1032 (2007)
13. Li, J.Q., Mirchandani, P.B., Borenstein, D.: The vehicle rescheduling problem: Model and algorithms. Networks **50**(3), 211–229 (2007)
14. Li, J.Q., Mirchandani, P.B., Borenstein, D.: A lagrangan heuristic for the real-time vehicle rescheduling problem. Transportation Research Part E: Logistics and Transportation Review **45**(3), 419–433 (2009)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Solving High School Timetabling with Satisfiability Modulo Theories

Emir Demirović · Nysret Musliu

Abstract High School Timetabling (HSTT) is a well known and wide spread problem. The problem consists of coordinating resources (e.g. teachers, rooms), time slots and events (e.g. lectures) with respect to various constraints. Unfortunately, HSTT is hard to solve and just finding a feasible solution for simple variants of HSTT has been proven to be NP-complete. In addition, timetabling requirements vary from country to country and because of this many variations of HSTT exist. Recently, researchers have proposed a general HSTT problem formulation in an attempt to standardize the problem from different countries and school systems.

In this paper, for the first time we provide a new detailed modeling of the general HSTT as a Satisfiability Modulo Theory (SMT) problem in the bit vector form. In addition, we present preliminary experimental results and compare to the winner of the Third International Timetabling Competition 2011 (ITC), using both artificial and real-world instances, all of which were taken from ITC 2011 benchmark repository. Our current approach provides feasible solutions for some examples, which in some cases could not have been obtained with the competition winner algorithm within 24 hours.

Vienna University of Technology
Database and Artificial Intelliegence Group
E-mail: $\{musliu \lor demirovic\}$@dbai.tuwien.ac.at

Keywords SMT · High School Timetabling · Modeling

1 Introduction

In this paper, we describe a modeling of the high school timetabling problem (HSTT) as a Satisfiability Modulo Theory (SMT) problem. By doing so, we were able to find feasible solutions to some problem instances, which were proposed by the International Timetabling Competition 2011 [13], which in some cases could not have been obtained using the winning algorithm of the competition in 24 hours, GOAL. In two smaller instances, optimization could also be performed, rather than just finding a feasible solution, but optimization is difficult for our method at its current state.

The problem of timetabling is to coordinate resources (e.g. rooms, teachers, students) with time slots in order to fulfill certain goals or events (e.g. lectures).

Timetabling is encountered in a number of different domains. Every educational institution, airport, public transport system, etc requires some form of timetabling. The difference between a good and a bad timetable can be significant, but constructing timetables by hand can be time consuming, very difficult, error prone and in some cases impossible. Therefore, developing high quality algorithms which would automatically do so is of great importance. Note that there are many different timetabling problems and algorithms for one type of problem (e.g. HSTT) might not directly be suitable for another problem (e.g. University Timetabling), because of their different requirements. In this work, we focus on HSTT. Respecting constraints is very important, as timetables directly contribute to the quality of the educational system, satisfaction of students and staff and other matters. Every timetabling decision affects hundreds of students and teachers for prolonged amounts of time, since each timetable is usually used for at least a semester.

Unfortunately, High School Timetabling is hard to solve and just finding a feasible solution of simple variants of High School Timetabling has been proven to be NP-complete [7].

Apart from the fact that problems that need to be solved can be very large and have many different constraints, high school timetabling requirements vary from country to country and because of this many variations of the timetabling problem exist. Because of this, it was unclear what the state of the art was, as comparing algorithms was difficult.

Recently have researchers proposed a general high school timetabling problem formulation [14] in an attempt to standardize the problem from different countries and school systems and this formulation has been endorsed by the Third International Timetabling Competition 2011 (ITC 2011) [13] [14]. This was a significant contribution, as now algorithms can be compared on standardized instances, that were proposed from different researchers [12].

The winner of the competition was the group GOAL, followed by Lectio and HySST. All of the algorithms were based on heuristics. In GOAL, an initial solution is generated, which is further improved by using Simulated Annealing and Iterated Local Search, using seven different neighborhoods [8]. Lectio uses an Adaptive Large Neighborhood Search [16], while HySST uses a Hyper-Heuristic Search [9]. Recently, [17] used Integer Programming (IP) in a Large Neighborhood Search algorithm and [15] introduced a two phase IP algorithm for a different timetabling problem, but have managed to adjust the method for a number of high school timetabling instances.

All of the best algorithms on the competition were heuristic algorithms and this is why introducing a new exact method (our approach) is important. Some advantages are being able to provide proofs of optimality or infeasibility, calculate lower bounds as well as an opportunity to hybridize algorithms, as well as create valuable benchmarks for SMT solvers. Even though significant work has been put into HSTT, optimal solutions for most instances are still not known and this is still an active research area.

In this paper, we investigate the formulation of HSTT as SMT. A SMT problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. It is a generalization of the Satisfiability problem (SAT) in which sets of variables are allowed to be replaced by predicates from a variety of underlying theories. SMT is usually used for verification and program analysis, but researchers have recently been investigating solving Constraint Satisfaction Problems with SMT [3] and other optimization problems [11]. There is a natural connection between timetabling and logical formulas. HSTT as itself has many logic based characteristics and as such some of its constraints can easily be encoded as SMT. This has motivated us to investigate how efficient can a SMT formulation for HSTT be. However, due to the generality of the specification that we use, devising a complete model is not a trivial task, because as we will see later, some of the constraints are cumbersome. In addition to formulating a general formulation, one needs to take care of important special cases which arise in practice and can significantly simplify the encoding.

The main contributions of this paper are as follows:

- We show that HSTT can be modeled as a SMT problem, despite the fact that HSTT is very general and has many different constraints, both hard and soft versions. All constraints are included in their general formulations, as well as important alternative encodings for special cases.
- We give preliminary experimental evaluation of our model using both artificial and real-world instances, all of which were taken from the Third International Timetabling Competition 2011 benchmark repository. A comparison with the winning algorithm from ITC 2011 is given.

The rest of the paper is organized as follows: in the next section, we give a more detailed look into the problem description which serves as an introduction for Section 3, where the detailed presentation of our approach in modeling HSTT as

SMT is given. In Section 4, we provide computational results obtained on artificial and real life problems. Finally, we give concluding remarks and ideas for future work.

2 Problem Description

In our research we consider the general formulation of the High School Timetabling problem, as described in [14].

The general High School Timetabling formulation specifies three main entities: times, resources and events. Times refer to time slots which are available, such as Monday 9:00-10:00, Monday 10:00-11:00, etc. Resources correspond to available rooms, teachers, students, etc. The main entities are the events, which in order to take place require certain times and resources. An event could be a Mathematics lecture, which requires a math teacher and a specific student group (both considered resources) and two time slots.

Constraints impose limits on what kind of assignments are legal. These may constraint that a teacher can teach no more than five lessons per day, that younger students should attend more demanding subjects (e.g. Mathematics) in the morning, etc. We describe the constraints in the next section when we present the SMT formulations.

Each constraint has a nonnegative cost function associated with it, which penalizes assignments that violate it. It is important to differentiate between hard and soft constraints. Hard constraints are constraints that define the feasibility of the solution and are required for the solution to make sense, while soft constraints define desirable situations, which define the quality of the solution. Therefore, the cost function consists of two parts: infeasibility value and objective value. The goal is to first minimize the infeasibility and then minimize the objective function value part. The exact way these two are calculated will be discussed in the next section.

3 Our Approach - Modeling HSTT for SMTs

Modern SMT solvers (e.g. z3 [5], Yices [6]) offer a number of underlying theories to choose from, which are described in detail in the standardization SMT-LIB [2]. Modeling of the problem at hand depends heavily on which theory we have chosen. In our initial phase, we used two different theories: linear integer arithmetic and bit vector. In the following, we present a bit vector formulation for HSTT, as it was more successful in initial experiments and afterwards discuss briefly the problems encountered with linear integer arithmetic.

3.1 Bitvector Theory

A bitvector is a vector of bits. The size of the vector is arbitrary, but fixed. A number of standard operations (e.g. *addition*, *and*, *or* operations on bitvectors) and predicates (e.g. equality) are defined over bitvectors and an instance consists of a conjunction of predicates. Most SMT solvers accept formulas written in SMT-LIB file format, but can have their own formats, like Yices. Since these files use prefix notation, we will do so as well in the description of the constraints with the addition of brackets and comas in order to ease reading. E.g. In infix notation one would write $(a = b)$, while in prefix notation the same expression would be written as $(= a\ b)$, while we choose to write $(= (a, b))$.

Most operations are interpreted as usual and all bitvector operands are of the same length. In the following we present some of the notations we will use in which bv_a and bv_b are bitvectors and k is a constant integer:

- $inv(bv_a)$ - inverts bv_a bits (e.g. $inv(1011001) = 0100110$).
- $add(bv_a, bv_b)$ - adds two bitvectors in the same way two unsigned integers would be added (overflow might occur).
- $or(bv_a, bv_b)$ - performs bitwise *or* on its operands.
- $lshift(bv_a, k)$ - applies noncyclic left shift by k operation on bv_a (e.g. $lshift(10011, 2) = 01100$).
- $rshift(bv_a, k)$ - similar to $lshift$, but uses right shifting.

– $extract(bv_a, k)$ - returns the $k - th$ bit of bv_a

An example SMT instance would be the following:

$$(= (1010, lshift(bv_a, 1)) \wedge (< (bv_a, 1000)) \qquad (1)$$

The problem is to determine whether there exists a bitvector bv_a for which the above formula holds. It states that bv_a must be equal to 1010 after it is shifted to the left by one place (first clause) and bv_a must be less than (in the standard way binary numbers are compared) 1000 (second clause). The formula is satisfiable and $bv_a = 0101$ is a model since it satisfied both clauses, while $bv_a = 1101$ is not due to not satisfying the second clause. Note that this is a decision problem.

In the optimization variant, weights may be assigned to clauses and the goal is to find a model which will satisfy all clauses without weights and will minimize the sum of weights of unsatisfied clauses. Optimization is not part of standard SMT solvers by default, although Yices [6] supports it. E.g. if we assigned a weight of 10 to the second clause in the previous example, both $bv_a = 1101$ and $bv_a = 0101$ would be considered solutions to the problem, but the latter would be considered a better solution.

3.1.1 Variables and Definitions

For each event e (e.g. a lesson), we create a number of bit vectors all of length n, where n is the number of time slots available in the instance. The vectors along with their meanings are as follows:

– Y_e - the $i - th$ bit is set (a bit is *set* if it has value 1) if the event is taking place at time slot i and is not set otherwise. In xHSTT terminology, Y_e covers all subevents of event e. This implies that two subevents of the same event can never clash in this representation.
– S_e - the $i - th$ bit is set if the $i - th$ time slot is declared as a starting time for event e and is not set otherwise.

– $K_{e,d}$ - the $i-th$ bit is set if the $i-th$ time slot is declared as a starting time of duration d for event e and is not set otherwise.

As an example of the above variables, take the following bitvectors:

7	6	5	4	3	2	1	0	(*time slot*)
0	1	1	0	0	1	1	0	(Y_e)
0	1	1	0	0	0	1	0	(S_e)
0	1	1	0	0	0	0	0	$(K_{e,1})$
0	0	0	0	0	0	1	0	$(K_{e,2})$

(2)

From Y_e, we see that event e (e.g. a Math lesson) is taking place at time slot 1, 2, 5 and 6, because those bits are set within Y_e. Similarly, time slots 1, 5 and 6 are labeled as starting times from S_e, meaning event e has been split into three subevents. Time slot 1 is labeled as a double lesson by $K_{e,2}$, while 5 and 6 as lessons of duration 1 by $K_{e,1}$. Note that time slot 5 could have also been labeled as a double lesson instead of having two lessons of duration 1. Reasons for choice one possibility over the other is regulated by constraints.

In the formal specification of HSTT, there are no restrictions on what can be defined as a starting point. One could regard a starting point as a time t where a lecture takes place, but has not took place at $t-1$. However, while this is true, this cannot be the only case when a time would be regarded as a starting time, since e.g. time $t = 5$ and $t = 6$ might be interpreted as *last time slot of Monday* and *first time slot of Tuesday* and an event could be scheduled on both of these times, but clearly we must regard both times as starting times, since a double lecture does not extend over such long periods of time. Therefore, any time can in general be regarded as a starting time. It is of interest to note that the previous assignment, by the general formulation, could also be treated as a double lesson for the purpose of constraints, even though it extends over two days. Constraints give more control over these kind of assignments.

Formalities that are tied to starting times with regard to the specification are expressed as follows:

If a starting time for event e has been assigned at time t, then the corresponding event must also take place at that time:

$$\bigwedge_{\forall e \in E} (= (or(S_e, Y_e), Y_e)) \tag{3}$$

The *or* and $=$ statements are required to ensure that Y_e has bits set at least in the same positions as S_e. This type of encoding is used frequently and one should become accustomed to it.

Event e starts at time t if e is taking place at time t and it is not taking place at time $(t-1)$:

$$\bigwedge_{\forall e \in E_{spec})} = (or(and(Y_e, lshift(inv(Y_e, 1)), S_e), S_e) \tag{4}$$

Note that the ordering of the application of inv and $lshift$ is important.

Let K_e^+ be the bit vector which $i-th$ bit is set if any of $K_{e,d}$ vectors have an $i-th$ bit set. This is obtained by taking the *or* of all of the $K_{e,d}$. If time t has been set as a starting time, associate a duration with it:

$$\bigwedge_{\forall e \in E_{spec}} (= (K_{e,d}^+, or(S_{e,t}, K_{e,d}^+)) \tag{5}$$

Let S_e^d be the vector obtained as $rshift(S_e, d)$. If a subevent of duration d has been assigned and immediately after the event is still taking place, then assign that time as a starting time:

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ \forall d \in D}} (= (or(and(rshift(Y_e, d), K_{e,d}), S_e^d), S_e^d)) \tag{6}$$

Let $K_{e,d}^*$ be the vector obtained by taking the *and* of all of Se^k for $k = 1..d$ and Y_e. When a bit in $K_{e,d}$ is set, ensure that the event in question must take place for d consecutive hours during this specified time:

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ \forall d \in D}} (= (or(K^*_{e,d}, K_{e,d}), K^*_{e,d})) \tag{7}$$

Let $K^{inv(k)}_{e,d}$ be the vector obtained as $rshift(inv(K_{e,d}), k)$, $K^{\#k}_{e,d}$ be the vector obtained by taking and of all $K^j_{e,d}$ for $j = 1..(k-1)$ and $K^{\&k}_{e,d}$ be the vector obtained by taking the and of all $K^{\#i)}_{e,d}$ for $i \neq k$. If a duration has been specified for time t, make sure that no other starting point other appropriate $K_{e,t,d}$ variables must be false:

$$\bigwedge_{\forall e \in E_{spec}\ atopd \in D} (= (or(K_{e,d}, K^{\#d}_{e,d}), K_{e,d})) \tag{8}$$

3.2 Cardinality Encodings

An important constraint that arises often is to determine the number of set bits in a bit vector, as well as to impose penalties if the appropriate number of bits are not set. E.g. if an event must take place for two hours, then exactly two bits in its Y_e must be set.

Let us define a unary operation $reduceBit(bv_a) = bv_a \wedge sub(bv_a, 1)$. When applied to bv_a, as the name suggests, it produces a new bitvector which has one less bit set then bv_a. For example:

$$\begin{array}{c|cccccc|c} \wedge & 1 & 1 & 0 & 1 & 0 & 0 & (bv_a) \\ & 1 & 1 & 0 & 0 & 1 & 1 & (sub(bv_a, 1)) \\ \hline & 1 & 1 & 0 & 0 & 0 & 0 & (reduceBit(bv_a)) \end{array} \tag{9}$$

The original bitvector had three bits set, while the produced one was two set. The $reduceBit$ operations is an important part for defining cardinality constraints.

In order to ensure that *at least* k bits are set in a bitvector, we apply $reduceBit$ $k-1$ times and require that the resulting bitvector must be different from zero. For *at most* k, we apply *reduceBit* k times and constrain that the resulting bitvector must be equal to zero. For *exactly* k we encode *at least* k and *at most* k. For example, asserting that *at least* 2 bits are set is done in the following way:

$$\begin{array}{c|cccccc|c}
\wedge & 1 & 1 & 0 & 1 & 0 & 0 & (bv_a) \\
 & 1 & 1 & 0 & 0 & 1 & 1 & (sub(bv_a, 1)) \\
\hline
\wedge & 1 & 1 & 0 & 0 & 0 & 0 & (reduceBit(bv_a)) \\
 & 1 & 0 & 1 & 1 & 1 & 1 & (sub(reduceBit(bv_a)), 1) \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 & (reduceBit(reduceBit(bv_a)))
\end{array} \tag{10}$$

Since the final bitvector is different from the zero bitvector, we conclude that *at least* 2 bits are set in bv_a.

For the soft cardinality constraints which penalize the objective value if a certain number of bits is set rather than forbidding their assignments, a similar technique. For *at least* k, it is asserted before each $i-th$ application of *reduceBit* that the current bitvector is different from zero and is penalized by some weight if it is not the case. For example, asserting that *at least* 2 bits are set is done in the following way for the soft version:

$$\begin{array}{c|cccccc|c}
\wedge & 0 & 1 & 0 & 0 & 0 & 0 & (bv_a \neq 0, no\ penalty) \\
 & 0 & 0 & 1 & 1 & 1 & 1 & (sub(bv_a, 1)) \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 & (reduceBit(bv_a) = 0, penalize)
\end{array} \tag{11}$$

Note that we checked for penalties in two cases (for the initial bitvector bv_a and $reduceBit(bv_a)$), but only one case was penalized. For *at most* k, a similar algorithm is used. First, *bitReduce* is applied k times as in the regular cardinality constraint version. Then, *bitReduce* is applied $n-k$ times to this bitvector (n is the size of bv_a) and before each application it is asserted that the current bitvector is zero and is penalized by some weight if it is not the case. Note that if we have some hard constraint limiting the maximum number of bits that may be set in a bitvector to some k_{max}, we do not perform the second part of the algorithm $n-k$ times, but rather just $k_{max}-k$ times. This was used frequently in the implementation.

3.3 Constraints

Each constraint has its points of application and each point generates a number of deviations. Cost of the constraint is obtained by applying a cost function on the set of deviations produced and multiplying it by a weight. A cost function may simply be the sum of all deviations. Our current implementation supports cost functions of sums of deviations, while cost function sum of squares of deviations is supported by the model but not implemented. The HSTT specification allows for other cost functions as well, such as square of sums, but we do not have an encoding for them currently. Fortunately, only two instances use nonsupported cost functions (KosovaInstance1 and StPaulEngland instances). Some constraints are always encoded as hard (e.g. Avoid Clashes Constraints, Assign Times Constraints) and because of this we avoid discussing their soft constraint variants.

We simplify the objective function by not tracking the infeasibility value, rather regarding it was zero or nonzero. By doing so we simplify the computation, possibly offering a faster algorithm.

E, T and R are sets of events, times and resources, respectively. Each constraint is applied to some subset of those three and will be denoted by E_{spec}, T_{spec} and R_{spec}. These subsets are naturally in general different from constraint to constraint. Note that it is possible to have several constraints of the same type, but with different subsets defined for them.

We present encodings used in the experimental results, in which we assume that all resources are already assigned to events. We make this assumption as this eases the modeling and readability of the constraints. Later on we provide a description on how this limitation can be overcome.

3.3.1 Assign Time Constraints

Every event must be assigned a given amount of times. For example, if a lecture lasts for two hours, two time slots must be assigned to it.

Each event's Y_e vector must have exactly d bits set, where d is the duration of the event:

$$\bigwedge_{\forall e \in E} (exactly\_d[Y_{e,t} : t \in T]) \tag{12}$$

3.3.2 Avoid Clashes Constraint

Specified resources can only be used at most by one event at a time. For example, a student may attend at most one lecture at any given time.

Let $E(r)$ be the set of event which require resource r. For each resource r, each time slot i and each combination of two Y_e vectors of events from $E(r)$ at most one bit at $i-th$ location may be set.

$$\bigwedge_{\forall r \in R \forall e_1, e_2 \in E(r) e_1 \neq e_2} (= (and(Y_{e_1}, Y_{e_2}), 0)) \tag{13}$$

For example, for resource r let $E(r) = \{Y_{e_1}, Y_{e_2}, Y_{e_3}\}$ and let this constraint be defined for r.

$$\begin{array}{cccccccc} \wedge & 0 & 0 & 1 & 1 & 1 & 1 & (Y_{e_1}) \\ & 0 & 1 & 0 & 0 & 0 & 0 & (Y_{e_2}) \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & (= 0) \end{array} \tag{14}$$

The previous check ensures that there are no clashes for Y_{e_1} and Y_{e_2}.

$$\begin{array}{cccccccc} \wedge & 0 & 0 & 1 & 1 & 1 & 1 & (Y_{e_1}) \\ & 0 & 0 & 0 & 0 & 1 & 0 & (Y_{e_3}) \\ \hline & 0 & 0 & 0 & 0 & 1 & 0 & (\neq 0, violated) \end{array} \tag{15}$$

However, since a clash exists between Y_{e_1} and Y_{e_3}, the constraint is detected to be violated and some changes to the Y_{e_1}, Y_{e_2}, Y_{e_3} bitvectors must be made.

3.3.3 Avoid Unavailable Times Constraints

Specified resources are unavailable at certain times. For example, a teacher might be unable to work on Friday.

For each resource r, each unavailable time slot i and each Y_e vector of events from $E(r)$ we force the $i-th$ bit to be set to zero.

$$\bigwedge_{\forall r \in R_{spec} \forall e \in E(r) \forall i \in T_{spec}} (= (extract(Y_e, i), 0)) \tag{16}$$

If this constraint is used as a soft constraint, all of the above clauses would be assigned the given weight, as points of application are resources and deviations are calculated as the number of times a resource is assigned to an unavailable time.

For example, if time slots 1 and 4 are unavailable for resource r and event e requires r:

$$\begin{array}{cccccccc|c} 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & (time\ slot) \\ \hline 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & (Y_e) \end{array} \tag{17}$$

Y_e would violate this constraint, as Y_e is taking place on time slot 1, which is a unavailable one, meaning that a different bitvector needs to be assigned to Y_e.

3.3.4 Split Events Constraints

This constraint has two parts.

The first part limits the number of starting times an event may have within certain time frames. For example, an event may have at most one starting time during each day, preventing it from being fragmented within days.

The second part limits the duration of the event for a single subevent. For example, if four time slot must be assigned to a Mathematics lecture, we may limit that the minimum and maximum duration of a subevent is equal to 2, thus ensuring that the lecture will take place as two blocks of two hours, forbidding having the lecture performed as one block of four hours.

This constraint specifies the minimum A_{min} and maximum A_{max} amount of starting times for the specified events:

$$\bigwedge_{\forall e \in E_{spec}} (atLeast\_A_{min}[S_{e,t}] \wedge atMost\_A_{max}[S_{e,t}) \qquad (18)$$

In addition, this constraint also imposes the minimum d_{min} and maximum d_{max} duration for each subevent.

$$\bigwedge_{\forall e \in E_{spec} \forall d \in ili < d_{min} \vee i > d_{max}} (atMost\_0[K_{e,d}]) \qquad (19)$$

If the constraint is specified as soft, then the soft cardinality encodings are used instead. Points of applications are events and deviations are calculated as the number of times an event has been assigned a duration which is less than d_{min} or greater than d_{max}, plus the amount by which the number of starting times for the event event falls short of A_{min} or exceeds A_{max}.

3.3.5 Spread Events Constraints

Certain events must be spread across the timetable, e.g. in order to avoid situations in which an event would completely be scheduled only in one day.

An event group eg is a set of events. Let vector Z_{eg} be a bit vector which has its $i-th$ bit set iff an event $e \in eg$ is taking place at time i. This is obtained by applying *or* to all of the appropriate Y_e vectors.

This constraint specifies event groups to which it applies, as well as a number of time groups (sets of times) and for each such time group the minimum and maximum number of starting times an event must have within times of that time group. Let TG_{spec} denote this set of sets of times and let $mask_{tg}$ be the bit vector which has its $i-th$ bit set iff i is a time slot of time group tg:

There must be at least d_i^{min} starting times within the given time groups (min is a subscript, not exponentiation):

$$\bigwedge_{\substack{\forall tg_i \in TG_{spec} \\ eg \in EG_{spec}}} (atLeast\_d_i^{min}[and(Z_{eg}, mask_{tg})]) \qquad (20)$$

A similar encoding to the one above is also used, but with $atMost\_d^{max}$.

If this constraint is used as a soft constraint, the soft cardinality constraint is used instead. Points of application are event groups (not events) and deviations are calculated as the number by which the events group falls short of the minimum or exceeds the maximum.

3.3.6 Distribute Split Events Constraint

This constraint specifies the minimal and maximum number of starting times of a specified duration. For example, if $duration(e) = 10$, we may impose that the lecture should be split so that at least two starting times must have duration three. Formally:

There must be at least d_{min} starting times with given duration d:

$$\bigwedge_{\forall e \in E_{spec}} (atLeast\_d_{min}[K_{e,d}] \wedge atMost\_d_{max}[K_{e,d}]) \qquad (21)$$

3.3.7 Limit Busy Times Constraints

This constraints imposes limits on the number of times a resource can become busy within certain a time group, if the resource is busy at all during that time group. For example, if a teacher teaches on Monday, he or she must teach at least for three hours. This is useful in preventing situations in which teachers or students would need to come to school for only to have a lesson or two.

A resource is busy at a time group tg iff it is busy in at least one of the time slots of the tg. Let TG_{spec} denote this set of sets of times:

$$\bigwedge_{\substack{\forall r \in R_{spec} \\ \forall tg \in TG_{spec}}} (or(atLeast\_b_{min}[and(Y_e, mask_{tg})], (= (and(Y_e, mask_{tg})), 0))) \tag{22}$$

A similar encoding to the one above is also used, but with $atMost\_b_{max}$. Note that in this case *or* represents *logical or*, rather than *bitvectoror*.

If this constraint is used as a soft constraint, the soft cardinality constraint is used instead, although special care must be given as this is a conditional cardinality constraint: if the calculated vector is different from zero then the cardinality constraints need to be fulfilled. Points of application are resources and each resource generates multiple deviations (one for each time group) which calculated as the number by which the events group falls short of the minimum or exceeds the maximum.

3.3.8 Limit Idle Times Constraints

This constraint specifies the minimal and maximum number of times in which a resource can be idle during the times in the specified time groups. For example, a typical constraint is to impose that teachers must not have any idle times.

A time slot t is idle with respect to time group tg (set of times) if it is not busy at time t, but is busy at an early time and at a later time of the time group tg. For example, if a teacher teaches classes Wednesdays at $Wed2$ and $Wed5$, he or she is idle at $Wed3$ and $Wed4$, but is not idle at $Wed1$ and $Wed6$. This constraint places limits on the number of idle times for each resource. Let vector $G_{e,tg}$ be the vector obtained by taking *or* of bitvectors $and(and(Y_e, mask_{tg}), rshift((Y_e, mask_{tg}), k))$ where $k = 1..n$ and n is the number of times in time group tg. Let vector $H_{e,tg}$ be similar, except using $lshift$ instead of $rshift$. We then encode the constraint as follows:

There must be at least $idle_{min}$ idle times during a time group:

$$\bigwedge_{\substack{\forall tg \in TG_{spec} \\ r \in R_{spec}}} (atLeast\_idle_{min}[and(inv(Y_e), and(H_{e,tg}, G_{e,tg}))]) \tag{23}$$

A similar encoding to the one above is also used, but with $atMost\_idle_{max}$. If this constraint is used as a soft constraint, the soft cardinality constraint is used instead.

3.3.9 Cluster Busy Times Constraints

This constraint specifies the minimal and maximum number of specified time groups in which a specified resource can be busy. For example, we may specify that a teacher must fulfill all of his or her duties in at most three days of the week.

We first define a helper bitvector B_r for each resource, in which $i-th$ bit is set iff the resource is busy at the $i-th$ time group. Therefore, $i-th$ bit in B_r is equal to the *or* operation on all of the $i-th$ bits of bitvectors in E(r). With this helper bitvector, we may now encode the constraint as:

There must be at least b_{tg}^{min} busy time groups:

$$\bigwedge_{\forall r \in R_{spec}} (atLeast\_b_{tg}^{min}[B_r]) \tag{24}$$

A similar encoding to the one above is also used, but with $atMost\_b_{max}$. If this constraint is used as a soft constraint, the soft cardinality constraint is used instead.

3.3.10 Prefer Times Constraints

This constraint specified that certain events should be held at certain times. If an optional parameter d is given, then this constraint only applies to subevents assigned duration d. For example, a lesson of *duration 2* must be scheduled on Monday, excluding the last time slot on Monday.

Let P_e be the bitvector in which $i-th$ bit is set iff i is a preferred time. We then encode:

$$\bigwedge_{\forall e \in E_{spec}} (atMost\_0[and(\star, inv(P_e))]) \tag{25}$$

where $\star$ is either Y_e or $K_{e,d}$, depending on whether the optional parameter d is given. Note that this constraint is not the same when the optional parameter is not given and when $d = 1$.

3.3.11 Order Events Constraints

This constraint specifies that one event can start only after another one has finished. In addition to this, parameters B_{min} and B_{max} are given which define the minimum and maximal separations between the two events and are by default set to zero and the number of time slots, respectively. The constraint specifies a set of pairs of events to which it applies.

If the first event in a pair is taking place at time t, then the second event cannot take place at time $t + B_{min}$ nor at any previous times:

$$\bigwedge_{\forall (e_1,e_2) \in E^2_{spec}} (< (lshift(e_i, B_{min}), e_j)) \tag{26}$$

A similar encoding to the one above is also used, but with $>$ and B_{max}. Special care must be taken as overflows may happen during the shift operations.

3.3.12 Link Events Constraints

Certain events must be held at the same time. For example, physical education lessons for all classes of the same year must be held together. This constraint specifies a certain number of event groups and imposes that all events within an event group must be held simultaneously. Let EG_{spec} denote this set of sets of events. All events within an event group must be held at the same times:

$$\bigwedge_{\substack{\forall eg \in EG_{spec} \\ e_j \in eg}} (atMost\_0[and(Z_{eg}, inv(Y_{e_j}))]) \tag{27}$$

If the constraint is declared a soft one, the soft cardinality constraint is used instead. Points of application are event groups (not events) and deviations are calculated as the number of times in which at least one of the events within the event group is taking place, but not all of them.

3.3.13 Extending the Model

As mentioned in the beginning, we made the assumption that all resources have been already assigned to events, as it is easier to model, implement and present the formulation. This is a reasonable assumption, as most instances are of this form. Still, a significant part of the instances require assignments of resource to events. Our model is easy to extend with these requirements by introducing new bitvectors: for each event e and resource r, a bitvector is created in which $i - th$ bit is set iff resource r has been assigned to event e at time i. With these bitvectors, the other resource assigning constraints (we direct interested readers to [14]) can be encoded in a similar fashion as the ones already presented, along with certain modifications need to be made to Assign Time and Avoid Clash constraints.

However, special care needs to be given when doing so to concrete instances, as requirements for resource assignments can be diverse. For example, in instance $SpainInstance$ given in the ITC repository, assignments consist of assigning one gym room out of two available. For instance $EnglashStPaul$, room need to be assigned and many symmetries appear because all rooms are identical. Hence, it might be a better idea to restrict the number of events at each time to the number of rooms, rather than assigning rooms directly to events.

In addition, it may be of interest to simplify the $K_{e,d}$ and S_e encodings which would simply state that if an event has three consecutive bits set it is treated as a subevent of duration 3 rather than of the complicated formulation given or that only the first constraint regarding S_e should be used. The reason the encoding is so complicated is because of the way the general formulation specifies starting times, but this is not necessary for all instances.

3.3.14 Other Theories

We have done some initial experiments with linear integer arithmetic theories in two different formulations. One of the formulations had variables which were restricted to binary values and the encodings were similar to a pure SAT formulation, except that the cardinality constraints could be encoded more elegantly. The second encoding took more advantage of the integer arithmetic in which for each event we create a number of variables equal to its duration. The value of the variable determines which time slot the event takes place. This reduced the number of variables significantly when compared to the binary version, but some constraints were harder to encode. However, regardless of that, both modeling options failed to produce any solutions to problem instances, even when only Assign Time, Avoid Clashes and Prefer Times constraints were used. Therefore, we did not continue with these modelings and continued with the bitvector formulation, which performed better in these initial experiments.

4 Computational Results

In our current experiments we evaluated our approach on some benchmark instances from HSTT which can be found on the repository of the International Timetabling Competition 2011 [1]. A subset of instances which were suggested by the competition as test beds, as well as the ones used in the competition have been chosen (these two sets intersect). All tests were performed on (Intel Core i3-2120 CPU @ 3.30GHz with 4 GB RAM) and each instance was given a single core. We restricted the computational time to 24 hours per instance.

In the instances, the number of time slots ranges from 25 to 142, number of resources from 8 to 99, number of events from 21 to about 350 (exception is the Italy4 instance with

748 events) with total event duration from 75 to around 1000. These numbers are approximations and vary heavily from instance to instance. We do not provide detailed information, but direct the interested reader to [12] [1] [14].

In the tables below, we denote the objective function cost by (x, y), where x is the infeasibility value and y is the objective value. If a dash is used, that means that the solver failed to produce a solution. If an x is used, that means that we had not provided an objective value, as in initial solution has been generate in which only hard constraints had been considered and the resulting objective value is essentially random.

We experimented with the SMT solver Yices 1.0.40 (released December 4, 2013) [6]. It was chosen because it is the only SMT solver to our knowledge which directly supports optimization, rather than just checking for satisfiability.

4.1 Comparisons of Results

We compare results we had obtained with our approach and GOAL (the winning team of the competition). GOAL's algorithm first generates an initial solution using KHE [10] and then performs its heuristic search algorithm. Note that the initial solution generated can be unfeasible and in some cases the algorithm fails to improve this solution to a feasible one.

In the table below we present the computational results. To make our comparison fair, we ran our approach and GOAL on the same computer platform and each solver was restricted to 24 hours and was given one core. The source code of GOAL was provided by their authors [4]. The time to convert an instance from xHSTT to a SMT instance is negligible (a few seconds at most) when compared to the SMT solution process.

As we see in the table, we provide experimental results for 11 instances. Other instances were not included because the current implementation does not support them. The reasons for this are either that we did not yet implement constraints which allow resource assignments (e.g. Assign Resource Constraints), use the square of sums (we currently do not have a model for this cost function) or the sum of squares cost function (we have a model for this cost function but is not yet implemented).

The abbreviations used in the columns are as follows: OA is our approach, GOAL is the winning team algorithm:

Name	OA	GOAL
BrazilianInstance1	**(0, 47)**	(0, 54)
BrazilianInstance2	**(0, 60)**	(1, 42)
BrazilianInstance4	**(0, x)**	(16, 95)
BrazilianInstance5	**(0, x)**	(4, 121)
BrazilianInstance6	**(0, x)**	(4, 195)
BrazilianInstance7	**(0, x)**	(11, 230)
SouthAfricaLewitt2009	(0, x)	**(0, 18)**
SouthAfricaWoodlands	(-, -)	**(2, 13)**
GreeceHighSchool	**(0, 0)**	**(0, 0)**
ItalyInstance1	(-, -)	**(0, 19)**
ItalyInstance4	(-, -)	**(0, 57)**

Table 1 Results obtained after 24 hours.

There might be differences in the results obtained by GOAL in the competition and obtained by our 24 hour runs, because in the competition competitors in the final phase were given one month to use whatever available resources to provide the best results. We focus here on the comparison with the winner of ITC competition, because we think that this gives a good idea how good our approach performs in a limited amount of time compare to one of best existing approaches for this problem. For some of the instances, better upper bounds were obtained after the competition by GOAL and other approaches without time or resource limitations.

It is interesting to note that Yices found an initial solution for all instance except three quickly (within 10 minutes for all but SouthAfricaLewitt which took several hours), but had not managed to perform any optimization for most instances within the given time limit. Even so, as we can see from Table 1, from the examples in which our encoding was done successfully, our approach could find feasible solutions in which GOAL could not in seven instances within the given time limit. A feasible solution has been found for all except for three instances. In the case of the Italy4 instance, the program ran out of memory.

Overall, we conclude that the SMT approach can provide feasible solutions in short time for several instances. Further research is needed to successfully apply this technique for the optimization variant. In general, it seems that SMT strengths are in satisfiabiliy rather than optimization, while GOAL could be used for optimizing solutions which are (near) feasibility.

5 Conclusion

High school timetabling is a wide spread and important problem and because of this, developing algorithms to solve the problem are of great importance.

In this paper, we have shown that the general HSTT problem [14] can indeed be modeled as a SMT problem, despite the generality of the specification, with the exception of not being able to model the square of sums of deviation cost function. We presented a complete and detailed encoding using theory of bitvectors in the general sense as required by the specification under the assumption that resources had been preassigned to events, but have sketched how the model can be extended and discussed some important special cases.

We implemented and evaluated our approach on a subset of benchmark instances suggested and used by the Third International Timetabling Competition 2011 and compared our results with GOAL, the winning team of the Third International Timetabling Competition 2011. For some of the tested instance, our approach managed to find feasible solutions within a given time limit and there is space for further improvements. Generated encodings solve practical problems and as such can be used as benchmarks for the evaluation of SMT solvers.

Furthermore, we plan on to investigate hybridization of our approach with heuristic techniques (e.g. develop a large neighborhood search algorithm) that will utilize SMT.

Acknowledgements The work was supported by the Vienna PhD School of Informatics and the Austrian Science Fund (FWF): P24814-N23.

References

1. International timetabling competition 2011. http://www.utwente.nl/ctit/hstt/itc2011/welcome/. Accessed: 2014-1-30
2. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: Version 2.0. In: Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England), vol. 13, p. 14 (2010)
3. Bofill, M., Suy, J., Villaret, M.: A system for solving constraint satisfaction problems with SMT. In: Theory and Applications of Satisfiability Testing–SAT 2010, pp. 300–305. Springer (2010)
4. Brito, S.S., Fonseca, G.H.G., Toffolo, T.A.M., Santos, H.G., Souza, M.J.F.: A SA-VNS approach for the high school timetabling problem. Electronic Notes in Discrete Mathematics **39**, 169–176 (2012)
5. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer (2008)
6. Dutertre, B., De Moura, L.: The Yices SMT solver. Tool paper at http://yices. csl. sri. com/tool-paper. pdf **2**, 2 (2006)
7. Even, S., Itai, A., Shamir, A.: On the complexity of time table and multi-commodity flow problems. In: Foundations of Computer Science, 1975., 16th Annual Symposium on, pp. 184–193. IEEE (1975)
8. Fonseca G. H. G., S.H.G.T.T.A.M.B.S.S.S.M.J.F.: A SA-ILS approach for the high school timetabling problem. In: In Proceedings of the ninth international conference on the practice and theory of automated timetabling, PATAT (2012)
9. Kheiri, A., Ozcan, E., Parkes, A.J.: HySST: hyper-heuristic search strategies and timetabling. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012) (2012)
10. Kingston, J.H.: The KHE high school timetabling engine (2010)
11. Nieuwenhuis, R., Oliveras, A.: On SAT modulo theories and optimization problems. In: Theory and Applications of Satisfiability Testing-SAT 2006, pp. 156–169. Springer (2006)
12. Post, G., Ahmadi, S., Daskalaki, S., Kingston, J., Kyngas, J., Nurmi, C., Ranson, D.: An XML format for benchmarks in high school timetabling. Annals of Operations Research **194**(1), 385–397 (2012). DOI 10.1007/s10479-010-0699-9. URL http://dx.doi.org/10.1007/s10479-010-0699-9
13. Post, G., Di Gaspero, L., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. Annals of Operations Research pp. 1–7 (2012)
14. Post, G., Kingston, J.H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., et al.: XHSTT: an XML archive for high school timetabling problems in different countries. Annals of Operations Research pp. 1–7 (2011)
15. Sørensen, M., Dahms, F.H.: A two-stage decomposition of high school timetabling applied to cases in Denmark. Computers & Operations Research **43**, 36–49 (2014)
16. Sørensen, M., Kristiansen, S., Stidsen, T.R.: International timetabling competition 2011: An adaptive large neighborhood search algorithm. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), p. 489 (2012)
17. Sørensen, M., Stidsen, T.R.: Comparing solution approaches for a complete model of high school timetabling. Tech. rep., DTU Management Engineering (2013)

Indoor football scheduling

Dries Goossens · Frits Spieksma

Abstract This paper deals with a real-life scheduling problem from an amateur indoor football league. The league consists of a number of divisions, in each of which a double round robin tournament is played. The goal is to develop a schedule which avoids close successions of matches involving the same team. This scheduling problem is interesting, because matches are not planned in rounds. Instead, each team has a number of time slots available to play its home games. Furthermore, in contrast to professional leagues, alternating home and away matches is hardly relevant. We present a mathematical programming formulation and a heuristic based on tabu search, which resulted in high-quality schedules that have been adopted in practice.

Keywords scheduling · non-professional · indoor football · time-relaxed · tabu search

1 Introduction

In this paper, we discuss the sports scheduling problem faced by the "Liefhebbers Zaalvoetbal Cup (LZV Cup)", an amateur indoor football league founded in 2002. This league currently involves 87 teams, all situated in the vicinity of Leuven (Belgium), grouped in 6 divisions. The LZV Cup focusses on teams that

D. Goossens
Ghent University, Faculty of Economics and Business Administration
Management Information Science and Operations Management
Tweekerkenstraat 2, B-9000 Ghent
E-mail: dries.goossens@ugent.be

F. Spieksma
K.U. Leuven, Faculty of Business and Economics
Operations Research and Business Statistics
Naamsestraat 69, B-3000 Leuven
E-mail: frits.spieksma@kuleuven.be

consist of friends, is open to all ages, and considers fair play of paramount importance. The matches are played without referees, since, according to the organizers, "referees are expensive, make mistakes, and invite players to explore the borders of sportsmanship" (see also http://www.lzvcup.be [in Dutch]).

Academic interest for scheduling amateur leagues is rather limited, compared to the attention that professional leagues receive. Perhaps this can be explained by the fact that a lot less money is involved with non-professional leagues. In addition, scheduling constraints coming from broadcasting rights, security forces, public transport, media and fans usually are non-existing. This does not imply however that amateur scheduling problems are less challenging than their professional counterparts. Indeed, stadium or ground availability is in general more limited, because the team's venue tends to be shared with other teams or sports disciplines. Moreover, practical considerations for the players are far more important, since they have other activities (e.g. family, work) as well.

In our indoor football scheduling problem, there are multiple divisions. In each division, each team plays against each other team twice. The teams provide dates on which they can play a home game, and dates on which they cannot play at all. The league organizers are not worried by (long) series of consecutive home games (or away games), but do not want a close succession of two matches featuring the same teams. The goal is to develop a schedule for each of the divisions, where each team has a balanced spread of their matches over the season.

In section 2, we give a formal description of the problem, followed by an overview of the literature on related problems in section 3. We provide a mathematical formulation in section 4, which is used to tackle to the problem with Ilog Cplex. In section 5, we develop a heuristic approach, based on tabu search. Computational experiments with both methods for real-life problem instances we solved for the LZV Cup are described in section 6. The paper ends with conclusions and future work in section 7.

2 Problem description

In this section, we provide a formal problem description, and we introduce the notation used in the remainder of this paper. The teams in the LZV Cup are grouped according to their strength into a division. In each division, a double round robin tournament is played, i.e. each team meets each other team twice (once at its home venue, and once at the opponent's venue). Apart from the number of teams, the problem description is identical for all divisions. A division has a set of teams T, with $|T| = n$, and a set of time slots $S = \{1, 2, ..., |S|\}$, ranging from the first day of the season till the last. All matches should be played within this time frame.

Each team $i \in T$ provides a list of time slots $H_i \subseteq S$ for which their home ground is available. Home games for a team can only be scheduled on time slots from this list. Obviously, if each match is to be scheduled, each team should at least provide as many time slots as it has opponents, i.e. $|H_i| \geqslant n-1$. This list is called the *home game set*. Some teams may have a slot on the same weekday every other week; other teams may have a more irregular home game set. Each team can also provide a list of calender days $A_i \subseteq S$ on which it doesn't want to play a match; we call this list the *forbidden game set*. Teams can use this list to avoid matches in the Christmas and New Year period, on holidays, during an exam period, etc. The forbidden game set implies that on all days not in the list, the team is able to play an away game. We assume that $H_i \cap A_i = \emptyset$. A team is not allowed to play twice on the same day, or more than twice in a period of R_{max} days. Finally, there should also be at least m calendar days between two matches featuring the same teams. Notice that it is allowed to meet an opponent for the second time, before all other opponents have been faced once.

In summary, we have the following constraints:

– each team plays a home match against each other team exactly once [C1]
– home team availabilities H_i are respected [C2]
– away team unavailabilities A_i are respected [C3]
– at least m days between two matches with the same teams [C4]
– each team plays at most one game per day [C5]
– each team plays at most 2 games in a period of R_{max} days [C6]

The goal is to develop a schedule with for each team a balanced spread of their matches over the season. More in particular, teams wish to avoid having two matches in a period of R_{max} days or less. We use p^r to denote the penalty incurred for every pair of consecutive matches played by a team within period of $r \in R = \{2, 3, ..., R_{max}\}$ days. Obviously, having 2 matches in 2 days is considered more unpleasant than having 2 matches in 4 days. The main idea behind this, is that most players prefer not to fully spend their weekend with their sport. Moreover, matches packed together could also lead to injuries. If a team has more than R_{max} days between two consecutive matches, we assume that the league organizers no longer care, and consider any number greater than R_{max} as equally adequate. Constraint C1 is in fact interpreted as a soft constraint, i.e. it is possible not to schedule a match, but only at a high cost. This guarantees feasibility of any instance (e.g. in case some team does not provide enough time slots for which their home ground is available). In practice, if a match cannot be scheduled, the league organizers leave it to the home team to find a suitable date and location to play the match (if the home team fails to organize the match, they lose the game).

3 Related work

A large number of sports scheduling papers deal with professional leagues (see e.g. [2], [4], and [6]; for a complete overview we refer to [9]). Academic interest in problems faced by amateur leagues is far more rare. In general, there are two main differences between professional and amateur leagues. Firstly, in amateur leagues matches are typically not planned in rounds. Scheduling problems in sports leagues can be divided into two types: temporally constrained and temporally relaxed problems. In the first type, the matches are grouped in rounds, and no more than the minimum number of rounds required to schedule all the matches is available. For leagues with an even number of teams, this means that each team plays exactly once in each round. Nowadays, all European professional football schedules are temporally constrained [7]. With temporally relaxed problems, the number of rounds is larger than the minimum number of rounds needed. In this case, some teams will have one or more rounds without a match. In the cricket world cup as reported in [1], 9 teams play a single round robin tournament, resulting in 36 games that need to be scheduled in a 26-day period. In this tournament, several constraints and practical considerations need to be respected, resulting in a schedule that is suitable for a worldwide TV audience. The Australian state cricket season also provides a temporally relaxed scheduling problem, where one of the constraints is that matches between the 6 states must be scheduled around predetermined international and test match fixture dates [15]. Scheduling the National Hockey League, as discussed in [3], involves planning 82 games per team in a period of 28 weeks. In addition to other constraints, no team should not play games on three days straight, nor should it play more than three games in five consecutive days. In our problem, the slots themselves can be seen as rounds, making the schedule extremely temporally relaxed.

Secondly, successions of home or away matches (breaks) are hardly relevant in amateur leagues. The main reason is that there are usually few spectators, and the home advantage is quite limited. Only extremely long series of consecutive home or away matches could be problematic. In our problem, however, they are unlikely, because a team's home game set is usually well balanced over the season, and the number of days where its home venue is available is not much larger than the number of home games. In professional leagues, however, alternation of home and away matches is usually the most important constraint. This is illustrated by the popularity of the so-called "first-break, then schedule" approach (see e.g. [11]), and the attention for the break minimization problem (see e.g. [12]). A break in this alternating sequence is not desirable for the spectators, and less spectators tend to show up for the second or third home match in a row [5].

The concept of a list of dates on which no game should be scheduled for some team has been introduced in [13]; they call it *suspension dates*. The authors discuss a scheduling problem from a regional amateur table-tennis federation in Germany, involving more than 30 leagues. This scheduling problem is quite similar to ours. In each league a double round robin tournament is played; the schedule is temporally relaxed. All games have to be scheduled within a given period, while each team may be involved in at most one game per day. Home teams provide a list of permitted dates, and away teams can also specify a number of dates on which they are not available. The teams also specify the number of days-off they want between two successive games. Unlike in our problem, the season is split in two halves, such that each teams meets each other team once in each half. Furthermore, to be able to make a meaningful ranking, the season is subdivided into six time periods of equal length, and the number of matches that each team has to play in each period is constrained by a lower and an upper bound. The authors solve this scheduling problem with a permutation based genetic algorithm for which feasibility preserving operators are defined. In a follow-up paper [14], the authors propose a memetic algorithm, backed by a constraint propagation based heuristic and use a co-evolutionary approach.

Knust [10] also starts from the table-tennis scheduling problem discussed in [13], but adds a number of constraints (e.g. some matches should be played on weekend days (instead of weekdays), and some matches should be scheduled in specific time intervals). More importantly, for each team home and away matches should be scheduled alternately (i.e. breaks should be avoided). Knust [10] models the problem as a multi-mode resource-constrained project scheduling problem, for which a 2-stage heuristic solution algorithm is proposed, involving local search and genetic algorithms.

4 A mathematical formulation

In this section, we write the scheduling problem more formally as an integer problem. Our main decision variable is x_{ijs}, which is 1 if team i plays a home game against team $j \neq i$ on slot s, and 0 otherwise. The variable y_{ist} is 1 if team i plays a match on slot s, followed by its next match on slot t, for each s and t such that $t > s$ and $t - s + 1 \leqslant R_{max}$, and 0 otherwise. The variable u_{ij} is 1 if no home game of team i against team j is scheduled, and 0 otherwise. Each unscheduled match results in a penalty P. We can now write the following formulation for our problem:

$$\text{minimize} \sum_{i \in T} \sum_{j \in T: i \neq j} P u_{ij} + \sum_{i \in T} \sum_{s \in S} \sum_{t \in S} p^{t-s+1} y_{ist}$$

subject to

$$\sum_{s \in H_i \setminus A_j} x_{ijs} + u_{ij} = 1 \qquad \forall i, j \in T : i \neq j \tag{1}$$

$$\sum_{j \in T} (x_{ijs} + x_{jis}) \leqslant 1 \qquad \forall i \in T, s \in S \tag{2}$$

$$y_{ist} \leqslant \sum_{j \in T} (x_{ijs} + x_{jis}) \qquad \forall i \in T, s, t \in S \tag{3}$$

$$y_{ist} \leqslant \sum_{j \in T} (x_{ijt} + x_{jit}) \qquad \forall i \in T, s, t \in S \tag{4}$$

$$\sum_{j \in T} (x_{ijs} + x_{jis}) + \sum_{j \in T} (x_{ijt} + x_{jit}) - \sum_{j \in T} \sum_{k=s+1}^{t-1} (x_{ijk} + x_{jik}) - 1 \leqslant y_{ist} \qquad \forall i \in T, s, t \in S \tag{5}$$

$$x_{ijs} + x_{jis'} \leqslant 1 \qquad \forall i, j \in T, s \in H_i, s' \in H_j : |s - s'| < m \tag{6}$$

$$\sum_{j \in T} \sum_{k=s}^{s+R_{max}-1} (x_{ijk} + x_{jik}) \leqslant 2 \qquad \forall i \in T, s \in S \tag{7}$$

$$x_{ijs} = 0 \qquad \forall i, j, \in T, s \notin H_i \vee s \in A_j \tag{8}$$

$$x_{ijs} \in \{0, 1\} \qquad \forall i, j, \in T, s \in S \tag{9}$$

$$y_{ist} \in \{0, 1\} \qquad \forall i \in T, s, t \in S \tag{10}$$

$$u_{ij} \in \{0, 1\} \qquad \forall i, j \in T : i \neq j \tag{11}$$

The objective function minimizes the number of unscheduled matches, and penalizes each pair of matches scheduled within R_{max} days. The first set of constraints ensures that each team meets each other team exactly once in a home game, unless the match is not scheduled. Consequently, each team will meet each other team exactly once in an away game as well, and these constraints are sufficient to construct a double round robin tournament [C1]. The next set of constraints make sure that each team plays at most once per time slot [C5]. Constraints (3)-(5) keep track of the number of days between two consecutive matches featuring the same team. The next set of constraints puts at least m calendar days between the two confrontations of a pair of teams [C4]. Constraints (7) enforce that a team plays at most two matches in a period of R_{max} slots [C6]. Constraints (8) make sure that there is no match between two teams on a particular time slot if the home team does not have its venue available [C2], or if the away team marked this time slot in its forbidden game set [C3]. The final sets of constraints state that all variables are binary.

5 A heuristic approach

This section describes our heuristic approach, which is based on tabu search. The core component of our algorithm consists of solving a transportation problem, which schedules (or reschedules) all home games of a team $i \in T$. This

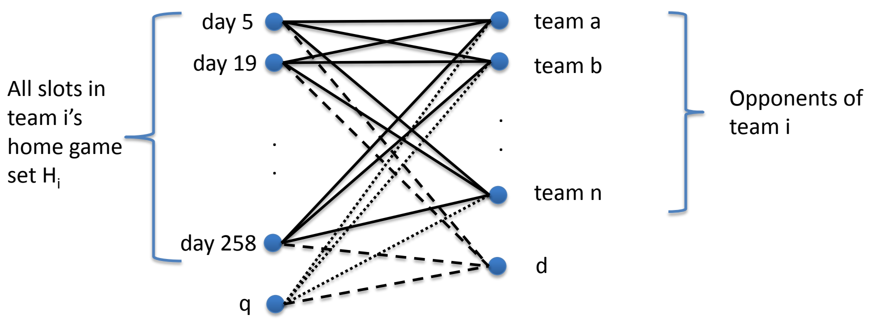


Fig. 1 The graph G_i (example)

transportation problem is explained in section 5.1. Our heuristic approach consists of three phases: the construction phase (section 5.2), the tabu phase (section 5.3), and the perturbation phase (section 5.4).

5.1 Transportation problem

For any given team i, we construct a bipartite graph $G_i = (U, V, E)$ as follows. We have a set of supply nodes U, containing a node with supply equal to 1 for each slot $s \in H_i$, i.e. the home team set of team i, and a node q with supply equal to $n-1$, corresponding to an unscheduled slot. The set of demand nodes V has a node with demand equal to 1 for each opponent of team i, i.e. $T \setminus \{i\}$, and a node corresponding to a dummy team. The demand of this last node is such that total supply equals total demand. Figure 1 represents an example of G_i.

The costs for each edge in E are set as follows. The costs on the edges between the dummy team node and any node in $U \setminus \{q\}$ are zero (dashed edges in Figure 1). The costs on the edges from node q to the non-dummy nodes are equal to P (dotted edges in Figure 1). Finally, an edge from a node $u \in U$ corresponding to a home slot $s \in H_i$ and a node $v \in V$ corresponding to a team $j \in T \setminus \{i\}$ has a cost that corresponds with inserting a home game of team i against j on time slot s in the current schedule (more details follow in Sections 5.2 and 5.3). If no matches involving team i or j have been scheduled so far, then the cost is zero. These edges correspond to the solid edges in Figure 1, and we will refer to them as such in the remainder of this text. Notice that the graph need not be complete: indeed, if the time slot corresponding to node u is in team j's forbidden game set, then there is no edge between u and j. Solving this transportation problem will schedule (or reschedule) the home games of team i; if flow is sent from node q to some opponent j, the home game of i against j is not scheduled. Notice that by construction, this problem is always feasible.

5.2 Construction phase

In the construction phase, we solve the transportation problem sequentially, for each team $i \in T$. The order of the teams is determined by H_i: we start with the team with the lowest number of available home game slots. The end result of the construction phase is a schedule (where some matches possibly remain unscheduled).

Initially, no matches have been scheduled, and hence the cost on the solid edges is zero. During the construction phase, the schedule will gradually be filled with matches, and the costs on the solid edges will increase accordingly. Indeed, the cost of an edge from a node $u \in U$ to a node $v \in V$, corresponding to scheduling the home game of team i against j on time slot s, will depend on the previous and next match of i, and the previous and next match of j in the preliminary schedule, with respect to time slot s. For instance, assume that a match of team b has been scheduled on day 21, and an away game of team i has been planned on day 18. In this case, the cost of the edge between day 19 and team b in Figure 1 is set to $p^1 + p^2$, which corresponds to the increase in objective function value of the mathematical formulation described in section 4 if a match between i and team b is inserted on day 19.

Furthermore, an edge from u to v, which corresponds to planning team i's home game against team j on time slot s is removed if at least one of the teams i and j already has a match scheduled on slot s, or if the match $j - i$ is already scheduled within m days of slot s.

Notice that there is in fact not a full correspondence between the objective function in section 4 and the costs as presented in this section. Indeed, when scheduling the home games of team i, we do not take into account costs related to scheduling two successive home games of i in less than R_{max} days (only away matches of team i are considered for this). Consequently, the total cost resulting from solving the transportation problem is an underestimation of the cost in the problem description section. In practice, however, this has little or no effect, since teams almost never specify two home game slots with less than R_{max} days in between (for the majority of the teams, the home ground is available on a fixed weekday, every other week).

5.3 Tabu phase

Tabu search is a heuristic search procedure which goes back to Glover [8] and has proven its value in countless applications. In our implementation, the tabu phase works with a tabu list of length 5, and is initially empty. We randomly pick a team i which is not in tabu list, and add it to the tabu list. Next we remove all the home games of this team from the current schedule, and solve the transportation problem for this team.

If the resulting schedule is better than previous schedule, we accept the new schedule and continue the tabu search phase with a new randomly picked team (different from i). In the other case, we impose changes to the home game assignment of team i. We do this by sequentially resolving the transportation problem, each time with a different solid edge that was part of the previous schedule removed from the graph. From these solutions we select the best one, and accept the resulting schedule. Notice that this schedule may be worse than the previous schedule. Also in this case, we continue the tabu search phase with a new randomly picked team (different from i).

5.4 Perturbation phase

In order to escape local optima, we apply the following perturbation of the current schedule if 500 iterations without improvement of the best schedule found so far occur. We randomly remove 10% of the scheduled matches, in order to open up some space in the schedule, and continue the tabu search phase.

6 Computational results

We solved the indoor football scheduling problem for all divisions for the seasons 2009–2010 till 2012–2013, which corresponds with 18 instances. Most divisions have 15 teams, although some division have 13 or 14 teams. The season is played from September 1st to May 31st, which results in $|S| = 273$ (leap years excepted). The home game set of a team has on average 4.4 slots more than what is needed for the league. However, in two instances, a team provided less home game slots than it has opponents, which inevitably leads to unscheduled matches. On average, teams ask not to play a match on 17 days. However, it turns out that 19 teams have a forbidden game set that contains more than the allowed 28 days. This is tolerated, since most of these teams also provide a home game set that largely exceeds the requirements. In the opinion of the league organizers, it suffices to have 3 days between two successive matches for a team (i.e. $R_{max} = 4$). The penalties were chosen as follows: $p^2 = 10$, $p^3 = 3$, $p^4 = 1$. We set $P = 1,000$ in order to maximize the number of scheduled matches, and to be able to clearly distinguish the contribution of unscheduled matches from matches in close succession in the objective function value.

We implemented the formulation provided in section 4 using IBM Ilog Cplex, version 12.2. Notice that constraints (10) and (11) can be relaxed by stating that all y and u variables should be between 0 and 1. Indeed, given

Table 1 Results for real-life instances from seasons 2009–2010 till 2012–2013

		IP formulation			Heuristic
Instance	Teams	Best solution	Lower bound	Comp. Time (s)	Best solution
1	14	0	0	18	0
2	14	16	0	5000	14
3	15	57	43	5000	2055
4	15	68	19	5000	2049
5	15	3000	3000	50	3000
6	15	27	0	5000	23
7	15	43	20	5000	55
8	15	2087	2067	5000	3086
9	13	0	0	9	0
10	14	21	0	5000	8
11	15	0	0	1976	0
12	15	50	12	5000	39
13	15	29	0	5000	11
14	15	7	0	5000	4
15	15	8	8	971	1006
16	15	113	60	5000	1065
17	15	60	9	5000	1052
18	15	2010	2000	5000	2006

the objective function, the integrality conditions on the x variables (9) are sufficient to ensure that the y and u variables are 0 or 1. All models were run on a Windows XP based system, with 2 Intel Core 2.8GHz processors, with a maximum computation time of 5000 seconds. The heuristic was implemented using C++ and run on the same machine, however with a maximal computation time of 500 seconds. The transportation problems were solved to optimality using an implementation of Kuhn-Munkres algorithm.

Table 1 presents our computational results. The first two columns provide the instance number and the number of teams in this division. The next two columns show the best found solution and lower bound found within the given computation time using the IP formulation. Only 5 of the 18 instances were solved to optimality; for other instances the maximal number of matches was scheduled, given team availabilities. The computation times in the fifth column indicate that if Cplex manages to find and prove optimality, this usually happens rather quickly. The final column shows the best found solution by our heuristic. In 4 cases, the heuristic approach found an optimal solution, however, for 6 instances, the heuristic failed to schedule the maximal number of matches. This is not as bad as it may appear in terms of objective function value, since in practice, a date for an unscheduled match is settled through negotiations under the responsibility of the home team. It is striking that in 7 cases, the heuristic resulted in a better solution than Cplex, despite being given 10 times less computation time.

7 Conclusions and future research

In this paper, we described and solved a sports scheduling problem for the LZV Cup, an amateur indoor football competition. This scheduling problem is interesting, because matches are not planned in rounds. Instead, each team has a number of time slots available to play its home games; away teams can specify days on which they are not able to play. Furthermore, successions of home or away matches are irrelevant. The goal is to balance each team's matches over the season, in the sense that there should be no close succession of matches involving the same team.

We developed an integer programming formulation and a heuristic approach, and used them to generate schedules for all divisions for the seasons 2009–2010 till 2012–2013. Overall, the performance of the tabu search based heuristic is comparable to that of Cplex, however, the reduced computation time, and the absence of expensive licenses make the heuristic implementation more suitable for (amateur) competitions such as the LZV Cup. These schedules were approved by the league organizers and have been implemented in practice, much to the satisfaction of the participating teams. In rare occasions where not all matches could be scheduled, the organizers appreciated that our approach makes it clear with which team to put the responsibility to find a solution.

Some future work remains. First of all, it would be interesting to integrate the planning of the cup competition in the scheduling process. Indeed, if the cup and the league are planned together instead of sequentially, an even better spread of the matches could be accomplished. Indeed, all teams take part in a cup competition, creating dependencies between the division schedules. Currently this matter is handled by scheduling the first round matches of the cup competition beforehand, simply by using the first time slot for which the home team has its venue available and which is not mentioned in the visitor's forbidden game set. If no such time slot exists between September 1st and October 15th, then we invert the home advantage. If this still does not result in a solution, we remove this cup match from the schedule, and leave it up to both opponents to find a suitable time slot themselves (e.g. by finding another venue). With this procedure, the divisions can be scheduled independently from each other. Whereas simultaneous scheduling of all divisions and the cup could clearly improve the quality of the schedules, a formulation like the one provided in section 4 may not be tractable in this case, even for advanced IP solvers. It could also be interesting to make an educated guess about which teams will survive the first and the following rounds in the cup, such that for these strong teams, we can leave gaps in their league schedule such that future cup matches can be fit in more easily. Finally, it would also be interesting to test the performance of the heuristic on a number of instances from other (amateur) indoor sports.

References

1. Armstrong, J. & Willis, R. (1993). Scheduling the cricket world cup - a case study, *Journal of the Operational Research Society* **44**: 1067–1072.
2. Bartsch, T., Drexl, A. & Kroger, S. (2006). Scheduling the professional soccer leagues of Austria and Germany, *Computers and Operations Research* **33**(7): 1907–1937.
3. Costa, D. (1995). An evolutionary tabu search algorithm and the NHL scheduling problem, *INFOR* **33**: 161–178.
4. Della Croce, F. & Oliveri, D. (2006). Scheduling the Italian Football League: an ILP-based approach, *Computers and Operations Research* **33**(7): 1963–1974.
5. Forrest, D. & Simmons, R. (2006). New issues in attendance demand: The case of the English football league, *Journal of Sports Economics* **7**(3): 247–266.
6. Goossens, D. & Spieksma, F. (2009). Scheduling the Belgian soccer league, *Interfaces* **39**(2): 109–118.
7. Goossens, D. & Spieksma, F. (2011). Soccer schedules in Europe: an overview, *Journal of Scheduling* **15**(5): 641–651.
8. Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* **13**(5): 533-549.
9. Kendall, G., Knust, S., Ribeiro, C. & Urrutia, S. (2010). Scheduling in sports: An annotated bibliography, *Computers and Operations Research* **37**: 1–19.
10. Knust, S. (2010). Scheduling non-professional table-tennis leagues, *European Journal of Operational Research* **200**: 358–367.
11. Nemhauser, G. & Trick, M. (1998). Scheduling a major college basketball conference, *Operations Research* **46**: 1–8.
12. Post, G. & Woeginger, G. (2006). Sports tournaments, homeaway assignments, and the break minimization problem, *Discrete Optimization* **3**: 165–173.
13. Schönberger, J., Mattfeld, D. & Kopfer, H. (2000). Automated timetable generation for rounds of a table-tennis league, *In: Zalzala, A. (Ed.), Proceedings of the IEEE Congress on Evolutionary Computation,* pp. 277–284.
14. Schönberger, J., Mattfeld, D. & Kopfer, H. (2004). Memetic algorithm timetabling for non-commercial sport leagues, *European Journal of Operational Research* **153**: 102–116.
15. Willis, R. & Terrill, B. (1994). Scheduling the australian state cricket season using simulated annealing, *Journal of the Operational Research Society* **45**: 276–280.

Asynchronous Island Model Genetic Algorithm for University Course Timetabling

Alfian Akbar Gozali

Telkom University

Tel.: +62-22-7564108

Fax.: +62-22-7562721

Email: alfian@telkomuniversity.ac.id

Jimmy Tirtawangsa

Telkom University

Tel.: +62-22-7564108

Fax.: +62-22-7562721

Email: tirtawangsa@yahoo.com

Thomas Anung Basuki

Telkom University

Tel.: +62-22-7564108

Fax.: +62-22-7562721

Email: tirtawangsa@yahoo.**com**

Abstract University course timetabling problem (UCTP) is similar to general timetabling problems with some additional unique parts. UCTP involves assigning lecture events to timeslots and rooms subject to a variety of hard and soft constraints. Telkom University has almost similar problem with its course timetabling. The current solution with Informed Genetic Algorithm for Telkom University UCTP still has the time consuming problem.

Island Model informed Genetic Algorithm was used in this research to solve this problem. The idea of this research is making distributed model exchanges an island's local best Individu with another island. Island model GA could create university course timetabling in reasonable time. This distributed model could run faster rather than single machine model decreasing constraint violations to reach optimum fitness. It could have less constraint violations because it could escape from stagnant local optimum easier. Island model GA could even produce great accuracy for Telkom University dataset (99.74%) and acceptable accuracy at 96.80% for Purdue dataset for student level timetabling.

Keywords *UCTP, informed genetic algorithm, island model genetic algorithm*

1 Introduction

University course timetabling problem (UCTP) is general timetabling problems with some additional unique parts [2]. One of the most recently studied for UCTP is the application of genetic algorithms (GAs), which are based on the theory of evolution [4], and that have proved to be efficient for problems of moderate and realistic size [5, 6, 7]. It is well known that fitness evaluation is the most time consuming part of the genetic programming (GAs) system. This limits the types of problems that may be addressed by GAs, as large numbers of fitness cases make GAs runs impractical.

Telkom University had almost similar problem with its course timetabling. University timetabling for Telkom University has been previously studied by Suyanto [9, 10] by implementing informed GA. The result was great, it can reduce student meeting violation down to 741 from 58,660 student meetings [10]. But it took until 3 days in practice.

Generic approach for university timetabling was done by Hana Rudová, et.al. [11]. They built a generic timetabling engine named UniTime [12]. However, it is not generic enough as several constraints required for Telkom University timetabling are not covered, for example “some special lecturers should be scheduled in their time constraints”, “lecturer meeting spread”, and “lecturers time preferences”. Moreover the result of UniTime may not be useful for Telkom University. When the Telkom University timetabling constraints are simplified to meet UniTime requirements, its resulting schedule still has high number of conflicts.

2 Telkom University Timetabling Problem

Telkom University has 6,570 students in 4 departments and 9 study programs. In one semester, there are 316 lecturers with 1,034 lecture meetings and 58,660 student meetings to schedule. It has 80 rooms categorized in 4 different capacities: extra-large (XL), large (L), medium (M) and small (S). There are 24 time slots per week, and have high occupancy of more than 78.11%.

Previously, there was a research attempt conducted by Suyanto [10]. He claimed that the most challenge in this case is that the courses are conducted in around 4 parallel classes in average and up to 27 parallel classes in maximum. It makes reducing student conflicts is complex.

3 Genetic Algorithm Model

Refer to the [10]; GA variant used in this research is Informed Genetic Algorithm (IGA). Directed mutation is also used for this research. This research used only mutation process but not mutation and crossover like Karol [16] did. Because crossover just scrambles the genes and does not make significant fitness improvement [9].

3.1 Fitness Function

Penalty determines the value of an interest. Higher penalty value means more important constraint. Fitness value can be calculated by the formula:

$$f = \sum_{i=1}^{N} (p_i * V_i) \tag{1}$$

Where N is the total number of events, p_i is declared value limits for penalty for all i and V_i is the number of violations that occurred on the i-th constraint. This fitness function is inverted fitness. Therefore, smaller fitness value shows better solution and bigger fitness value shows worse solution.

3.2 Hard and Soft Constraints

Hard constraints (HC) are constraints that must be met. While soft constraints (SC) have no restrictions to be complied with, but should be met in order to improve quality of the class schedule. This research used same constraints with [10] that have 12 constraints (5 HCs and 7 SCs) in total.

4 Island Model Genetic Algorithm

Architecture of island model GA used in this research consists of two types of islands: master and slave islands. Island model architecture used in this research described as below:

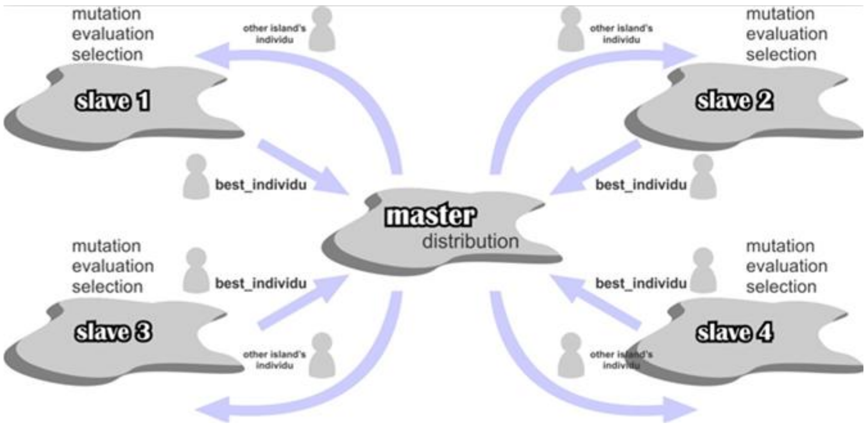


Figure 1. View of Asynchronous Island Model GA (Architecture Level)

Master Island works as a controller who distributes the given Individu to Slave Islands based on optimum fitness. Master Island can be attached to same computer with one of the slave islands because of its low resource consuming process. **Slave Islands** are the computational processor in the Island Model GA. It does mutation, fitness evaluation, and selection between iteration/generation processes. **Individual** or chromosome is a result representation of the GA process.

This Island Model will be run in asynchronous way. This means each island runs its own process independently. Process in one island is not directly depended on other island process. But at certain time, this island will take in another island's result to make better result. Figure 2 will explain the process of asynchronous island model used in this research.

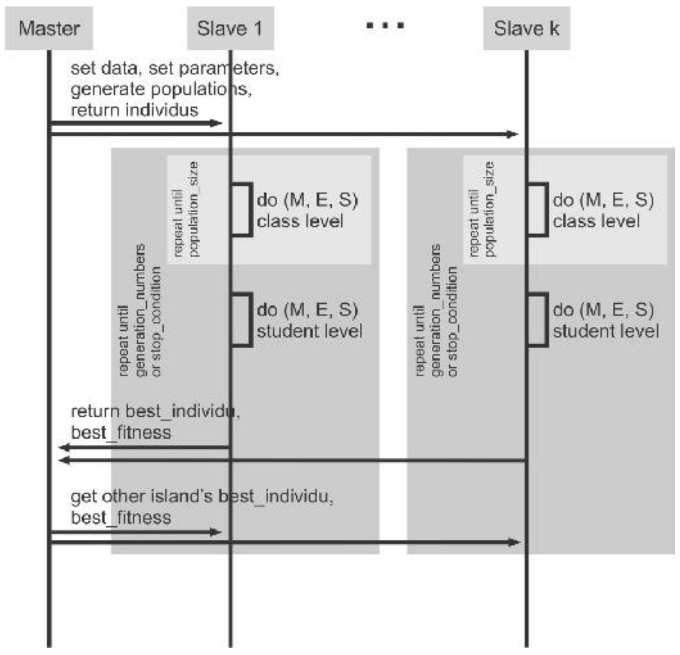

Master
Slave 1
...
Slave k
set data, set parameters,
generate populations,
return individus
repeat until population_size
do (M, E, S) class level
repeat until generation_numbers or stop_condition
do (M, E, S) student level
return best_individu,
best_fitness
get other island's best_individu,
best_fitness

Figure 2. View of Asynchronous Island Model GA (Process Level)

5 Constraints Mapping

One of these research objectives is to compare result of the current world best-known solution [12] with proposed Island Model GA for university course timetabling problem. It must be defined and mapped the format and constraint mapping from UniTime to Telkom University data format and vice versa.

5.1 Constraint Mapping from Island Model GA into UniTime

The constraint mapping from Island Model GA into Unitime constraint is listed in Table 2. Can be seen that there are some Island Model GA constraints cannot be mapped into UniTime constraints directly.

Table 1. Constraint Mapping Island Model GA to Unitime

Island Model GA	**UniTime**
No lecturer conflict	No lecturer conflict
No class conflict	No class conflict
Lecture suitable capacity room	Lecture suitable capacity room, SAME_ROOM
Lecturers time departments cons	Lecturers time departments cons
Some special lecturers should be scheduled in their time constraints	Not supported directly
Lecturer meeting spread	Not supported directly
Class meeting spread	SPREAD
Lecturers time preferences	Not supported directly
Time constraints between meetings of the same lectures	NHB_GTE, NHB_LT, NHB
Time constraints between different lecture meetings in the same group	NHB_GTE, NHB_LT, NHB
Minimizing student conflicts	Minimizing student conflicts

5.2 Constraint Mapping from Unitime GA into Island Model

The constraint mapping from Island Model GA into Unitime constraint is listed in Table 3. Can be seen that there are some Island Model GA constraints cannot be mapped into UniTime constraints directly:

Table 2. Constraint Mapping from Unitime GA into Island Model

UniTime	**Island Model GA**
SAME_ROOM (same room)	Supported
SAME_TIME (same time)	Supported
SAME_START (same start time)	Supported
SAME_DAYS (same days)	Supported
BTB_TIME (back-to-back time)	Not supported
BTB (back-to-back)	Not supported
NHB_GTE(1)	Not supported
NHB_LT(6)	Not supported
NHB(x) (x hr(s) between)	Not supported
DIFF_TIME (different time)	Not supported
SPREAD (time spread)	Supported

6 Results

Population/sampling used in this research is Telkom University 2011-12 odd semester schedule and Purdue University 2007 fall lecture large room [12]. The result of system performance testing scenario for Telkom University course timetabling with Island Model GA are shown in Table 5.

Table 3. Fitness comparison in scenario 1

Island numbers		**Execution time**	**Optimum fitness**	**Number of Violations (lecturer/class)**
Suyanto's[10]	1	0h 41m 5s	90000	38/142
Proposed scheme	2	0h 44m 34s	85000	32/133
	3	0h 45m 43s	82500	29/133
	4	0h 46m 29s	83000	30/130
	5	0h 48m 1s	79000	24/130

Furthermore, figure 3 compares single and multiple island performance by its time consumption for reaching single island's optimum fitness. Compared to single island model, multiple islands can reach single island's optimum value in around half of the single island's time consumption because single machine model possibility to trap into a local optimum.

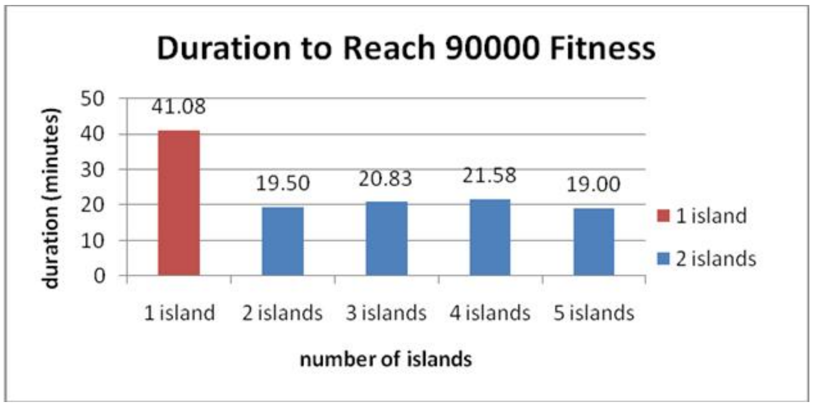


Figure 3. Duration Every Island to reach 90000 fitness

The result of system performance testing scenario for UniTime course timetabling with Island Model GA with one until five islands is shown in figure 4. There is a wide optimum (minimum) fitness gap between Purdue and Telkom University dataset. The reason behind this is because of both of them different characteristics.

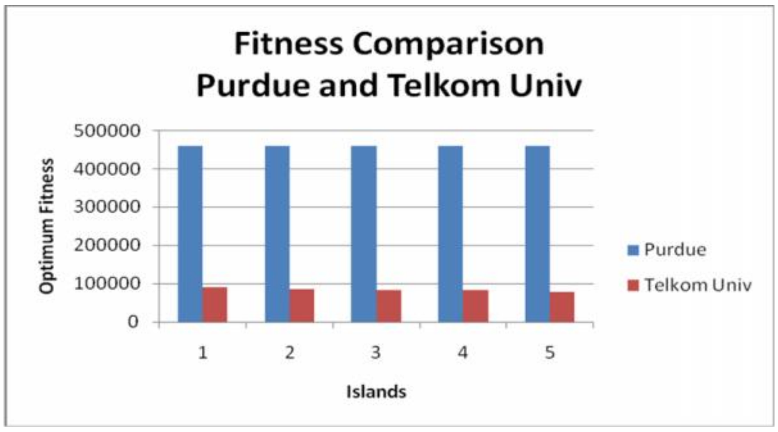


Figure 4. Fitness Comparison between Purdue and Telkom University Datasets

The comparison result of time consumption in same iteration (100 iterations) was shown in Table 9 below. UniTime and Island Model GA completes its running in just 16 minutes difference when applying Telkom University dataset. But when applying Purdue dataset, the difference can extremely widen the time gaps, more than 6 hours. Same with previous explanation, the reasons are in numbers of UniTime soft constraints and suitability of the engine.

Table 4. UniTime and Island Model GA Result Time Comparison

Engine	**Dataset**	**Time**	**Accuracy**
UniTime	Telkom University	0h 32m 1s	86.15%
Island GA	Telkom University	0h 48m 1s	99.74%
UniTime	Purdue	0h 33m 51s	80.43%
Island GA	Purdue	6h 38m 41s	96.80%

6 Conslusions

Island model GA could create university course timetabling in reasonable time. This distributed model could run faster rather than single machine model to decrease constraint violations to reach optimum fitness. It could have less constraint violations because it could escape from stagnant local optimum easier.

Island model GA could even solve another UCTP problem (Purdue University) but not quite well as Telkom University case. It produced great accuracy for Telkom University dataset (99.74%) and acceptable accuracy at 96.80% for Purdue dataset for student level timetabling.

Characteristics of datasets significantly influence the result of timetabling creating process. The main influencing characteristics are varieties and numbers of the soft constraints. And the most efficient number of island for Telkom University dataset is five islands.

References

[1] D. Johnson and M. Garey, Computers and intractability: A guide to the theory of NP-completeness, San Francisco: W.H. Freeman, 1979.

[2] R. Lewis, "A survey of metaheuristic-based techniques for university timetabling problems," *OR Spectrum*, vol. 30, no. 1, p. 167–190, 2007.

[3] S. Abdullah, E. K. Burke and B. Mccollum, "A Hybrid Evolutionary Approach to the University Course Timetabling Problem," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, Singapore, 2007.

[4] D. E. Goldberg, Genetic Algorithm in Search, Optimization, and Machine Learning, Boston: Addison Wesley, 1989.

[5] J. Abela and D. Abramson, ""A Parallel Genetic Algorithm for Solving the School Timetabling Problem," CSIRO, Clayton South, 1991.

[6] E. K. Burke, J. P. Newall and R. F. Weare, "A Memetic Algorithm for University Exam Timetabling," *Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science,* vol. 1153, no. 1, pp. 241-250 , 1996.

[7] D. Corne, P. Ross and H. L. Fang, "Fast Practical Evolutionary Timetabling," in *AISB Workshop on Evolotionary Computation*, Leeds, 1994.

[8] R. Megasari, An Optimization of University Course Timetabling Using Case Based Reasoning and Graph Coloring (Master Thesis), Bandung: Institut Teknologi Telkom, 2011.

[9] Suyanto, Algoritma Optimasi: Deterministik atau Probabilitik, Yogyakarta: Graha Ilmu, 2010.

[10] Suyanto, "An Informed Genetic Algorithm for University Course and Student Timetabling Problems," *Artifical Intelligence and Soft Computing Lecture Notes in Computer Science,* vol. 6114, no. 1, pp. 229-236, 2010.

[11] H. Rudova, T. Muller and K. Murray, "Complex university course timetabling," *Journal of Scheduling,* vol. 14, no. 2, pp. 187-297, 2011.

[12] T. Muller, "unitime," UniTime LLC, 13 September 2012. [Online]. Available: www.unitime.org. [Accessed 26 December 2012].

Partially-Concurrent Open Shop Scheduling

Tal Grinshpoun · Hagai Ilani · Elad Shufan

Abstract The *partially-concurrent* open shop scheduling problem is presented. The standard open shop scheduling problem is generalized by allowing some operations to be processed concurrently. This generalization is directly motivated from a real-life timetabling project of assigning technicians to airplanes in an airplane garage. A schedule for the partially-concurrent problem is represented by a digraph. We show that the scheduling problem is equivalent to a problem of orienting a given undirected graph, called a conflict graph. The schedule digraph is then modeled by a matrix, generalizing the rank matrix representation. The problem is shown to be NP-Hard. The rank matrix representation is also used in an algorithm that heuristically constructs an open shop schedule.

Keywords Open Shop Scheduling · Concurrent machines · Technician timetabling · Graph orientation

1 Introduction

An open shop scheduling (OSS) problem consists of n jobs that should be processed on m machines. An operation (i, j) refers to the processing of job $i = 1, 2, \dots, n$ in machine $j = 1, 2, \dots, m$. The processing time of operation

Tal Grinshpoun
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: talgr@ariel.ac.il

Hagai Ilani
Department of Industrial Engineering and Management, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Elad Shufan
Physics Department, SCE – Shamoon College of Engineering, Beer-Sheva, Israel
E-mail: elads@sce.ac.il

(i, j) is denoted by p_{ij}. In a standard OSS every job visits one machine at a time, and every machine hosts only one job at a time.

Shop scheduling problems were originally designed for machines. Nevertheless, there are many task scheduling applications in which employees are represented as the machines; hence the resulting solution is a set of timetables (schedules) for the employees.

An important aspect of OSS concerns the mathematical representation of a given schedule. Bräsel and Kleinau have introduced the rank matrix representation [7,5], which is significant for both theory and practice. A major advantage of the rank matrix representation when compared to its alternative [16] is the one-to-one correspondence between a matrix and a schedule, provided that the schedule is semi-active [18], for which operations are performed as early as possible once the order of processing is known. Constructive heuristic algorithms were suggested, based on rank matrices [10]. Rank matrix procedures were applied to neighbourhood definitions in local search algorithms, including those of crossover and mutations in genetic algorithms [1]. Among the theoretical achievements that are based on the idea of rank matrix are the irreducibility theory [2,8] as well as complexity analysis of some special OSS problems [9].

In this work we extend the rank matrix representation to a problem of *partially-concurrent* open shop scheduling (PCOSS). In a PCOSS problem some operations are allowed to be processed concurrently, while some are not. In the PCOSS presented herein preemption is not allowed, and no due or release dates are given. A variety of previously discussed issues can be extended by the more general PCOSS matrix representation. Several such extensions are discussed in this article.

A related problem previously presented in the literature is that of *concurrent* open shop scheduling [21,17,15]. It is also referred to as a parallel machine environment with m fully dedicated machines [14], denoted PDm. In a PDm, a job can be split to be simultaneously processed on several machines. The two extremes of PCOSS are the standard OSS, where operations of the same job are never processed concurrently, and the fully-concurrent open problem, where all the operations of a given job are allowed to be processed concurrently. The PDm notation does not reflect the connection between the concurrent open shop and the standard one. We suggest that for all the discussed types of an OSS, the machine environment field α of Graham's $\alpha|\beta|\gamma$ classification scheme [12] will include O (or Om when the number of machines should be specified explicitly). Concurrency issues will be included in the β field, with the "*conc*" entry for the concurrent open shop case, and "*pconc*" when a partially concurrent open shop is considered.

Roemer discusses the complexity of $O|conc|\sum w_i C_i$ [19], where C_i is the completion time of job i, and w_i the corresponding weight. It is shown that the general problem is NP-hard in the strong sense. Minimizing the makespan $C_{max} = max\{C_i \mid 1 \leq i \leq n\}$, which is commonly considered in a standard OSS, is not considered in the concurrent version. This is due to its over simplicity: any semi-active schedule will be optimal with $C_{max} =$

$max\{\sum_{k=1}^{n} p_{kj} \mid 1 \leq j \leq m\}$. For the general PCOSS problem the complexity of a schedule designed to minimize the makespan should be discussed, with $O|conc|C_{max}$ being trivially polynomial, and a general $O||C_{max}$ being NP-Hard.

The PCOSS problem is more general than the standard OSS and can therefore describe a large variety of real-life scenarios. In fact, the present study is directly motivated from a timetabling project of assigning technicians to airplanes in an airplane garage. The airplane garage task scheduling problem is mentioned in the literature with respect to both the standard [8] and the concurrent [21] OSS versions. Consider a fleet of airplanes. A set of tasks should be performed on each plane in order to prepare it for action. Every task is done by a technician who has the expertise to do this task alone. In reality, some tasks can be performed simultaneously on a plane, e.g., while one technician checks the engine, another technician can check the wing. But other tasks exist that disturb each other, and therefore cannot be performed concurrently, which is similar to the standard open shop. Indeed, in the timetabling project that inspired the present research, some of the technicians' tasks could be performed in parallel. The $O|pconc|f_{obj}$, with any objective function f_{obj}, naturally describes this scenario, with airplanes corresponding to jobs, and tasks (or technicians) corresponding to machines.

In section 2 the generalization of the rank matrix representation is given. Section 3 deals with the complexity of PCOSS. A constructive algorithm, first suggested with respect to the standard OSS [10], is extended to the PCOSS problem in section 4, followed by experiments (section 5). A discussion is given in section 6.

2 Matrix representation of Partially-Concurrent Open Shop Scheduling

The standard OSS is reviewed first. A scheduling problem consists of a set of n jobs J_i, where $i \in I = \{1, 2, \ldots, n\}$, which should be processed on a set of m machines M_j, $j \in J = \{1, 2, \ldots, m\}$. In addition, the processing times of the operations are listed as elements of an $n \times m$ matrix $PT = [p_{ij}]$, with p_{ij} denoting the processing time of an operation (i, j). In a general scenario the jobs might visit only a partial set of machines. For clarity of presentation we assume that every job visits all the machines, i.e., $p_{ij} > 0$ for every $i \in I$ and $j \in J$. In the discussion (Section 6) we explain how to treat the general scenario. In a non-concurrent OSS two operations of the same job cannot be processed concurrently. Therefore, a given schedule necessarily defines an order between operations of the same job (machine order). Similarly, a schedule sets an order between operations of the same machine (job order). Operations that do not share either a job or a machine can obviously be processed concurrently. Therefore, the schedule does not imply any order between such operations. This is illustrated in the following OSS example.

Example 1 In this example we consider $n = 3$ jobs and $m = 4$ machines, and the following processing times matrix:

$$PT = \begin{pmatrix} 23 & 21 & 40 & 6 \\ 15 & 30 & 18 & 35 \\ 28 & 8 & 25 & 24 \end{pmatrix} \tag{1}$$

A possible schedule for this instance is given in Figure 1(a) by its Gantt chart. Jobs J_1, J_2, and J_3 are recognized by their colours: dark red, white, and light green, respectively. In the given schedule the machine order of Job 1, taken as an example, is $M_4 \rightarrow M_2 \rightarrow M_1 \rightarrow M_3$. For machine 1, the job order is $J_2 \rightarrow J_1 \rightarrow J_3$. Both orders are easily read from the Gantt chart for this compact example.

The machine and job orders can be represented by a non-cyclic digraph called a *sequence graph* [7,6] – every operation is represented by a vertex (i, j) and each pair of consecutive operations (vertices) are connected by an arrow. The sequence graph that corresponds to the given schedule of example 1 is shown in Figure 1(b).

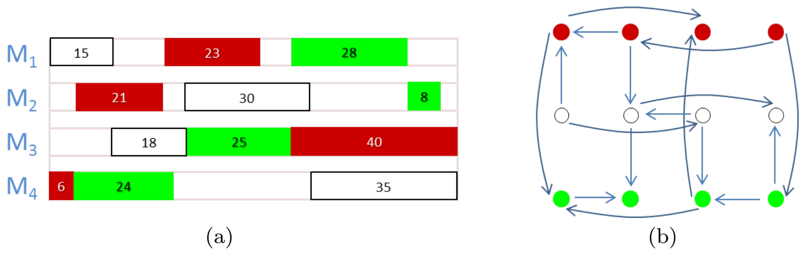


Fig. 1 A possible schedule for an OSS with 3 jobs and 4 machines is given by (a) a machine-oriented Gantt chart, and (b) a sequence graph.

A convenient matrix representation for sequence graphs was suggested [7, 5]. The matrix, termed a rank matrix, is denoted by R. An entry r_{ij} is equal to the number of vertices in the longest path from a source to the vertex (i, j). A source is a vertex with zero indegree. It represents an operation that is scheduled first. The rank matrix of the schedule given in Figure 1 is

$$R = \begin{pmatrix} 3 & 2 & 4 & 1 \\ 1 & 3 & 2 & 4 \\ 4 & 5 & 3 & 2 \end{pmatrix} \tag{2}$$

The machine order or job order are both represented by the order of the entries of the relevant row or column, respectively. If $r_{ij} > 1$, then in row i or column j there exists an entry that equals $(r_{ij} - 1)$. Operation (i, j) is scheduled after the operation that corresponds to this entry. According to the "Latin-rectangle theorem" [7], there is a one-to-one correspondence between a unique Latin rectangle and a semi-active schedule: Consider a matrix of size $n \times m$ whose elements are taken from the set $\{1, 2, \ldots, q\}$. If in each row and each column no entry appears more than once, the matrix is called a Latin rectangle, denoted $LR(n, m, q) = [lr_{ij}]$. The Latin rectangle $LR(n, m, q)$ corresponds to a schedule of an open shop with n jobs and m machines, if for every entry $lr_{ij} \neq 1$ there exists an entry $(lr_{ij} - 1)$ in either row i or column j.

In this article we show that the above ideas can be naturally extended to the case of PCOSS. An instance of a PCOSS contains, in addition to PT, a concurrence graph, or its complement, a conflict graph. These graphs describe whether pairs of operations may be processed concurrently or not. More precisely, the concurrence graph, CG, has $I \times J$ as its vertex set, and two operations (i, j) and (k, l) are adjacent if they may be processed concurrently. In the conflict graph, $\overline{CG}$, (i, j) and (k, l) are adjacent if they may not be processed concurrently.

To summarize, for any pair of operations:

– if they relate to different machines and different jobs the pair is an edge of the concurrence graph.
– if they relate to the same machine and different jobs the pair is an edge of the conflict graph.
– if they relate to different machines and to the same job the pair can be either an edge of the concurrence graph or the conflict graph, depending on whether the pair can be processed concurrently or not (respectively). A pair of operations that is an edge of the conflict graph will be called a *conflict pair*. Given an operation (i, j), we say that operation (k, l) is a *conflicting operation* if the pair $\{(i, j), (k, l)\}$ is a conflict pair.

In Figure 2 we show an example of CG and $\overline{CG}$ for 3 jobs and 4 machines. For clarity, arrows connecting two operations that do not share a machine or a job are not shown in CG.

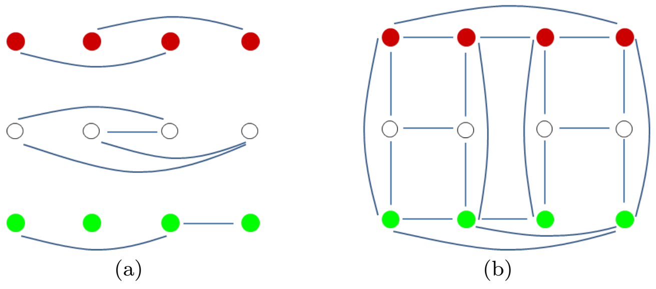


Fig. 2 An instance of a PCOSS includes (a) a concurrence graph (only horizontal edges are shown) or (b) its corresponding conflict graph.

A schedule for the PCOSS is given by the $n \times m$ matrix $ST = [st_{ij}]$, of the starting times of all the operations, or by the matrix $CT = [ct_{ij}]$, of the completion times of all the operations. Obviously, $CT = ST + PT$. A schedule is *feasible* if for any conflict pair, $\{(i,j),(k,l)\}$, either $ct_{ij} \leq st_{kl}$ or $ct_{kl} \leq st_{ij}$. A given feasible schedule naturally defines an orientation of the conflict graph: an edge $\{(i,j),(k,l)\}$ will be oriented, $(i,j) \rightarrow (k,l)$, if $ct_{ij} \leq st_{kl}$ and $(k,l) \rightarrow (i,j)$, if $ct_{kl} \leq st_{ij}$. The resulting digraph, DG, is acyclic, because a cycle $(i,j) \rightarrow (k,l) \rightarrow \ldots \rightarrow (i,j)$ in DG means $ct_{ij} \leq st_{kl} < ct_{kl} \leq \ldots \leq st_{ij}$, which is impossible – an operation cannot be completed before it starts.

An acyclic orientation of the conflict graph will be called a *partial-sequence graph.* It generalizes the sequence graph representation.

Lemma 1 *A partial-sequence graph defines a schedule with the property that $\{(i,j),(k,l)\}$ implies $ct_{ij} \leq st_{kl}$. The schedule defined is unique, assuming semi-activeness.*

Proof Given a partial-sequence graph, DG, a schedule is defined inductively step by step as follows: For each stage i we define A_i to be the set of operations that have already been scheduled up to stage i. Initially $A_0 = \emptyset$. In the first stage ($i = 1$), we schedule at $t = 0$ all the operations that have indegree 0. Because DG is acyclic, at least one operation with indegree 0 exists. At stage i we schedule all the operations that have indegree 0 in $DG \backslash A_{i-1}$. An operation (i,j) will start at $st_{ij} = max\{ct_{kl} | (k,l) \rightarrow (i,j)\}$. The uniqueness is forced by semi-activeness; i.e., each operation is scheduled as early as possible while keeping the order defined by DG. ■

Corollary 1 *Solving $O|pconc|C_{max}$ is equivalent to the problem of orienting the edges of the conflict graph so that the digraph obtained will be acyclic and the maximally-weighted path will be minimal.*

The weight of a path is the sum of all the processing times of operations (vertices) in that path.

Example 2 A PCOSS instance is composed of a matrix PT and a graph CG, or $\overline{CG}$. We take PT of Example 1 and $\overline{CG}$ of Figure 2(b). A possible schedule for this PCOSS instance is shown in Figure 3 by its Gantt chart and the corresponding orientation of the conflict graph, i.e., the partial-sequence graph DG.

The rank matrix representation follows. We define the rank r_{ij} of an operation (i,j) as the number of vertices in a longest path in DG from a source vertex to the operation (i,j). The rank matrix is denoted $R = [r_{ij}]$.

The rank matrix of Example 2 is

$$R = \begin{pmatrix} 3 & 2 & 4 & 1 \\ 1 & 3 & 1 & 3 \\ 4 & 1 & 3 & 2 \end{pmatrix} \tag{3}$$

The rank matrix R has the following two properties:

1. $r_{ij} \neq r_{kl}$ for each conflict pair (i,j) and (k,l).

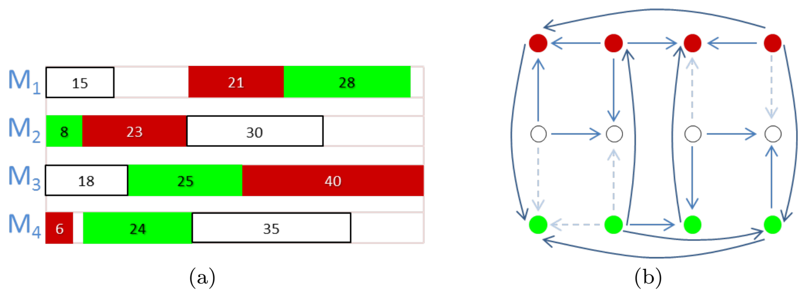


Fig. 3 A possible PCOSS schedule is given by its (a) Gantt chart, or its corresponding (b) partial-sequence graph. Dashed arcs can be removed due to transitive closure.

2. For any operation (i, j) with $r_{ij} > 1$ there exists a conflicting operation (k, l) with $r_{kl} = r_{ij} - 1$.

Given a concurrence graph for a PCOSS instance, we call a positive integer-valued matrix $n \times m$ a *CG-rectangle* if it satisfies conditions 1 and 2. In the case of a standard OSS the concurrence graph has only edges connecting operations if they belong to different jobs and different machines. The CG-rectangle then reduces to a Latin-rectangle. The following theorem generalizes the "Latin-rectangle theorem" [7].

Theorem 1 *An $n \times m$ matrix $A = [a_{ij}]$ is a CG-rectangle iff it is a rank matrix of a semi-active schedule for a PCOSS problem with n jobs and m machines.*

Proof Given a semi-active schedule for a PCOSS problem, a rank matrix is constructed as defined previously. On the other hand, given a CG-rectangle, A, we construct an appropriate partial-sequence graph by orienting the edges of the conflict graph as follows: an edge $\{(i, j), (k, l)\}$ will be oriented $(i, j) \to (k, l)$ if $a_{ij} < a_{kl}$ and $(k, l) \to (i, j)$ if $a_{kl} < a_{ij}$. The obtained graph is obviously acyclic. The theorem's assertion then follows from Lemma 1. ■

3 Complexity issues

Concerning the complexity of $O|pconc|C_{max}$, it is proved next that the problem with only one job and unitary processing times, denoted $O|pconc, n = 1, p_{ij} = 1|C_{max}$, is already NP-Hard. It immediately follows that a general PCOSS is also NP-Hard. It is worth noting that with one job, minimizing the makespan in both the standard OSS problem and the concurrent open shop is a trivial task. For $O|n = 1|C_{max}$, any order of the operations leads to a schedule with $C_{max} = \sum_{j=1}^{m} p_{1j}$. For $O|conc, n = 1|C_{max}$, any order of the operations leads to a schedule with $C_{max} = max\{p_{1j}|1 \leq j \leq m\}$. Moreover, for the standard open shop problem with unitary processing times, Bräsel et al. developed a polynomial-time algorithm for minimizing the makespan for *any number* of jobs [9].

Theorem 2 *The problem $O|pconc, n = 1, p_{ij} = 1|C_{max}$ is NP-Hard.*

Proof In the case where there is only one job and all processing times equal 1, the makespan of any schedule S is the length of the longest path in the sequence graph defined by S, which is the maximum rank in the matrix defined by S. By Corollary 1 the opposite is also true – the length of the longest path in any acyclic orientation of the conflict graph is the makespan of a feasible schedule for the problem. Because the conflict graph can be any arbitrary undirected graph on the set of the n operations, it follows that $O|pconc, n = 1, p_{ij} = 1|C_{max}$ is equivalent to the problem of orienting an undirected graph in order to minimize the size of the longest directed path. The latter is polynomially equivalent to the Graph-Colouring problem by the proof of the Gallai-Roy-Vitaver theorem [3], which asserts that the minimal size of the longest directed path in an orientation of an undirected graph G is equal to the chromatic number of G. Because the Graph-Colouring problem is NP-Hard it follows that $O|pconc, n = 1, p_{ij} = 1|C_{max}$ is NP-Hard. ■

Corollary 2 *The general problem $O|pconc|C_{max}$ is NP-Hard.*

4 Constructive heuristic

The constructive heuristic considered in this section is an adaptation to the partially-concurrent case of the insertion algorithm that was proposed for standard OSS [10]. The algorithm builds a full schedule (rank matrix) in an iterative manner. At each iteration, the algorithm *inserts* one operation into a partial schedule, until a full schedule is reached. The order at which the operations are inserted is determined before the iterative process commences.

Following [10], the operation insertions are combined with *beam search.* That is, only a limited number of solution paths within the complete search space are investigated. During the search process, each partial schedule has one *parent* and several *children.* The parent is the partial schedule, excluding the last inserted operation. The children result from the insertion of the next operation. The number of parallel solution paths is limited by the beamwidth p.

We consider the following possibilities for the insertion of an operation (i, j) into the partial rank matrix R:

1. $r_{ij} = 1$. This means that i is the first job in the job order on machine j and j is (one of) the first machine(s) in the machine order of job i.
2. $r_{ij} = r_{kj} + 1$, where r_{kj} is any of the values that appear in column j of R. This means that operation (i, j) becomes a direct successor of one of the operations on machine j.
3. $r_{ij} = r_{il} + 1$, where r_{il} is any of the values that appear in row i of R that correspond to an operation of J_i, which is in conflict with operation (i, j). This means that operation (i, j) becomes a direct successor of one of the operations of job i, with which it has conflict.

These possibilities correspond to the cases (c1) and (c2) in the original insertion algorithm [10].

An insertion of r_{ij} to R results in a new (partial or full) matrix R'. However, R' may possibly not be a CG-rectangle. This happens when one or more of the following conflicts occur: (I) $\exists k \neq i \bullet [r_{ij} = r_{kj}]$; (II) $\exists l \neq j \bullet [r_{ij} = r_{il}$ and (i,j) conflicts with $(i,l)]$. Note that there may possibly be several instances of conflict (II) due to concurrent machines. All the conflicts are resolved by incrementing by 1 all the conflicting entries. The entries with incremented values may in turn be in conflict with a new set of entries. This process continues until all conflicts in R' are resolved.

The set of all obtained R' matrices (each corresponding to a possible value of r_{ij}) forms the list of children. In each iteration of the beam search, a list of the most promising p children should be selected. Following [10], we consider two variants:

- **INSERT1:** In each iteration we select the p-best children from the whole set. This means that some selected children may have the same parent.
- **INSERT2:** For each of the p parents we select the best child. This means that all children have different parents. The first variant is applied as long as we do not have p parents.

We still must decide which children are considered the best in each step. Similarly to [10], we assign to each child the cost of the longest path (cost-wise) that goes through the newly inserted operation (i,j) in the relevant R' matrix. A path in PCOSS is a series of adjacent operations in the conflict graph $\overline{CG}$. The children that are considered best are those with lowest costs. A more detailed description of the insertion algorithm can be found in [10].

5 Experimental evaluation

The objective of the experimental evaluation of the present paper is twofold. Naturally, one would like to evaluate the effectiveness of the proposed constructive heuristic in terms of the quality of the obtained solutions. Nevertheless, perhaps even more interesting is the question of how partial concurrency affects the problems themselves, regardless of the chosen solution method.

In an attempt to shed light on these two issues, we turned to the commonly used OSS problem instances that were proposed by Taillard [20]. We used the entire set of Taillard's OSS benchmark that consists of six problem sizes (4×4, 5×5, 7×7, 10×10, 15×15, 20×20), with 10 instances of each problem size. For each of these 60 standard OSS problems we created 90 new PCOSS instances with varying concurrency levels, where the concurrency level of a problem relates to the percentage of non-conflicting operation pairs out of all operation pairs sharing the same job. We used concurrency levels with 10% intervals, varying from 10% to 90%. For each concurrency level we generated 10 problem instances, each with a randomly chosen set of non-conflicting operation pairs. For each problem in Taillard's benchmark, we also considered the two PCOSS extremes, which are the original (standard OSS) problem (0% concurrency) and the fully-concurrent problem (100% concurrency).

Following [10], we consider three versions of the constructive heuristic: one with beamwidth $p = 1$, and the two INSERT variants with beamwidth $p = 2$. In order to evaluate the quality of the obtained solutions we relate to the deviation of the corresponding C_{max} from the machine lower bound. For each problem, the machine lower bound is the maximal working time of any of the machines, given by $max\{\sum_i p_{ij} | 1 \leq j \leq m\}$.

Figure 4 presents the quality of obtained solutions for the PCOSS problems of size 4×4. The results shown for each concurrency level are the average deviations of all the PCOSS problems of that level. The results in Figure 4 reveal interesting information regarding both issues of the evaluation's objective. First, the obtained solutions of the heuristic are clearly of high quality, even when using a very low beamwidth. These findings coincide with the respective results for standard OSS [10][1]. Second, the graph shows that as the concurrency level increases the optimal C_{max} gets closer to the machine lower bound. This is not surprising, since more concurrency brings with it more efficient scheduling possibilities that in turn reduce C_{max} towards the bound set by the machines.

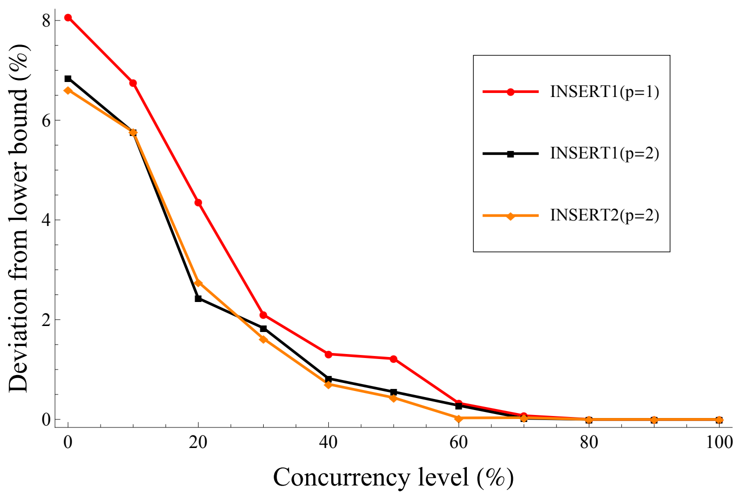


Fig. 4 The average deviation of C_{max} from the machine lower bound as a function of the concurrency level for problems of size 4×4.

For high levels of concurrency the heuristic algorithm gave the optimal solution. Therefore, in order to evaluate the scalability of the constructive heuristic we focus on a low concurrency level (10%). Figure 5 presents the quality of obtained solutions for PCOSS problems of different sizes. The displayed results

[1] The results for 0% concurrency are worse than those presented in [10], since the machine lower bound used herein is looser than the (machine and job) lower bound used in [10].

are the average deviations of 100 PCOSS problems of each problem size, with the size ranging from 4×4 to 20×20 according to the Taillard benchmark. Again, the obtained solutions of the heuristic are of very high quality. The results also indicate that as the problems get larger the optimal C_{max} gets closer to the lower bound. A similar phenomenon was observed for standard OSS problems [10].

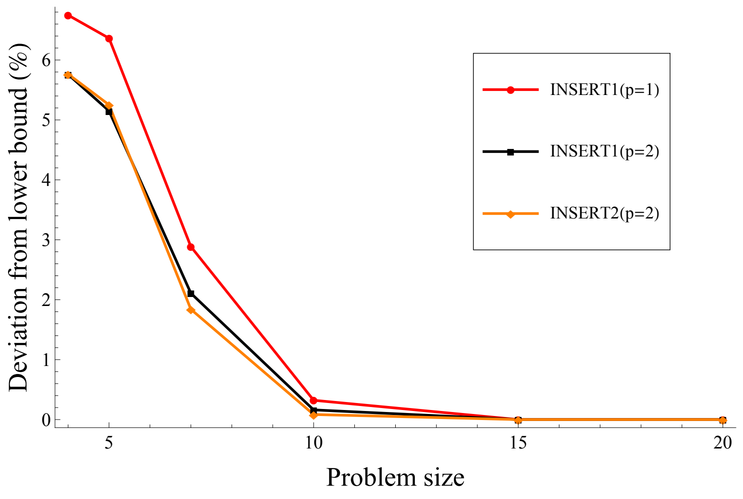


Fig. 5 The average deviation of C_{max} from the machine lower bound as a function of the problem size $(n \times n)$ for problems with 10% concurrency level.

Another aspect is the runtime of the constructive heuristic. The advances in hardware capabilities enable us to run the above experiments in reasonable time. In fact, even the hardest considered PCOSS problems (size 20×20) were solved in less than half a second with beamwidth $p = 2$ on a hardware comprised of Intel i5 4th generation and 8GB memory.

6 Discussion

The presented PCOSS is a general open shop problem, connecting two previously discussed scheduling problems – that of the standard open shop and its concurrent version. PCOSS enables natural representation of various realistic problems, such as that of the airplane garage that was mentioned in the introduction section. By extending the rank matrix scheme to the more general scenario of PCOSS, one may utilize efficient solving techniques, such as the presented constructive heuristic.

Investigating this general problem has highlighted the fact that generating an open shop schedule (whether concurrent or not) is equivalent to orienting a conflict graph, i.e., generating an acyclic digraph DG. The digraphs formally presented in the literature with respect to the non-concurrent OSS are basically transitive reductions of the partial-sequence graphs presented herein. Existing studies on the topic of generating acyclic orientations of a given undirected graph [4] can shed light on problems of OSS.

In this article we assumed that jobs in a PCOSS can be split to be processed on several machines simultaneously, but each machine hosts only one job at a time. Consequently, all the vertical edges existed in the conflict graph. Our proposed formalism enables the removal of this limitation, allowing also some operations of a given machine to be processed concurrently. The obtained rank matrix might then have several entries on a given column that are the same, as long as the corresponding operations are not conflicting.

We have also assumed that $p_{ij} > 0$ for all $i \in I$ and $j \in J$, i.e., that all the vertices $I \times J$ represent real operations, with each job visiting all the machines. In real life it often happens that some jobs visit only a partial set of machines. It is possible to include these scenarios without changing the given formalism: $I \times J$ vertices are considered as proposed before. A non-processing operation, i.e., any operation of a job i that does not visit machine j, has zero processing time. A vertex (i, j), which represent a non-processing operation has concurrent edges to all the other vertices, i.e., in the conflict graph it is not connected to any other vertex. The rank of non-processing operations will then be $r_{ij} = 1$, because it has 0 indegree and is therefore, by definition, a source. Note that this procedure can lead to several 1's in a given column. Nevertheless, the rank matrix remains a CG-rectangle.

PCOSS was shown to be NP-Hard. Yet, the heuristic algorithm proposed in Section 4 reaches the optimum in many instances, even when using a narrow beam. The examined instances are that of Taillard, with varying concurrency levels. For these instances, increasing the concurrency level resulted in obtaining C_{max} values that are very close to the machine lower bound. These results can be misleading, suggesting that increasing the concurrency level makes the problem easier to solve. Taillard's benchmark is a standard for non-concurrent OSS, for which the most difficult problems are those of square PT, with $n = m$. Contrary to that, the possibility of processing a given job in several machines concurrently suggests that it might be harder to solve problems with $m > n$. The logic behind this assertion is demonstrated by considering a uniform $n \times n$ OSS instance (uniform in the sense that $p_{ij} = 1$ for all i and j). This is an easy OSS problem – a schedule represented by any Latin rectangle $LR(n, n, n)$ is optimal with $C_{max} = n$. A uniform OSS problem with $m > n$ is still easy – an optimal schedule is given by $LR(n, m, m)$, with $C_{max} = m$. However, in uniform PCOSS problems things are more complicated. The squared problem remains easy, because C_{max} cannot be lower than the machine lower bound. Nevertheless, a uniform PCOSS with $m > n$ is completely non-trivial. Even obtaining an appropriate job lower bound is NP-Hard, being equivalent to the maximum independent set problem.

Further generalizations can be achieved considering PCOSS and its representation. For example, it is possible to extend the definition of a reducible sequence given in irreducibility theory [2,8]. The known benchmarks should be extended to include PCOSS instances that describe the schedule difficulties more appropriately. We leave these interesting issues for future research.

Another interesting direction for future work is to model and solve the most general scenario of an airplane garage, which was the initial motivation for the present study. In the general real-life problem the airplanes are located in several hangars, and some of the tasks need teams of technicians. We plan to additionally generalize the OSS model to include issues of technicians' transportation and teaming, which were previously referred to in *Workforce Scheduling and Routing Problems* [11]. Additionally, some real-life scenarios are *multi-mode* (cf. [13]), i.e., some tasks may be performed in several ways (modes) using different amounts of resources (technicians). The adaption of PCOSS to enable multiple modes is another challenging prospect.

References

1. Andresen, M., Bräsel, H., Mörig, M., Tusch, J., Werner, F., Willenius, P.: Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. Mathematical and Computer Modelling **48**(7), 1279–1293 (2008)
2. Andresen, M., Dhamala, T.N.: New algorithms and complexity status of the reducibility problem of sequences in open shop scheduling minimizing the makespan. Annals of Operations Research **196**(1), 1–26 (2012)
3. Bang-Jensen, J., Gutin, G.: Theory, algorithms and applications. Springer Monographs in Mathematics, Springer-Verlag London Ltd., London (2007)
4. Barbosa, V.C., Szwarcfiter, J.L.: Generating all the acyclic orientations of an undirected graph. Information Processing Letters **72**(1), 71–74 (1999)
5. Bräsel, H.: Matrices in shop scheduling problems. In: Perspectives on Operations Research, pp. 17–41. Springer (2006)
6. Brasel, H., Harborth, M., Willenius, P.: Isomorphism for digraphs and sequences of shop scheduling problems. Journal of combinatorial mathematics and combinatorial computing **37**, 115–128 (2001)
7. Bräsel, H., Kleinau, M.: On the number of feasible schedules of the open-shop-problem-an application of special latin rectangles. Optimization **23**(3), 251–260 (1992)
8. Bräsel, H., Kleinau, M.: New steps in the amazing world of sequences and schedules. Mathematical methods of operations research **43**(2), 195–214 (1996)
9. Bräsel, H., Kluge, D., Werner, F.: A polynomial algorithm for an open shop problem with unit processing times and tree constraints. Discrete Applied Mathematics **59**(1), 11–21 (1995)
10. Bräsel, H., Tautenhahn, T., Werner, F.: Constructive heuristic algorithms for the open shop problem. Computing **51**(2), 95–110 (1993)
11. Castillo-Salazar, J.A., Landa-Silva, D., Qu, R.: A survey on workforce scheduling and routing problems. In: Proceedings of the 9th international conference on the practice and theory of automated timetabling, pp. 283–302 (2012)
12. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics. v5 pp. 287–326 (1977)
13. Kolisch, R., Drexl, A.: Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions **29**(11), 987–999 (1997)
14. Leung, J.Y.T., Li, H., Pinedo, M.: Order scheduling in an environment with dedicated resources in parallel. Journal of Scheduling **8**(5), 355–386 (2005)
15. Mastrolilli, M., Queyranne, M., Schulz, A.S., Svensson, O., Uhan, N.A.: Minimizing the sum of weighted completion times in a concurrent open shop. Operations Research Letters **38**(5), 390–395 (2010)

16. Naderi, B., Fatemi Ghomi, S., Aminnayeri, M., Zandieh, M.: A contribution and new heuristics for open shop scheduling. Computers & Operations Research **37**(1), 213–221 (2010)
17. Ng, C., Cheng, T.C.E., Yuan, J.: Concurrent open shop scheduling to minimize the weighted number of tardy jobs. Journal of Scheduling **6**(4), 405–412 (2003)
18. Pinedo, M.: Scheduling: theory, algorithms, and systems. Springer (2012)
19. Roemer, T.A.: A note on the complexity of the concurrent open shop problem. Journal of scheduling **9**(4), 389–396 (2006)
20. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)
21. Wagneur, E., Sriskandarajah, C.: Openshops with jobs overlap. European Journal of Operational Research **71**(3), 366–378 (1993)

A Mathematical Model and Metaheuristics for Time Dependent Orienteering Problem

Aldy GUNAWAN[⋆] · **Zhi YUAN**[⋆] ·
Hoong Chuin LAU

Abstract This paper presents a generalization of the Orienteering Problem, the Time-Dependent Orienteering Problem (TDOP) which is based on the real-life application of providing automatic tour guidance to a large leisure facility such as a theme park. In this problem, the travel time between two nodes depends on the time when the trip starts. We formulate the problem as an integer linear programming (ILP) model. We then develop various heuristics in a step by step fashion: greedy construction, local search and variable neighborhood descent, and two versions of iterated local search. The proposed metaheuristics were tested on modified benchmark instances, randomly generated problem instances, and two real world problem instances extracted from two popular theme parks in Asia. Experimental results confirm the effectiveness of the developed metaheuristic approaches, especially an iterated local search with adaptive perturbation size and probabilistic intensified restart mechanism. It finds within an acceptably short computation time, the optimal or near optimal solutions for TDOP instances of realistic size as in our target application.

Keywords Time-Dependent Orienteering Problem · Integer Linear Programming · Metaheuristics · Iterated Local Search

1 Introduction

The Orienteering Problem (OP) is originated from the sport game of orienteering [2]. The main goal is to find a single route by visiting as many nodes as possible

[⋆] Contributed equally

A. Gunawan and H.C. Lau
School of Information Systems, Singapore Management University, Singapore
E-mail: aldygunawan, hclau@smu.edu.sg
Z. Yuan
Professorship of Applied Mathematics, Department of Mechanical Engineering, Helmut Schmidt University, Hamburg, Germany
E-mail: yuanz@hsu-hh.de

that maximizes the total collected score subject to a given time budget frame. It is assumed that the starting point and the end point are fixed. Many OP applications are described in the literature: selective travelling salesperson problem [21]), home fuel delivery problem [8], single-ring design problem [20], and mobile tourist guide [17].

Several variants of the OP include: 1) Team Orienteering Problem (TOP) [2, 18], 2) Orienteering Problem with Time Windows (OPTW) [15], 3)Team Orienteering Problem with Time Windows (TOPTW) [23], 4) Time-Dependent Orienteering Problem (TDOP) [5].

In this paper, we study the Time-Dependent Orienteering Problem (TDOP), which is a generalization of OP. In the classical OP, the changes to the network over time are not taken into account. However, in certain networks, the route between two nodes actually depends on the network properties, such as congestion levels, construction zone on certain segments, etc., which will affect the travel time between two nodes. Our target application of this work is to provide automatic tour guidance to theme park visitors, taking into account the waiting time of a theme park varies over time. The goal is to maximize the overall utility of the visited attractions within the tourist's available visiting period.

We formulate the TDOP as an Integer Linear Programming (ILP) model. Due to the computational inefficiency in solving large-scale instances with a commercial ILP solver, we then develop various metaheuristics, including a greedy construction heuristic, two local search operators and variable neighborhood descent, and two versions of iterated local search: a basic version and a further improved version by adaptive perturbation strength and probabilistic intensification mechanism. All these approaches were tested on modified benchmark instances, randomly generated instances and two case studies extracted from real world theme park data.

The paper is organized as follows. We first provide a brief review of the OP and TDOP in Section 2. We then describe the TDOP and formulate it as an Integer Linear Programming model in Section 3. In Section 4, metaheuristics are proposed to solve the problem. Section 5 provides the computational results together with the analysis of the results. Finally, we provide concluding perspectives and directions for future research.

2 Literature Review

The Orienteering Problem (OP) [21], also known as the selective travelling salesperson problem [11] or traveling salesman problem with profits [3], has received increasing attention among researchers during recent decades. A comprehensive survey of OP can be found in [24]. An earlier work [3] also provided a survey of different classes of applications, modeling approaches and solution techniques.

[21] is the first to introduce a general description of the sport of orienteering and develop heuristic approaches based on a Monte Carlo technique for the OP. [9] introduced a new procedure which is based on four concepts: center of gravity, randomness, subgravity, and learning. Several metaheuristics for solving the OP have been proposed by researchers, such as Tabu Search [7], Genetic Algorithm (GA) [19] and Ant Colony Optimization [16].

Time-Dependent Orienteering Problem (TDOP) is an extension of OP by taking into account changes to the network over time. The travel time from one node to another node varies with time and depends on the start time. It was first proposed by [5], where a $(2+\varepsilon)$-approximation algorithm is also proposed for solving it. However, [5] considers only equal utility for all nodes, thus the problem actually reduces to visiting as many nodes as possible rather than maximizing total collected utilities; besides, the proposed algorithm has not been empirically studied. A very recent work [25] proposed ant colony system for solving a variant of TDOP based on speed model, where the travel time between two nodes does not depend on the starting time but on the speed given at each time step during the travel. The speed for travelling between two nodes at a time step is assumed invariant under different starting time. This speed model is very interesting since it preserves the FIFO property: a vehicle that starts earlier will always arrive earlier. And this is practical in many real-world applications. However, this speed model may not hold in the general TDOP. For example, if the travel time between two nodes includes waiting for a shuttle that arrives according to a fixed time table, then it is not straightforward to compute the waiting time by the speed model due to the speed invariance. In this work, we follow the more general time-dependence defined in [5] that the travel time between two nodes depends only on the starting time, and these travel times are assumed to be given. Besides, no assumption of FIFO property is assumed from the input travel time data.

Other related work for TDOP includes [12] that proposed the Time-Dependent Team Orienteering Problem; [1] that proposed two genetic algorithms for the Time-Dependent Orienteering Problem with Time Windows which stems from the application of tour itinerary planning in complex and large urban areas in Tehran; [6] that presented the Time-Dependent Team Orienteering Problem with Time Windows which originated from the development of personalised electronic tourist guides by integrating the tourist planning problem and the use of public transportation, and two different approaches based on Iterated Local Search are proposed to solve a set of test instances based on real data for the city of San Sebastian, Spain.

3 Time Dependent Orienteering Problem

A graphical description of the Orienteering Problem (OP) [24] can be briefly introduced as follows. Given a set of nodes $N := \{1, 2, \ldots, n\}$, where node 1 is the starting point, and n is the end point; also given the utility of each node $U := \{u_i : i \in N\}$, the distance matrix $D := \{d_{i,j} : i, j \in N\}$ representing the travel time between any two nodes i, j, and a maximum travel time budget T_{max}, the objective is to find a path P that starts from node 1 and ends at node n before T_{max}, such that the total utility collected at all visited nodes in path P is maximized.

In the time dependent orienteering problem (TDOP), the travel time from nodes i to j depends on the time when the trip starts. Given k time horizons $H := \{h_1, h_2, \ldots, h_k\}$ with $h_i = \underline{h_i}, \underline{h_i} + 1, \ldots, \overline{h_i}$, where the travel time within each horizon is constant $D := \{d_{i,j,h} : i, j \in N, h \in H\}$.

In our target practical application, theme park tour guidance, the travel time $d_{i,j,h}$ from node i to node j includes traveling from i to j, waiting time at j, and the service time at j. The length of $d_{i,j,h}$ mainly depends on the waiting time at attraction j. The high-utility attractions are usually preferred by most of the

Table 1 Parameters and decision variables

Notations	**Descriptions**
u_i	the utility score for node i
T_{max}	the time budget to leave node n; it also specifies the number of time steps.
$d_{i,j,t}$	the travel time from node i to node j, started at time period t.
X_{ijt}	$= 1$ if travel occurs from node i to node j at time period t; otherwise, 0

tourists, and thus usually have a long queue during busy hours. If our tour guidance is used by a small portion of the visitors, taking into account their attraction preference as utility value, and the historical data or real-time crowd statistics as prediction of the waiting time, an optimal tour is expected to maximize the total utility of the visited attractions, while avoiding visiting a popular attraction at its most crowded hour.

We formulate the TDOP as an integer linear programming (ILP) model. Table 1 presents parameters and decision variables required to formulate the ILP model.

$$Maximize \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{T_{max}} u_i \times X_{i,j,t} \tag{1}$$

The objective function (1) is to maximize the total collected utility score when visiting nodes at certain time periods.

$$\sum_{i>1}^{n} \sum_{t=1}^{T_{max}} X_{i,1,t} = 0 \tag{2}$$

$$\sum_{j>1}^{n} \sum_{t=1}^{T_{max}} X_{1,j,t} = 1 \tag{3}$$

Constraint (2) ensures that there is no return trip to the start point and constraint (3) ensures that the start point is node 1.

$$\sum_{j=1}^{n-1} \sum_{t=1}^{T_{max}} X_{n,j,t} = 0 \tag{4}$$

$$\sum_{i=1}^{n-1} \sum_{t=1}^{T_{max}} X_{i,n,t} = 1 \tag{5}$$

Constraints (4) and (5) ensure that the last visited node is node n.

$$\sum_{\substack{i=1 \\ i \neq e}}^{n-1} \sum_{t=1}^{T_{max}} X_{i,e,t} = \sum_{\substack{j=2 \\ j \neq e}}^{n} \sum_{t=1}^{T_{max}} X_{e,j,t} \; \forall e = 2, 3, \ldots, (n-1) \tag{6}$$

Constraint (6) guarantees the connectivity of the path for each node visited.

$$\sum_{\substack{j=2 \\ j \neq i}}^{n} \sum_{t=1}^{T_{max}} X_{i,j,t} \leq 1 \; \forall i = 2, 3, \ldots, (n-1) \tag{7}$$

Constraint (7) ensures that each node is visited at most once.

$$\sum_{e \neq i,j} \sum_{u=t+d_{i,j,t}}^{T_{max}} X_{j,e,u} \geq X_{i,j,t}; \forall i,j = 1,\ldots,n-1, i \neq j, j \neq 1, t \leq T_{max} - d_{i,j,t} \quad (8)$$

The constraint (8) enforces if a trip from i to j starts at time t, and j is not the end point, then a trip must start from j at a time period later than after visiting j.

$$X_{i,j,t} = 0, \forall i \neq j, t > T_{max} - d_{i,j,t} \quad (9)$$

(9) removes infeasible trips that starts too late.

4 Metaheuristics

As proved by Golden et al. in 1987 [8] that orienteering problem (OP) is NP-hard, i.e. no polynomial time algorithm could be designed to solve this problem to optimality. As a generalisation of the OP with time-dependent travel time, the TDOP is also NP-hard. The mathematical model introduced in Section 3 can be regarded as a time-expanded graph of the OP, which substantially increases the problem dimension, making computation even more challenging.

In this paper, our target application of the TDOP is to provide real-time tour guidance to theme park visitors, so it is practically infeasible to make the tourists wait minutes or even hours for an optimal route. Therefore, a fast and effective heuristic approach is essential for devising a practical theme park routing tool in a dynamic environment.

4.1 Greedy Construction Heuristic

A greedy construction heuristic is a myopic strategy that always chooses one solution component that is with the best immediate desirability based on a greedy criterion. In TDOP, A path P is initialized with the starting node 1 and end node n. Then, the construction proceeds iteratively by adding a solution component, in our case, one unvisited node, to P. In each iteration, the selected unvisited node is added to the end of the path, right before the end node n.

One important feature in the TDOP is to handle the time-dependent travel time $d_{i,j,h}$. Since travel time changes as the starting time changes, and no FIFO property is assumed, we are faced with the following non-trivial subproblems:

- EarliestArrival: given a starting time t^{start} to travel from node n^{prev} to current node n^{cur}, find the earliest arrival time t^{arr} at n^{cur}; and
- LatestDeparture: given an arrival time t^{end} at node n^{next}, find the latest departure time t^{dep} from node n^{cur}.

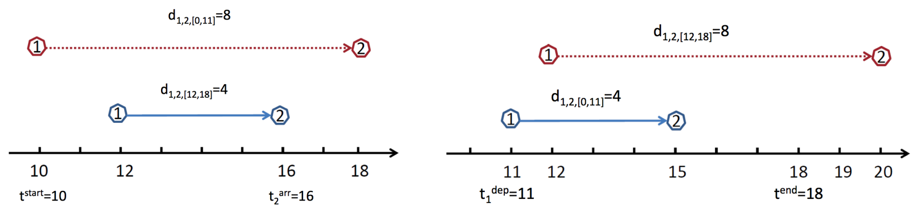
An example of these two subproblems can be illustrated in Figure 1. They are handled is as follows: EarliestArrival, as outlined in Procedure 1, iteratively checks the earliest possible arrival time of each proceeding time horizon, until a horizon surpasses current earliest possible arrival time; LatestDeparture in

Procedure 1 EarliestArrival (t^{start}, n^{prev}, n^{cur}, D, H)

```
$h^{cur} \leftarrow$ the time horizon that contains $t^{start}$
$t^{end} \leftarrow t^{start} + d_{n^{prev},n^{cur},h^{cur}}$
for all $h \in \{h^{cur}+1, \dots, H\}$ do
  if $t^{end} < \underline{h}$ then
    break
  else
    if $\underline{h} + d_{n^{prev},n^{cur},h} \le t^{end}$ then
      $t^{end} \leftarrow \underline{h} + d_{n^{prev},n^{cur},h}$
    end if
  end if
end for
return $t^{end}$
```

Procedure 2 LatestDeparture (t^{end}, n^{cur}, n^{next}, D, H)

```
$h^{cur} \leftarrow$ the time horizon that contains $t^{end}$
$t^{start} \leftarrow t^{end} - d_{n^{prev},n^{cur},h^{cur}}$
for all $h \in \{h^{cur}-1, \dots, 1\}$ do
  if $t^{start} > \overline{h}$ then
    break
  else
    $t^{start} \leftarrow \min(t^{end} - d_{n^{prev},n^{cur},h}; \overline{h})$
  end if
end for
return $t^{start}$
```



(a) Starting from node 1 at time 10, the earliest arrival time at node 2 is 16.

(b) Requiring arrival at node 2 at time 18, the latest departure time at node 1 is 11.

Fig. 1 Example of finding the earliest arrival and latest departure time in time dependent travels. The travel time from node 1 to node 2 is 4 if it starts within time interval $[0, 11]$, and is 8 within [12, 18].

Procedure 2 on the contrary, iteratively checks backwards each time horizon until one horizon h in which a trip can be started, then the latest departure time is set to either $t^{end} - d_{n^{prev},n^{cur},h}$, or the horizon boundary $\overline{h}$, whichever starts first.

In each construction iteration, a node is feasible to be appended to the path only if its earliest arrival time computed by Procedure 1 is no later than its latest departure time computed by Procedure 2. From the set of all feasible nodes N^*, a best node n^* is then selected by the following greedy criterion:

$$n^* \leftarrow \arg\max_{i \in N^*} \frac{u_i^\alpha}{\delta_i^\beta} \cdot \text{rand}(\frac{1}{\gamma}, 1). \tag{10}$$

That is, the desirability of a node $i \in N^*$ depends on two terms: the utility value u_i, and the distance δ_i which is calculated as the difference between the earliest

Procedure 3 FORWARDPROPAGATION $(P,\ n_s)$

```
$t^{start} \leftarrow t^{arr}_{n_{s-1}}$
for all $i \in \{s, s+1, \ldots, m\}$ do
    $t^{arr}_{n_i} \leftarrow$ EARLIESTARRIVAL$(t^{start}, n_{i-1}, n_i)$
    $t^{start} \leftarrow t^{arr}_{n_i}$
end for
```

Procedure 4 BACKWARDPROPAGATION $(P,\ n_s)$

```
$t^{end} \leftarrow t^{dep}_{n_{s+1}}$
for all $i \in \{s, s-1, \ldots, 1\}$ do
    $t^{dep}_{n_i} \leftarrow$ LATESTDEPARTURE$(t^{end}, n^i, n^{i+1})$
    $t^{end} \leftarrow t^{dep}_{n_i}$
end for
```

arrival time at i and earliest possible leaving time at the previous node. α and β are parameters determining the impact of utility and distance, respectively. The rand$(\frac{1}{\gamma}, 1)$ is the noise term that generates a uniformly random number ranging from $\frac{1}{\gamma}$ to 1, where $\gamma \geq 1$ is a parameter that allows candidates of γ times worse than the best one to be selected. If γ is set to 1, it selects deterministically the most desirable node at each step. The greedy construction terminates when either the visitation time budget T_{max} is finished, or no more unvisited node can be appended.

4.2 Local Search and Variable Neighborhood Descent

Two types of basic local search operators are adopted in our work, the insert and replace operators. We also consider hybridizing the two operators within a variable neighborhood descent framework. In order to make the feasibility check of each operator more efficient, a starting time propagation procedure is used and described below.

4.2.1 Starting Time Propagation

Given a path P of m nodes, $P := \{n_i : i = 1, 2, \ldots, m\}$, the starting time propagation concerns assigning the earliest arrival time t^{arr}_i and the latest departure time t^{dep}_i for each node $n_i \in P$. It can be classified into two different procedures, the FORWARDPROPAGATION in Procedure 3 that propagates the earliest arrival time in the forward direction, and the BACKWARDPROPAGATION in Procedure 4 that propagates the latest departure time in the backward direction. Note that both procedures can also propagate for a partial path, starting from a certain index n_s.

The FORWARDPROPAGATION procedure iteratively takes the earliest arrival time of the previous node to obtain the earliest arrival time of the current node by using the EARLIESTARRIVAL procedure, while the BACKWARDPROPAGATION procedure iteratively takes the latest departure time of the next node to compute the latest departure time of the current node by the LATESTDEPARTURE procedure.

Procedure 5 Insert $(P,\ N)$

```
for all $i \in$ random nodes of $N$ do
  for all $j \in$ random positions of $P$ do
    $t_i^{arr} \leftarrow$ EarliestArrival($t_{n_j}^{arr}, n_j, i$)
    $t_i^{dep} \leftarrow$ LatestDeparture($t_{n_{j+1}}^{dep}, i, n_{j+1}$)
    if $t_i^{arr} \leq t_i^{dep}$ then
      insert $i$ into path $P$ at position $j$
      $N \leftarrow N \setminus \{i\}$
      ForwardPropagation($P, j+1$)
      BackwardPropagation($P, j-1$)
      break
    end if
  end for
end for
```

4.2.2 Local Search Operators: Insert and Replace

The random first improvement strategy is applied for the insert operator, as outlined in Procedure 5. Each random unvisited node $i \in N$ is tried to be inserted between random position j and $j+1$ in the path P. The earliest arrival time of node i is computed based on the earliest arrival time at node j, and the latest departure time of node i is computed based on the latest arrival time of node $j+1$. If the node i can arrive earlier than its latest departure time, it is inserted at position j, and earliest arrival time of the nodes after j will be updated using ForwardPropagation and the latest departure time of the nodes before j will be updated using BackwardPropagation.

Similarly, the replace operator also uses a random first improvement strategy. An unvisited node i is considered better than node n_j at position j of the path P, either when its utility value is strictly better, or in the case of equal utility, the difference between its earliest arrival time and latest departure time is strictly larger. In such case, the two nodes are exchanged, and the starting time of the rest of the nodes are updated by propagation.

4.2.3 Variable Neighborhood Descent

The basic idea of Variable Neighborhood Descent (VND) [14] is to apply a set of local search operators iteratively, such that the final solution obtained is locally optimal with respect to all local search operators (subject to iteration order). In such a way, variable neighborhoods, such as insert and replace operators in Section 4.2.2, can be hybridized. Note that VND starts with a complete solution and returns a modified solution, hence itself can also be regarded as a local search operator.

4.3 Iterated Local Search

Iterated local search (ILS) [13] is a simple yet effective, general-purpose metaheuristic. It starts with an initial solution and a local search, and then iterate the three components of ILS: perturbation, local search, and acceptance criterion.

Procedure 6 PERTURBATION $(P,\ N)$

```
randomly remove $s$ nodes from $P$
tabu the removed nodes for one local search iteration
ForwardPropagation$(P, 1)$
BackwardPropagation$(P, |P|)$
```

Procedure 7 ACCEPTANCEBASIC $(P,\ P^{gb},\ P^{rb},\ I^{max})$

```
if Incumbent $P$ is better than restart best $P^{rb}$ then
    $P^{rb} \leftarrow P$
    if Incumbent $P$ is better than global best $P^{gb}$ then
        $P^{gb} \leftarrow P$
    end if
else
    $P \leftarrow P^{rb}$
    if consecutive non-improved iteration count exceeds maximum $I^{max}$ then
        Restart by a greedy construction followed by a variable neighborhood descent
    end if
end if
```

Here, the initial solution is constructed by the greedy method in Section 4.1, the variable neighborhood descent in Section 4.2.3 is adopted as the subsidiary local search procedure. In the PERTURBATION procedure outlined in Procedure 6, s random nodes are removed from the incumbent path P. Note that a node can be removed only if the earliest arrival time at the next node is not delayed. This is usually not an issue in a Euclidean-distance graph, however, it may not hold if the travel time on an edge is time dependent due to traffic conditions as in some benchmark instances mentioned in Section 5.1. s is a parameter reflecting the perturbation strength. A high value of s may result in slow convergence, while a smaller value of perturbation strength may quickly lead to a good solution at the beginning but is more likely to be trapped in a deep local optimum. These s removed nodes are tabued for one local search iteration, so that they cannot be immediately inserted or replaced back to the incumbent path, allowing more diversification.

We have considered two versions of the iterated local search in this work, a basic version named Basic ILS and a modified version named Adaptive ILS. Their main difference lies in the acceptance criterion, or more precisely, in how to handle algorithm stagnation. The stagnation is referred to when the maximum number of non-improved iterations I^{max} is reached. In Basic ILS, the algorithm is restarted by a greedy construction and variable neighborhood descent, as detailed in Procedure 7. Adaptive ILS in Procedure 8 first increments the perturbation strength s, and resets the iteration counter to zero, until the maximum perturbation strength s^{max} is reached, then restarts the search. However, since Adaptive ILS allows more non-improved iterations before restart, if it is trapped in an uninteresting region, many iterations will be wasted. To this end, we developed a probabilistic intensification mechanism. For each unsuccessful iteration, with an intensification probability p^{in}, the best-so-far solution, instead of the restart best solution, will be copied into the incumbent.

Procedure 8 AcceptanceAdaptive (P, P^{gb}, P^{rb}, p^{in}, I^{max}, s^{max})

```
if Incumbent $P$ is better than restart best $P^{rb}$ then
    $P^{rb} \leftarrow P$
    Set perturbation strength to minimum $s \leftarrow s^{min}$
    if Incumbent $P$ is better than global best $P^{gb}$ then
        $P^{gb} \leftarrow P$
    end if
else
    if rand$(0, 1) < p^{in}$ then
        $P \leftarrow P^{gb}$        // Intensify by copying the global best solution to incumbent
    else
        $P \leftarrow P^{rb}$
    end if
    if consecutive non-improved iteration count exceeds maximum $I^{max}$ then
        Increment perturbation strength $s \leftarrow s + 1$
        if $s > s^{max}$ then
            Restart by a greedy construction followed by a variable neighborhood descent
        end if
    end if
end if
```

5 Computational Results

The comprehensive computational results including experimental setup will be described below.

5.1 Instance Setup

Three classes of time dependent orienteering problem (TDOP) instances are considered in this study:

– `Benchmark`. The benchmark instances are adopted from [22]. These instances were initially developed by [2]. The number of nodes in these instances varies from 21 to 102. These instances were further adapted by Verbeek et al. [25] by varying travel time at different time horizons. Each instance has a time span from 7 am to 9 pm. These 14 hours were divided into four different time horizons: 7 am to 9 am, 9 am to 5 pm, 5 pm to 7 pm, and 7 pm to 9 pm, since the travel time on each edge may depend on the traffic load at different period of the day. In order to use our time-expanded model for these benchmark instances, the original travel time is discretized by a unit of $\mu = 1, 5, 15, 30$ minutes, which corresponds to a total number of time steps $T_{max} = 840, 168, 56, 28$. In order to guarantee the feasibility, the discretization of a travel time d is by rounding up after divided by the time unit, $\lceil d/\mu \rceil$. The larger the time unit, the less the number of time steps, and thus, easier for the time-expanded model to be solved.
– `Random`. The second class of instances is randomly generated with varying values of the parameters: number of nodes n and time budget T_{max}. Each time step is considered a time horizon. The utility score for each node is generated randomly between 1 to 9. The start and end nodes are allocated with zero utility scores.

Table 2 Characteristics of problem instances and their optimal or best-known solution computed by CPLEX with the mathematical program model. The instance characteristics include number of nodes $|N|$, number of discrete time steps T_{max}, and number of time horizons k.

Class	**Name**	$\|N\|$	T_{max}	k	**CPLEX Result**	
					Optimal	**Time**
`Benchmark`	$TDOP1$-{1, 5, 15, 30}	32	{840, 168, 56, 28}	4	$220^{‡}$	18.47 mins
	$TDOP2$-{1, 5, 15, 30}	21	{840, 168, 56, 28}	4	$355^{‡}$	55.69 secs
	$TDOP3$-{1, 5, 15, 30}	33	{840, 168, 56, 28}	4	$590^{‡}$	1.88 hours
	$TDOP4$-{1, 5, 15, 30}	100	{840, 168, 56, 28}	4	$623^{‡*}$	>24 hours
	$TDOP5$-{1, 5, 15, 30}	66	{840, 168, 56, 28}	4	$850^{‡}$	13.11 hours
	$TDOP6$-{1, 5, 15, 30}	64	{840, 168, 56, 28}	4	$768^{‡}$	21.23 hours
	$TDOP7$-{1, 5, 15, 30}	102	{840, 168, 56, 28}	4	$690^{‡*}$	>24 hours
`Random`	$Rand10 \times 10$	10	10	10	20	0.17 secs
	$Rand10 \times 20$	10	20	20	41	0.47 secs
	$Rand10 \times 30$	10	30	30	46	2.06 secs
	$Rand10 \times 40$	10	40	40	37	5.60 secs
	$Rand20 \times 10$	20	10	10	42	1.20 secs
	$Rand20 \times 20$	20	20	20	80	2.14 mins
	$Rand20 \times 30$	20	30	30	84	10.73 mins
	$Rand20 \times 40$	20	40	40	107	14.10 mins
	$Rand30 \times 10$	30	10	10	47	1.94 secs
	$Rand30 \times 20$	30	20	20	103	3.49 mins
	$Rand30 \times 30$	30	30	30	151*	>24 hours
	$Rand30 \times 40$	30	40	40	143*	>24 hours
	$Rand40 \times 10$	40	10	10	63	10.13 secs
	$Rand40 \times 20$	40	20	20	145	26.26 mins
	$Rand40 \times 30$	40	30	30	176*	>24 hours
	$Rand40 \times 40$	40	40	40	217*	>24 hours
`Real world`	$Real1$	17	36	9	195	15.3 mins
	$Real2$	40	42	42	208*	>24 hours

‡ optimal or best known solution for discrete time unit 30.
* best known solution obtained after 24 hours

– `Real world`. Two real-world instances are obtained from two of the most popular theme parks in Asia. Each node represents an attraction, the utility vector of each attraction is derived from user preferences data, and the travel time from one attraction to another includes the traveling time (by shuttle or on foot), and service time at an attraction, and the waiting time that varies over time.

The details of instance characteristics can be referred to in Table 2.

5.2 Computational Results of Mathematical Model

The mathematical programming model in Section 3 is solved by commercial solver CPLEX 10.2 on a computing server with multi-core Intel Xeon CPU ES-2667 at 2.90 GHz with 256GB RAM running Microsoft Server 2008 R2 Enterprise. Up to 24 threads are used per run.

The last two columns of Table 2 summarizes the optimal solution and computation time obtained by CPLEX for each instance. The computational scalability of the problem is best illustrated in the `Random` class of instances. Although most instances can be solved to optimality, the computation time explodes quickly as the number of nodes and number of time steps increase. Instances with more than 30 nodes and 30 time steps cannot be solved to provable optimality within 24 hours. It takes minutes of computation time to compute an optimal route for instances from 20 nodes and 20 time steps. In the instance class `Benchmark`, CPLEX is only applied to instances discretized by time unit 30 minutes, resulting in 28 time steps in 14 hours.[1] Note that since the number of time horizons k is reduced to 4, the computational time required is also reduced noticeably. Here, instances with up to 66 nodes and 28 time steps can be solved to provable optimality within 24 hours. However, the computation time required is very long: It takes over 1 hour to solve instances from 33 nodes. Concerning the two `real world` theme park instances, the smaller one $Real1$ with 17 nodes and 36 time steps is solved to optimum in around 15 minutes, while the larger one $Real2$ cannot be solved to provable optimality within 24 hours.

Although most of the instances considered can be solved optimally by CPLEX, the computation time is unpractically long, usually minutes to hours for a realistic problem size. However, our target application is a time-critical problem: each instance is generated for each visitor on the fly based on their personal preference and available time, and then the tour guidance system is expected to compute a good solution within an acceptable time, i.e., maximum one second. Besides, the coarse time discretization required by the mathematical model also reduces the accuracy of travel time input. Therefore, the mathematical programming may not be an ideal approach for this application, however, the optimal or best known solution computed in this section can be a good reference in assessing the quality of our metaheuristic approaches in the next section.

[1] For instances with smaller discretization time units such as 15, 5, and 1 minutes, most of the instances cannot be solved to optimality within 24 hours.

Table 3 The computational results of the four metaheuristics: greedy, variable neighborhood descent (VND), basic iterated local search (Basic ILS), and adaptive iterated local search (Adaptive ILS). Each metaheristic is run 30 trials on each instance, and best, mean, and worst performance of each 30 runs are listed below. Percentage deviation from optimal or best-known solution is listed, where available in Table 2, or else the objective value is listed. Statistically significantly best results are marked in bold face.

Instance	Greedy			VND			Basic ILS			Adaptive ILS		
	Best	Mean	Worst	Best	Mean	Worst	Best	Mean	Worst	Best	Mean	Worst
$TDOP1$-1	280	273.8	270	280	280	280	285	**284.3**	280	285	**284.8**	280
$TDOP1$-5	270	264.7	260	280	273.8	270	280	**280**	280	280	**280**	280
$TDOP1$-15	245	238.7	235	260	251.8	245	260	257.5	255	260	**260**	260
$TDOP1$-30	2.27%	4.02%	6.82%	0.00%	1.29%	4.55%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$TDOP2$-1	450	437.7	430	450	**450**	450	450	**450**	450	450	**450**	450
$TDOP2$-5	430	422.0	415	440	430.7	430	440	**440**	440	440	**440**	440
$TDOP2$-15	385	376.3	375	395	**395**	395	395	**395**	395	395	**395**	395
$TDOP2$-30	0.00%	2.96%	5.63%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$TDOP3$-1	760	744.3	740	770	762.7	750	780	**778.3**	770	780	**777.7**	770
$TDOP3$-5	730	725.7	720	740	731	730	750	**742.3**	730	750	**744.3**	730
$TDOP3$-15	660	647.7	640	670	660.7	650	670	**665**	660	670	**663**	660
$TDOP3$-30	0.00%	0.90%	3.39%	0.00%	**0.06%**	1.69%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$TDOP4$-1	1014	980.2	964	1032	1003.3	989	1056	1032.2	1012	1085	**1048.5**	1015
$TDOP4$-5	935	910.9	891	956	927.2	909	964	942	921	989	**953.9**	925
$TDOP4$-15	791	777.1	761	824	791	776	831	800.8	771	842	**812.7**	783
$TDOP4$-30	2.25%	4.03%	5.78%	3.21%	4.20%	5.14%	1.93%	4.56%	5.94%	1.61%	**3.55%**	6.10%
$TDOP5$-1	1495	1458.2	1410	1495	1468.7	1425	1505	1488.8	1460	1505	**1497.3**	1465
$TDOP5$-5	1260	1233.8	1195	1260	**1238.2**	1200	1260	**1244.7**	1220	1260	**1246.8**	1215
$TDOP5$-15	865	851.7	835	870	854	845	870	859.3	845	870	**865**	850
$TDOP5$-30	0.59%	1.78%	2.94%	0.59%	1.47%	3.53%	0.00%	0.69%	1.76%	0.00%	**0.24%**	1.18%
$TDOP6$-1	1326	1303.4	1290	1326	1316.4	1302	1338	1328.8	1320	1344	**1337.6**	1332
$TDOP6$-5	1224	1186.4	1158	1236	1198.4	1176	1242	1221	1206	1254	**1237.0**	1212
$TDOP6$-15	1182	1153.2	1134	1206	1180.2	1164	1236	1211	1194	1254	**1233.6**	1206
$TDOP6$-30	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$TDOP7$-1	1182	1164.6	1152	1232	1211.6	1192	1285	1260.1	1236	1318	**1283.2**	1253
$TDOP7$-5	1084	1065.2	1050	1113	1093.6	1078	1182	1140	1116	1195	**1159.1**	1134
$TDOP7$-15	960	940.9	920	1000	982.7	967	1026	1005.8	980	1046	**1019.8**	983
$TDOP7$-30	1.30%	1.82%	2.46%	0.58%	1.42%	2.03%	0.87%	1.46%	1.59%	0.58%	**1.14%**	1.59%
$Rand10 \times 10$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$Rand10 \times 20$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$Rand10 \times 30$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$Rand10 \times 40$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$Rand20 \times 10$	7.14%	7.14%	7.14%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$Rand20 \times 20$	0.00%	0.80%	1.27%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$Rand20 \times 30$	0.00%	0.16%	1.19%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$Rand20 \times 40$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$Rand30 \times 10$	6.38%	6.38%	6.38%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$Rand30 \times 20$	3.88%	3.88%	3.88%	0.00%	1.62%	1.94%	0.00%	**0.45%**	0.97%	0.00%	**0.58%**	1.94%
$Rand30 \times 30$	3.29%	4.30%	4.61%	1.32%	2.11%	3.29%	0.00%	**1.64%**	2.63%	0.00%*	**1.36%**	2.63%
$Rand30 \times 40$	0.00%	0.40%	0.70%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.00%
$Rand40 \times 10$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$Rand40 \times 20$	0.00%	**0.62%**	0.69%	0.00%	**0.67%**	0.69%	0.00%	0.74%	1.38%	0.00%	0.83%	2.07%
$Rand40 \times 30$	0.56%	1.98%	3.95%	0.56%	1.53%	2.26%	0.56%	1.54%	2.82%	0.00%‡	**1.21%**	2.26%
$Rand40 \times 40$	2.30%	2.69%	3.23%	0.46%	1.03%	1.38%	0.46%	0.89%	1.38%	0.00%	**0.68%**	1.38%
$Real1$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$Real2$	0.96%	1.54%	1.92%	0.00%	**0.19%**	0.48%	0.00%	**0.18%**	0.48%	0.00%	**0.14%**	0.48%

* New best known solution found: 152. ‡ New best known solution found: 177.

5.3 Computational Results of Metaheuristics

The metaheuristics introduced in Section 4 were implemented in Java, compiled by JDK 7, and run on a MacBook Air with 1.7GHz Intel Core i7 and 8GB memory. Only single thread is used for each metaheuristic run.

Four metaheuristics are considered here: a restart greedy append construction heuristic (Greedy), a restart variable neighborhood descent (VND), a basic version of iterated local search with restart (Basic ILS), and a modified iterated local search with adaptive perturbation size and probabilistically intensified restart (Adaptive ILS). The following parameter settings are adopted for these algorithms. For greedy construction, the weight of utility and distance is set to $\alpha = \beta = 1$ following most of the existing work on orienteering problem; the noise factor γ is set to 1 in the first run to make it deterministic, and set to 5 in all the following runs. For Basic ILS, the perturbation strength s is set to 3, and the maximum non-improved iteration I^{max} is set to 10. For Adaptive ILS, I^{max} is also set to 10, and the minimum perturbation strength s^{min} is set to 3, and the maximum $s^{max} = \lceil |P|/4 \rceil$, i.e. the roundup of one fourth of the incumbent path size; the intensification probability p^{in} is set to 0.05.

Each metaheuristic is allowed a maximum runtime of one second, and is performed 30 independent runs on each instance. The best, mean and worst performance of 30 runs are recorded in Table 3. For each instance, the significantly best performing metaheuristics by the Wilcoxon's signed rank test at 0.05 level is marked in bold face. As is clearly shown, Adaptive ILS is the significantly best performing algorithms for all benchmark instances and real world instances. Especially in the largest benchmark instances $TDOP4$ and $TDOP7$, Adaptive ILS is usually over 1% better than the Basic ILS, and around 2 to 5% better than VND and greedy. It remains the significantly best performing algorithm for all but one random instances, where it was slightly but statistically significantly outperformed by greedy and VND. Basic ILS as runner-up significantly outperforms VND, which in turn performs significantly better than greedy.

Comparing the metaheuristic approaches with the mathematical programming approach, regardless of only one second computation time, Adaptive ILS is able to improve the best known solutions of random instances $Rand30 \times 30$ and $Rand40 \times 30$ that are computed by CPLEX 10.2.0 solver in 24 hours. The best run of Adaptive ILS has found the optimum of all random instances, real world instances, and the benchmark instances up to size 66. It misses the optimum of the largest instances $TDOP4$-30 and $TDOP7$-30 composed of 100 nodes, leaving a gap of 0.6 to 1.6 % at its best run, or 1.1 to 3.6% at its average. One second is probably too short for instances of such size. Considering mean performance, for instance size under 30, Adaptive ILS finds the optimal solution in each of the 30 runs; the average deviation from the optimum is less than 1.4% for the random instances up to size 40, 0.14% for the real world instances, and 0.24% for the benchmark instances up to size 66.

Another important advantage of the metaheuristic approaches over the time-expanded mathematical model is that, it does not require a time discretization, and thus can use the travel time of arbitrary accuracy. Since the travel time is discretized by rounding up to guarantee feasibility, it compromises the quality of the obtained solution. Comparing the result of benchmark instances with time unit of 1 minute in Table 3 to optimal result with time unit 30 minutes in Table 2, the solution quality loss is from over 20% as on instance $TDOP2$ with size 21, to around 45% on instance $TDOP5$ with size 66. A finer time discretization than 30 minutes suffers the time-expanded mathematical model, however, using a coarser discretization at the expense of solution quality also demeans the original motivation of using mathematical model, which is to guarantee optimality. The metaheuristic approaches explored in this work, especially the Adaptive ILS, appear to be practical and promising for our application problem.

6 Conclusion

This paper presents a general form of the Time-Dependent Orienteering Problem (TDOP). We formulated this problem by an integer linear programming (ILP) model based on time-expanded graph. We further adopted two real-world instances from two most popular theme parks in Asia, together with modified benchmark instances, and randomly generated instances to study the scalability. The computational difficulty turns out to explode quickly for a commercial ILP solver as problem size increases.

As the underlying application problem is time critical, the development of a good metaheuristic is essential. Several heuristics are developed. From experimental results, we showed that our proposed approach, iterated local search with adaptive perturbation size and probabilistic intensified restart, appears to be fast and effective: within one second's computation time, it manages to find the optimal solution for most of the instances considered. It even improves for two instances the best known solutions computed by CPLEX for over 24 hours.

An interesting idea to extend our current ILS is to consider a hierarchical iterated local search approach [10], as well as using automatic algorithm configuration tool to determine the algorithm setting. It will be also interesting to compare with some state-of-the-art approaches such as ant colony systems [25]. From mathematical programming point of view, it would be interesting to consider further Branch-and-Cut techniques as in [4]. In the application aspect, our future research includes extracting more instances from our real world theme park tour guidance data, and extending the applicability of our approach to other types of real-world variants including time-dependent utility score, and the time-dependent team orienteering problem.

Acknowledgements This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA). Zhi Yuan acknowledges support by BMBF Verbundprojekt E-Motion (grant number 05M13GBA).

References

1. Abbaspour, R.A., Samadzadegan, F.: Time-dependent personal tour planning and scheduling in metropolises. Expert Systems with Applications **38**(10), 12,439–12,452 (2011)
2. Chao, I.M., Golden, B.L., Wasil, E.A.: Theory and methodology - the team orienteering problem. European Journal of Operational Research **88**(3), 464–474 (1996)
3. Feillet, D., Dejax, P., Gendreau, M.: Traveling salesman problems with profits. Transportation Science **39**(2), 188–205 (2005)

4. Fischetti, M., Salazar, J.J., Toth, P.: Solving the orienteering problem through branch-and-cut. INFORMS Journal on Computing **10**, 133–148 (1998)
5. Fomin, F.V., Lingas, A.: Approximation algorithms for time-dependent orienteering. Information Processing Letters **83**, 57–62 (2002)
6. Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaza, M.T.: Integrating public transportation in personalised electronic tourist guides. Computers and Operations Research **40**(3), 758–774 (2013)
7. Gendreau, M., Laporte, G., Semet, F.: A tabu search heuristic for the undirected selective travelling salesman problem. European Journal of Operational Research **106**(2-3), 539–545 (1998)
8. Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics **34**(3), 307–318 (1987)
9. Golden, B., Wang, Q., Liu, L.: A multifaceted heuristic for the orienteering problem. Naval Research Logistics **35**(3), 359–366 (1988)
10. Hussin, M.S., Stützle, T.: Hierarchical iterated local search for the quadratic assignment problem. In: M. Blesa, et al. (eds.) Proceeding of Hybrid Metaheuristics (HM 2009), *Lecture Notes in Computer Science*, vol. 5818, pp. 115–129. Springer (2009)
11. Laporte, G., Martello, S.: The selective travelling salesman problem. Discrete Applied Mathematics **26**(2-3), 193–207 (1990)
12. Li, J.: Model and algorithm for time-dependent team orienteering problem. In: S. Lin, X. Huang (eds.) Communications in Computer and Information Science, *Communications in Computer and Information Science*, vol. 175, pp. 1–7 (2011)
13. Lourenço, H., Martin, O., Stützle, T.: Iterated local search. In: Handbook of metaheuristics, pp. 320–353. Springer (2003)
14. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & Operations Research **24**(11), 1097–1100 (1997)
15. Righini, G., Salani, M.: Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers and Operations Research **36**(4), 1191–1203 (2009)
16. Souffriau, W., Vansteenwegen, P., Berghe, G., Oudheusden, D.V.: Automated parameterisation of a metaheuristic for the orienteering problem. In: C. Cotta, M. Sevaux, K. Sörensen (eds.) Adaptive and multilevel metaheuristics, *Studies in Computational Intelligence*, vol. 136, pp. 255–269 (2008)
17. Souffriau, W., Vansteenwegen, P., Vertommen, J., Berghe, G.V., Oudheusden, D.V.: A personalised tourist trip design algorithm for mobile tourist guides. Applied Artificial Intelligence **22**(10), 964–985 (2008)
18. Tang, H., Miller-Hooks, E.: A tabu search heuristic for the team orienteering problem. Computer and Operations Research **32**(6), 1379–1407 (2005)
19. Tasgetiren, M.: A genetic algorithm with an adaptive penalty function for the orienteering problem. Journal of Economic and Social Research **4**(2), 1–26 (2001)
20. Thomadsen, T., Stidsen, T.: The quadratic selective travelling salesman problem. Informatics and mathematical modelling technical report IMM-Technical Report-2003-17, Technical University of Denmark (2003)
21. Tsiligirides, T.: Heuristic methods applied to orienteering. Journal of the Operational Research Society **35**(9), 797–809 (1984)
22. Vansteenwegen, P.: TDOP Format http://www.mech.kuleuven.be/en/cib/op/#section-20 (2013)
23. Vansteenwegen, P., Souffriau, W., Berghe, G.V., van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Computers and Operations Research **36**(12), 3281–3290 (2009)
24. Vansteenwegen, P., Souffriau, W., Oudheusden, D.V.: The orienteering problem: A survey. European Journal of Operational Research **209**(1), 1–10 (2011)
25. Verbeeck, C., Sörensen, K., Aghezzaf, E.H., Vansteenwegen, P.: A fast solution method for the time-dependent orienteering problem. European Journal of Operational Research **236**(2), 419–432 (2014)

Directed Selection using Reinforcement Learning for the Examination Timetabling Problem

Ryan Hamilton-Bryce · Paul McMullan · Barry McCollum

Abstract Traditional heuristic approaches to the Examination Timetabling Problem normally utilize a stochastic method during Optimization for the selection of the next examination to be considered for timetabling within the neighbourhood search process. This paper presents a technique whereby the stochastic method has been augmented with information from a weighted list gathered during the initial adaptive construction phase, with the purpose of intelligently directing examination selection. In addition, a Reinforcement Learning technique has been adapted to identify the most effective portions of the weighted list in terms of facilitating the greatest potential for overall solution improvement. The technique is tested against the 2007 International Timetabling Competition datasets with solutions generated within a time frame specified by the competition organizers. The results generated are better than those of the competition winner in seven of the twelve examinations, while being competitive for the remaining five examinations. This paper also shows experimentally how using reinforcement learning has improved upon our previous technique.

1 Introduction

The challenge of producing acceptable solutions for timetabling problems such as Course and Examination timetabling involves a combination of practical and research based approaches [1]. Due to the complexity of the underlying problems and the potential time requirement in providing acceptable solutions to these problems through the use of discrete methods, over this last few decades research has focused on the use of search based heuristic techniques. A number of review papers on the subject have been published [2], [3]. As

School of EEECS, Queen's University of Belfast, BT7 1NN, Northern Ireland
E-mail: {rhamiltonbryce01, p.p.mcmullan, b.mccollum}@qub.ac.uk

with Course timetabling, progress in research within the area of examination timetabling has been facilitated by the availability of benchmark data sets [4], [5]. Results generated using a wide range of techniques have been reported, with varying levels of success based on both generality of the solver and the time taken to solve [2]. A successful technique can be viewed as one which can produce feasible and workable solutions to a range of differing problems for a given problem domain within a practical timescale.

An examination scheduling track, based on the post-enrolment examination timetabling problem was introduced in the second International Timetabling Competition (ITC2007) [4]. This track introduced a number of real world datasets, drawn from anonymised data from several institutions worldwide. New result sets continue to be validated using the competition's online validation service despite the competition closing almost five years ago. The next timetabling competition to be announced will further develop the problem definition, to further extend the real world aspects of research in this area and to encourage innovation with the development of new problem solvers [6].

Solving the examination timetabling problem generally takes the form of an initial construction phase to produce a feasible solution, and an improvement phase which employ a number different search techniques to find high quality solutions when given specific objectives [7]. It has been observed that for certain construction techniques if construction is continued beyond the point at which a feasible solution has been acquired it is often possible to acquire a better quality for solution on which the improvement phase can operate [8].

Traditionally heuristic based approaches to timetabling problems have utilized a stochastic method for selection of the examination within a neighbourhood search process [9], [10], [11]. This allows for a rapid selection of examinations for the optimization process. It has been shown through experimentation that a link exists between the phases of construction and optimization [7]. It is possible to exploit this link to allow for a useful transfer of information between the phases

Directed Selection Optimization (DSO) exploits co-operation between the phases of construction and optimization. Information gathered and used, in the form of a weighted list, during the construction phase is used to influence and direct examination selection within the subsequent improvement phase. In the improvement phase the weighted list is split into portions, and using reinforcement learning techniques, the portions which show the greatest potential for improvement are preferentially used to influence examination selection. Highest Soft Constraint Optimization (HSCO) is a new optimization heuristic, where examination selection is directed by a weighted list, the values of which are calculated based on an examinations individual soft constraint penalty. Optimization occurs in order from the examination with the highest to lowest penalty.

The remainder of the paper is as follows: Section 2 briefly describes the examination timetabling problem. Section 3 describes the Squeaky Wheel constructor, Directed Selection Optimizer and the Highest Soft Constraint Cost Optimizer. Section 4 describes the experimental environment and time parameters used during experimentation. Section 5 presents and discusses the results and finally section 6 concludes the paper with a brief discussion on the effectiveness of the technique and potential future research areas

2 The Examination Timetabling Problem

Examination timetabling is a subset of the general timetabling problem, and has been proven to be NP-hard [12]. Examination timetabling involves allocating a set of events (exams), into a number of available resources (timeslots and rooms), subject to a series of constraints. Primarily, there are two types of constraints; hard and soft. Hard constraints must be satisfied for a timetable to be considered feasible, for example an exam cannot be scheduled in a room that is too small for the size of the exam, or a student must not have two exams at the same time. Soft constraints on the other hand represent desirable preferences, which are not required to be satisfied for the timetable to be considered feasible, but may affect the fitness or quality of the resultant solution. For example, while it may not be preferable for a student to have two exams in one day, a timetable can still be considered feasible if this does occur. The main goal when solving this problem is to minimize the number of soft constraint violations, while at the same time maintaining a feasible solution. As it is possible to assign a numeric value to the quality of a timetable based on how well it satisfies the various constraints, it is possible to directly compare two timetables, where the timetable with the lower overall penalty is considered the more acceptable solution.

Examination timetabling, unlike Course Timetabling, is overwhelmingly considered to be a post-enrolment problem. Student enrolment data is generally known at the time of scheduling, allowing for an accurate use of the available resources during the examination period.

The requirements for real-world examination timetabling problems are often unique for each individual institution, with the type and mix of hard and soft constraint options reflecting the preferences of the institution in question. However it is possible to identify a common set of both hard and soft constraints for benchmark and research use.

Carter, et al. introduced a set of 13 benchmark examination datasets in 1996 [5], drawn from three Canadian high schools, five Canadian universities, one American university, one British university and one university in Saudi Arabia. These datasets have been widely tested and used in examination timetabling research [2]. These datasets were supplemented by a series of new datasets, drawn from anonymised data provided by several institutions worldwide, for the 2007 International Timetabling Competition (ITC2007).

	Exams	Students	Periods	Rooms	Conflict Density	Period Hard Constraints	Room Hard Constraints
Exam 1	607	7891	54	7	5.05%	12	0
Exam 2	870	12743	40	49	1.17%	12	2
Exam 3	934	16439	36	48	2.62%	170	15
Exam 4	273	5045	21	1	15.00%	40	0
Exam 5	1018	9253	42	3	0.87%	27	0
Exam 6	242	7909	16	8	6.16%	23	0
Exam 7	1096	14676	80	15	1.93%	28	0
Exam 8	598	7718	80	8	4.55%	20	1
Exam 9	169	655	25	3	7.84%	10	0
Exam 10	214	1577	32	48	4.97%	58	0
Exam 11	934	16439	26	40	2.62%	170	15
Exam 12	78	1653	12	50	18.45%	9	7

Table 1 ITC 2007 Dataset Information

Table 1 lists the main characteristics for each of the examination datasets provided by the organizers of ITC2007. The conflict density is a measure of the number of examinations that are in conflict due to student enrolment, defining how tightly the problem is constrained by student module choice. It is initially observed that the conflict density for most of the datasets is quite low, which is reflective of the amount of choice available to students within a modern curriculum, with a large variation in course or subject choices between each student. The measure of problem size, based on the number of exams and students, varies across the datasets. The largest exam dataset could be argued to be either Exam 3/Exam 11 or Exam 7 and the smallest to be either Exam 9 or Exam 12. The amount of periods and rooms available will also have a measurable effect on the difficulty of constructing a feasible solution. Exam 3 and Exam 11 are almost identical, however Exam 11 has a much smaller set of period resources available. The differences between Exam 3 and Exam 11 reflect a "real-world" situation where an examination session has been shortened to minimize space and staff costs, while keeping all other existing constraints where possible.

Recent attempts to solve the examination timetabling problem continue to involve a variety of different techniques. Genetic Algorithms [13] are modelled on Darwins theory of evolution. Once an initial population has been constructed, it is refined over a series of iterations, with an evaluation function calculating the fitness of each individual within the population. Late Acceptance Hyper-heuristics were introduced by Burke and Bykov [14]. Traditionally, the approach in hyper-heuristics was to compare the current solution with the solution immediately preceding within the neighbourhood search process. In late acceptance, the current solution is compared with what was the current solution a number of iterations previously. Late acceptance techniques are able to produce competitive results in a short timeframe. Reinforcement Learning [15] techniques are used to influence heuristic selection for hyper-heuristics. A memory log of heuristic actions is kept during execution, with successful actions being rewarded and unsuccessful actions being punished. With this log, successful actions are chosen more often and unsuccessful actions are chosen less often across the search space. Both long term [15] and short term [16] memories have been explored in this technique. Tabu Search [17], [18] is a local search based technique. Unlike other such techniques, it maintains a list of solutions that have recently been visited, which is used to prevent the optimizer from repeatedly considering similar neighbourhoods, helping to avoid local optima. Hill Climbing is the simplest local search algorithm, introduced by Appleby in 1961 [19]. In Hill Climbing a candidate solution is only accepted if it has a better or equivalent fitness to the current one. Hill Climbing aims to converge quickly, but often has a final solution of relatively poor quality as it tends to get trapped in local optima. Simulated Annealing was introduced as a general optimization technique by Kirkpatrick, et al in 1983 [20]. Simulated Annealing is broadly similar to hill-climbing, however the technique is able to accept worse solutions through the use of a probability function and decreasing temperature parameter. Great Deluge was introduced by Dueck [21] in 1990 as a faster alternative to Simulated annealing. Great Deluge uses a boundary condition, rather than a probability function for the acceptance of worse solutions. In Great Deluge the boundary is initially set slightly higher than the initial solution, and is reduced gradually throughout the improvement process. The Extended Great Deluge was introduced by McMullan [22] for Course Timetabling, and later for Examination Timetabling [23]. The Extended Great Deluge algorithm adds a reheat mechanic similar to that employed in Simulated Annealing, where after a period of non-improvement the Great Deluge algorithm would self-terminate, the Extended Great Deluge employs a reheat function to widen the boundary condition to allow for the further acceptance of worse solutions in an attempt to escape local optima. Traditional problem solvers have primarily been implemented as single threaded applications. Modern desktop and server hardware are highly optimized for parallel workloads, and previously implemented solvers no longer take full advantage of the available hardware when executed on these machines. Ant Algorithms, introduced by Dorigo [10] and implemented for the examination timetabling problem by Eley [24], were among the first parallel implementations to solve the problem. Each ant works concurrently and independently to build a complete, or partial solution starting from an initial state defined by problem dependent criteria. The Scatter Search meta-heuristic has recently been implemented to execute in a parallel and distributed manner [25] over a series of independent servers.

3 Directed Examination Selection

Directed Selection, introduced in[26], is extended here to encompass a three phase process, building upon elements used in the Extended Great Deluge (EGD) algorithm introduced by McMullan [22]. The first phase is a Squeaky Wheel (adaptive) constructor, which is used to construct a series of initial timetables. Once construction has completed the best timetable and the weighted list used during construction is passed into the Directed Selection Optimization (DSO) phase. DSO utilizes the weighted list to influence the selection of the examination for optimization. After a number of non-improving reheat actions, the current best timetable is passed to the Highest Soft Constraint

Cost Optimization phase. When complete, the timetable is returned to the DSO phase while there is remaining execution time.

```
Start timer;
Read examination file;
Build clash information;
Squeaky Wheel Construction;
while time remaining do
    Directed Selection Optimization;
    Highest Soft Constraint Optimization;
end
Output results;
```

Algorithm 1: Sequence of Execution

3.1 Squeaky Wheel Construction

Squeaky Wheel (adaptive) construction [27] is an iterative construction process, building an initial schedule by placing one exam at a time, in the order determined by a weighted sequence. There are a number of different methods for determining the initial order of the weighted list, the technique presented here calculates the initial ordering based on examination size and the number of conflicts. Each exam is assigned to the first available time and room combination, where possible, ensuring that a feasible solution is maintained while minimizing soft constraint violations. If an exam cannot be scheduled in the current iteration, it is left unscheduled and the constructor moves onto the next exam. When an exam is scheduled a weighting based upon its current penalty, as defined by the various soft constraint violations, is added to the stored weighting in the weighted list. If an exam cannot be scheduled a suitably large weighting is used instead. Once an attempt has been made to schedule all exams, the weighted list is re-sorted and subsequently those exams with the highest weighted value (or most difficult exams) are first to be scheduled on the next iteration or "run" of the constructor. The weightings held in the list evolve over the duration of the entire construction process.

```
Read in the problem file into memory and build conflict and suitability matrices;
Calculate an initial weighting based on pre-defined criteria;
while stopping criteria not met do
    foreach exam ei in weightedList do
        for all suitable timeslots ti of ei do
            if CanSchedule(ei, ti) then
                best penalty and store best (bestti);
            end
        end
        if multipleBest found then
            Schedule(ei, randomBestti) and store associated weighting in
            weightedList;
        end
        else if bestTi found then
            Schedule (ei, bestti) and store associated weighting in weightedList;
        end
        else
            Leave exam unscheduled and add large weighting in weightedList;
        end
    end
    Sort(weightedList);
end
```

Algorithm 2: Squeaky-wheel (Adaptive) construction

3.2 Directed Selection Optimization

Directed Selection Optimization (DSO) is a new technique introduced by Hamilton-Bryce, McMullan and McCollum [26]. It is based on the premise that there exists a link where useful information can be passed from an adaptive based construction phase to an EGD based optimization phase [7]. It was shown in [26] that information which is traditionally discarded during the construction phase can be fed in to the optimization phase to influence and direct the selection of examinations for the neighbourhood search process. In Directed Examination Selection, the traditionally random selection of examinations is augmented with a portion of the weighted list generated during construction. After an initial learning period, reinforcement learning is used to influence examination selection to the portion(s) of the weighted list which show the greatest amount of improvement.

For example, the weighted list can be split into quarters. After the initial learning period the reinforcement learning list has the values (1, 4, 10, 7) for the first, second, third, and fourth quarters respectively, where 0 represents no improvement, and higher values represent greater amounts of improvement. While the highest value represents the potential for the greatest amount of improvement, it should not be used exclusively. Correspondingly while the lowest value represents the least potential for improvement, it should not be excluded from use. Therefore an element of chance should be introduced by simulating dice rolls, with the highest improving portion having approximately a 50% chance of being used, with each next portion of the weighted list having a lesser chance. Finally if no portion of the weighted list has been selected then a number of random examinations equal to the portion size of the weighted list are selected. Through experimentation it was found that sorting the reinforcement learning list approximately every ten seconds provided good performance while also ensuring that the values in the list do not become 'stale' as the improvement process continues. As in [26], for performance reasons the weighted list is sorted during a reheat action. It is possible to sort the reinforcement learning list more often than the weighted list, due to the relative size differences between the lists. During initial experimentation it was found that it is possible to over-influence the selection of examinations to the detriment of the optimization process. This is prevented by applying a simulated 'die roll' to determine whether to use specific portions of the weighted list, or default to selecting examinations at random.

```
Sort Reinforcement Learning List (rlList);
if rnd.Next (1,6) >= 3 then
    Use best portion of weightedList;
else if rnd.Next (1, 6) >= 4 then
    Use next best portion of weightedList;
else if rnd.Next (1,6) ¿= 5 then
    Use next best portion of weightedList;
else if rnd.Next (1,6) == 6 then
    Use next best portion of weightedList;
else
    Select random exams for optimization;
end
```

Algorithm 3: Influencing Examination Selection

Once examination selection has occurred, the remainder of the optimization process is similar to the EGD algorithm. On each iteration, one of two neighbourhood heuristics is selected; either move or swap, and an attempt is made to apply the chosen heuristic to the selected examination list. In the new technique, boundary acceptance has been replaced with Late Acceptance Criteria. Late Acceptance Criteria was introduced by Burke and Bykov [28] [14], wherein a candidate solution is compared for acceptance against the current solution a number of iterations previously. This can be implemented as a simple queue structure of a predefined size, wherein the current solution is compared against the head value. At each iteration the head value is removed, if the candidate solution is accepted its cost is inserted onto the end of the queue, and if the candidate solution is rejected the last accepted cost is inserted onto the end of the queue. At the beginning of the optimization process the entire queue is initialised to the value of the initial cost function of the timetable undergoing optimization. The reheat mechanic from the EGD algorithm has been retained, to allow the algorithm an attempt to escape from local optimum conditions. Finally as with the EGD algorithm, the process will self-terminate when a lack of improvement has been observed for a specified number of reheats of the late acceptance list.

```
Set the initial solution s using a construction heuristic;
Calculate initial cost function f(s); Initialize Late Acceptance list (laList);
while stopping criteria not met do
    Select portion of weightedList (optList) to use based on Reinforcement Learning
    criteria or a random exam;
    Select neighbourhood Heuristic S*;
    for all exams in optList do
        Apply S* on exam;
        Calculate f(s*);
        if f(s*) <= f(s) or f(s*) <= laList.FirstItem then
            Accept s = s*;
            Add new f(s) to laList;
            Update Reinforcement Learning Criteria with success;
        else
            Add existing f(s) to laList;
            Update Reinforcement Learning Criteria with fail;
    end
    if no improvement in given time T then
        Increase all values in laList by 10%;
end
```

Algorithm 4: Directed Selection Optimization (DSO)

3.3 Highest Soft Constraint Optimization

Highest Soft Constraint Optimization (HSCO) is a new optimization heuristic introduced here influenced by the Highest Cost construction heuristic introduced by Pillay and Banzhaf [29] [13]. The Highest Cost construction heuristic calculates the soft constraint cost of scheduling an examination given the current state of the timetable and the examination with the highest cost is scheduled first. In HSCO, the soft constraint penalty for each examination in the timetable is calculated. An attempt is then made to optimize the timetable in order from the examination with the highest soft constraint cost. As with the previous optimization phase, HSCO has been implemented with Late Acceptance criteria for boundary acceptance. During initial experimentation it was found that the neighbourhood swap heuristic used during the DSO phase resulted in degraded performance when used in the HSCO phase as it resulted in an examination being revisited multiple times during the search process. As such the HSCO phase contains only a neighbourhood move heuristic.

```
Set the initial solution s using a construction heuristic;
Improve the initial solution s using DSO;
Calculate cost function f(s);
Initialize Late Acceptance list (laList);
while stopping criteria not met do
    for all exams in timetable do
        Calculate soft constraint penalty on individual exam basis;
        Store penalty in scList;
    end
    Sort scList by penalty;
    for all exams in scList do
        Apply neighborhood move on exam;
        Calculate f(s*);
        if f(s*) <= f(s) or f(s*) <= laList.FirstItem then
            Accept s = s*;
            Add new f(s) to laList;
        else
            Restore last best s;
            Add existing f(s) to laList;
        end
    end
end
```

Algorithm 5: Highest Soft Constraint Optimization (HSCO)

4 Experimental Environment

The algorithm was implemented and tested on a PC with an Intel Xeon E5-1603 2.8GHz processor, 8GB RAM and Windows 7. The program was coded in C# targeting the .NET Framework 4.5. For each problem set, the program was executed for ten iterations, with a 240 second time limit per iteration determined by a benchmarking application released by the competition organizers. During initial experimentation it was found that allowing adaptive construction to execute for approximately 10% of the total execution time provided the best results with the new code.

5 Results and Analysis

The random seed used for generation of the results below has been recorded to ensure repeatability of the experiments. Initial experimentation identified that splitting the weighted list into sixths provided a greater improvement than any larger split, as well as good performance overall for the new optimization phases.

As with the EGD, the chosen neighbourhood search heuristics have been kept deliberately simple. While more complex heuristics can identify the optimal move, under previous experimentation these were found to have the effect of directing the search too intensively, resulting in more frequent local optimum situations [7]. The use of relatively simple search heuristics ensures that the process is not protracted by time consuming explorations of the search space. For reference, SD is the Standard Deviation.

	Exam 1		Exam 2		Exam 3		Exam 4	
	EGD	**DS**	EGD	**DS**	EGD	**DS**	**EGD**	DS
Worst	5865	5405	495	430	10909	10813	24405	22464
Best	5377	5186	435	405	10236	9399	19171	19031
Average	5598.5	5302.1	454.6	418.1	10444.4	10036.6	20241	20531.3
SD	198.9630	71.5766	17.9580	8.9747	201.5403	496.6016	1516.1996	1241.7174

Table 2a Results for Exams 1 to 4 using ITC 2007 time limit

	Exam 5		Exam 6		Exam 7		Exam 8	
	EGD	DS	**EGD**	DS	EGD	**DS**	EGD	**DS**
Worst	3349	3337	26465	26575	4688	4219	8669	7704
Best	3090	3117	25940	26055	4475	3997	8050	7303
Average	3199.7	3236.8	26240	26253.5	4567.8	4115.7	8353.5	7555.1
SD	75.1059	75.6333	157.9557	166.9340	60.1956	69.6867	177.7734	135.4035

Table 2b Results for Exams 5 to 8 using ITC 2007 time limit

	Exam 9		Exam 10		Exam 11		Exam 12	
	EGD	**DS**	EGD	**DS**	EGD	**DS**	EGD	**DS**
Worst	1185	1124	15805	15940	132483	34773	5871	5564
Best	1049	1048	14636	14789	31080	30311	5311	5369
Average	1113.6	1089.8	15332.2	15167.9	68217.8	31415.1	5596.9	5464.4
SD	41.6445	24.0361	344.1898	413.5965	39829.9124	1339.0058	151.0875	63.9083

Table 2c Results for Exams 9 to 12 using ITC 2007 time limit

Tables 2a, 2b and 2c compare the new Directed Selection technique against the Extended Great Deluge algorithm. The new technique is able to produce better results than the EGD algorithm in nine of the twelve datasets. For Exams 1, 2, 8, 9, 11 and 12 the new Directed Selection (DS) technique is also able to produce more consistent results, as measured by the standard deviation, than those generated with the EGD algorithm.

Exams 4, 5 and 6 are the only instances where the original EGD algorithm outperforms DS. These specific instances have the lowest number of room and time combinations, and as such there is relatively small freedom of movement during the optimization process. This affects both the DSO and HSCO phases due to the nature of the neighbourhood heuristics involved. Due to the smaller freedom of movement, and the more focused examination of the search space, fewer improvements are identified. EGD is not affected as much due to its exclusive use of stochastic selection, resulting in a greater examination of the whole search space.

	Directed Selection		Other Techniques			
	Best	Average	Müller ITC 2007 [30]	Adaptive Linear Combination [31]	Graph Colouring [9]	Multistage Algorithmic [32]
Exam 1	5186	5302.1	**<u>4370</u>**	5231	6234	5814
Exam 2	405	418.1	400	433	**<u>395</u>**	1062
Exam 3	9399	10036.6	10049	**<u>9265</u>**	13002	14179
Exam 4	19031	20531.3	18141	**<u>17787</u>**	17940	20207
Exam 5	3117	3236.8	**<u>2988</u>**	3083	3900	3986
Exam 6	**<u>26055</u>**	26253.5	26585	26060	27000	27755
Exam 7	**<u>3997</u>**	4115.7	4213	10712	6214	6885
Exam 8	**<u>7303</u>**	7555.1	7742	12713	8552	10449
Exam 9	1048	1089.8	**<u>1030</u>**	1111	N/A	N/A
Exam 10	**<u>14789</u>**	15167.9	16682	14825	N/A	N/A
Exam 11	30311	31415.1	34129	**<u>28891</u>**	N/A	N/A
Exam 12	**<u>5369</u>**	5464.4	5535	6181	N/A	N/A

Table 3 Comparison of best results and other techniques that keep competition time limits

Table 3 compares the new Directed Selection technique against those of Müller and other recently published research that utilizes the competition rules for time limits. Due to the prior unavailability of the hidden competition datasets, comparison results are not widely available for research that has been published post competition. While Müller's four phased technique continues to show its strength by producing the lowest penalties of all the approaches listed in four of the twelve datasets, Directed Selection is able to produce lower penalties for five of the twelve datasets. When compared to other post competition techniques, Directed Selection is able to produce significantly lower penalties for six of the eight public datasets.

	Directed Selection		Other Techniques			
	Best	Average	Extended Great Deluge [23]	Grammatical Evolution Hyper-heuristic [33]	Pursuit of Better Results Using Grid Resources [34]	Distributed Scatter Search [25]
Exam 1	5186	5302.1	4633	4362	4699	**<u>4128</u>**
Exam 2	405	418.1	405	**<u>380</u>**	385	**<u>380</u>**
Exam 3	9399	10036.6	9064	8991	8500	**<u>7769</u>**
Exam 4	19031	20531.3	15663	15094	14879	**<u>13103</u>**
Exam 5	3117	3236.8	3042	2912	2795	**<u>2513</u>**
Exam 6	26055	26253.5	25880	25735	25410	**<u>25330</u>**
Exam 7	3997	4115.7	4037	4025	3884	**<u>3537</u>**
Exam 8	7303	7555.1	7461	7452	7440	**<u>7087</u>**
Exam 9	1048	1089.8	1071	N/A	N/A	**<u>913</u>**
Exam 10	14789	15167.9	14374	N/A	N/A	**<u>13053</u>**
Exam 11	30311	31415.1	29180	N/A	N/A	**<u>24369</u>**
Exam 12	5369	5464.4	5693	N/A	N/A	**<u>5095</u>**

Table 4 Comparison of best results and other techniques that do not use ITC 2007 time limits

Table 4 compares the new Directed Selection technique against those of recently published research that do not utilize the competition rules for time limits. While the technique understandably is out preformed across all of the data sets due to the significantly shorter time limit used, the results produced remain competitive. It is worth noting that the difference between the best recorded penalties and the technique presented here is small when considering the difference in execution time; 4 hours for Distributed Scatter Search and 240 seconds for Directed Selection.

6 Conclusion

This paper presents a new optimization technique which can successfully utilize information gathered during adaptive construction to direct and influence the selection of examinations used in the neighbourhood search process, augmenting the traditional stochastic selection method, as well as a new optimization heuristic inspired by the Highest Cost construction heuristic. These techniques have successfully been used for solving the Examination Timetabling Problem as described in the second International Timetabling Competition, ITC 2007. The combined technique presented has not been tailored specifically for solving this problem, and could be adapted for solving other problem areas. In addition to testing against other benchmark datasets, the effectiveness of the technique to solve other timetabling and scheduling problems will be investigated in future work.

Traditional scheduling techniques have primarily been implemented as single threaded applications. Over the past five years, there has been a significant increase in the availability of multi-core processors, and it is now common for modern desktop and laptop computers to contain processors which have multiple physical cores and are highly optimized for parallel processing. Due to this hardware shift, traditional schedulers no longer take full advantage of the underlying hardware. Future work will look into exploiting parallelism inherent in modern computing. While the Directed Selection technique presented here does not extend itself easily to traditional parallelism, this capability of modern processors can be exploited in other ways. Work is currently involved in identifying how multiple threads or processes working independently on a single problem, while also sharing useful information about the nature of the underlying problem, can be exploited to further improve upon the optimization process.

References

1. B. McCollum, “A perspective on bridging the gap between theory and practice in university timetabling,” *Practice and Theory of Automated Timetabling VI*, vol. 3867, pp. 3–23, 2007.
2. R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee, “A survey of search methodologies and automated system development for examination timetabling,” *Journal of Scheduling*, vol. 12, no. 1, pp. 55–89, Oct. 2008.
3. S. Kristiansen and T. R. Stidsen, “A Comprehensive Study of Educational Timetabling - a Survey,” *Department of Management Engineering, Technical University of Denmark*, no. November, 2013.

4. B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. 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, no. 1, pp. 120–130, May 2009.
5. M. W. Carter, G. Laporte, and S. Y. Lee, "Examination Timetabling : Algorithmic Strategies and Applications," *The Journal of the Operational Research Society*, vol. 47, no. 3, pp. 373–383, 1996.
6. B. McCollum, P. Mcmullan, T. Müller, and A. J. Parkes, "Next Steps for the Examination Timetabling Format and Competition," *Proceedings of PATAT 2012*, pp. 418–420, 2012.
7. E. K. Burke, G. Kendall, B. McCollum, and P. Mcmullan, "Constructive versus improvement heuristics: an investigation of examination timetabling," *3rd Multidisciplinary International Scheduling Conference: Theory and Applications*, pp. 28–31, 2007.
8. E. Burke and J. Newall, "Solving Examination Timetabling Problems through Adaption of Heuristic Orderings," *Annals of Operations Research*, vol. 129, no. 1-4, pp. 107–134, Jul. 2004.
9. N. R. Sabar, M. Ayob, R. Qu, and G. Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Applied Intelligence*, vol. 37, no. 1, pp. 1–11, Aug. 2011.
10. M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant Algorithms for Discrete Optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, Apr. 1999.
11. J. H. Obit, D. Ouelhadj, D. Landa-Silva, and R. Alfred, "An Evolutionary Non-Linear Great Deluge Approach for Solving Course Timetabling Problems," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 4, pp. 1–13, 2012.
12. T. B. Cooper and J. H. Kingston, "The Complexity of Timetable Construction Problems," *Lecture Notes in Computer Science*, vol. 1153, pp. 281–295, 1996.
13. N. Pillay and W. Banzhaf, "An informed genetic algorithm for the examination timetabling problem," *Applied Soft Computing*, vol. 10, no. 2, pp. 457–467, Mar. 2010.
14. E. K. Burke and Y. Bykov, "A late acceptance strategy in hill-climbing for exam timetabling problems," *PATAT 2008 Conference, Montreal, Canada*, 2008.
15. E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 39–59, Jan. 2010.
16. R. Bai, J. Blazewicz, E. K. Burke, G. Kendall, and B. McCollum, "A simulated annealing hyper-heuristic methodology for flexible decision support," *4OR*, vol. 10, no. 1, pp. 43–66, Nov. 2011.
17. E. Ikonomovska, I. Chorbev, D. Gjorgjevik, and D. Mihajlov, "The Adaptive Tabu Search and Its Application to the Quadratic Assignment Problem," in *9th International Multiconference Information Society 2006*, M. Bohanec, M. Gams, V. Rajkovič, T. Urbančič, M. Bernik, D. Mladenić, M. Grobelnik, M. Heričko, U. Kordeš, O. Markič, J. Musek, M. Osredkar, I. Kononenko, and B. Novak Škarja, Eds., 2006, pp. 26–29.
18. S. Abdullah and H. Turabieh, "On the use of multi neighbourhood structures within a Tabu-based memetic approach to university timetabling problems," *Information Sciences*, vol. 191, pp. 146–168, May 2012.
19. J. S. Appleby, D. V. Blake, and E. A. Newman, "Techniques for producing school timetables on a computer and their application to other scheduling problems," *The Computer Journal*, 1961.
20. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing." *Science (New York, N.Y.)*, vol. 220, no. 4598, pp. 671–80, May 1983.
21. G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, vol. 90, no. 1, pp. 161–175, Sep. 1990.
22. P. Mcmullan, "An extended implementation of the great deluge algorithm for course timetabling," *Computational ScienceICCS 2007*, vol. 4487, pp. 538–545, 2007.
23. B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke, and S. Abdullah, "An extended great deluge approach to the examination timetabling problem," *Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications*, no. August, pp. 424–434, 2009.

24. M. Eley, “Ant algorithms for the exam timetabling problem,” *Practice and Theory of Automated Timetabling VI*, vol. 3867, pp. 167–180, 2006.
25. C. Gogos, G. Goulas, P. Alefragis, V. Kolonias, and E. Housos, “Distributed scatter search for the examination timetabling problem,” in *Proceedings of PATAT 2010*, 2010, pp. 211–223.
26. R. Hamilton-Bryce, P. McMullan, and B. McCollum, “Directing selection within an extended great deluge optimization algorithm,” *Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2013)*, 2013.
27. D. P. Clements and D. E. Joslin, “Squeaky Wheel Optimization,” *Journal Of Artificial Intelligence Research*, vol. 10, pp. 353–373, May 1999.
28. E. Ozcan, Y. Bykov, M. Birben, and E. K. Burke, “Examination timetabling using late acceptance hyper-heuristics,” in *2009 IEEE Congress on Evolutionary Computation*. IEEE, May 2009, pp. 997–1004.
29. N. Pillay and W. Banzhaf, “A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem,” *European Journal of Operational Research*, vol. 197, no. 2, pp. 482–491, Sep. 2009.
30. T. Müller, “ITC2007 solver description: a hybrid approach,” *Annals of Operations Research*, vol. 172, no. 1, pp. 429–446, Oct. 2009.
31. S. Abdul Rahman, A. Bargiela, E. K. Burke, E. Özcan, B. McCollum, and P. McMullan, “Adaptive linear combination of heuristic orderings in constructing examination timetables,” *European Journal of Operational Research*, vol. 232, no. 2, pp. 287–297, Jan. 2014.
32. C. Gogos, P. Alefragis, and E. Housos, “An improved multi-staged algorithmic process for the solution of the examination timetabling problem,” *Annals of Operations Research*, vol. 194, no. 1, pp. 203–221, Feb. 2010.
33. N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, “Grammatical Evolution Hyper-heuristic for Combinatorial Optimization problems,” *IEEE Transactions on Evolutionary Computation*, no. October, pp. 1–22, 2013.
34. C. Gogos, G. Goulas, P. Alefragis, and E. Housos, “Pursuit of better results for the examination timetabling problem using grid resources,” in *2009 IEEE Symposium on Computational Intelligence in Scheduling*. IEEE, Mar. 2009, pp. 48–53.

A Multi-Phase Hybrid Metaheuristics Approach for the Exam Timetabling

Ali Hmer and Malek Mouhoub

Department of Computer Science
University of Regina
Regina, Canada
{hmer200a,mouhoubm}@cs.uregina.ca

Abstract. We propose a Multi-Phase Hybrid Metaheuristics approach for solving the Exam Timetabling Problem (ETP). This approach includes a pre-processing phase, a construction phase and an enhancement phase. The pre-processing phase involves two stages: the propagation of ordering constraints and implicit constraints discovery stages. The construction phase uses a variant of the Tabu search with conflicts dictionary. The enhancement phase includes Hill Climbing (HC), Simulated Annealing (SA) and our updated version of the extended "Great Deluge" algorithm. In order to evaluate the performance of the different phases of our proposed approach, we conducted several experiments on instances taken from the ITC 2007 benchmarking datasets. The results are very promising and competitive with the well known ETP solvers.

Keywords: Timetabling, Constraint Optimization, Metaheuristics.

1 Introduction

The examination timetabling [1, 2], is an annual or semi-annual problem for educational institutions. Due to its complexity and practicality, it is extensively studied by researchers in operational research and artificial intelligence. Many approaches have been proposed and discussed for solving the problem [1–7] using one or a combination of some of the following methods: graph-based, sequential techniques, clustering-based techniques, constraint-based techniques, metaheuristics, hyper-heuristics, multi-criteria techniques, and case-based reasoning techniques. In this paper we propose a Multi-Phase Hybrid Metaheuristics approach for solving the Exam Timetabling Problem (ETP). This approach consists of the preprocessing, construction and enhancement stages; and includes Tabu Search, Hill Climbing, Simulated Annealing and a modified version of Extended Great Deluge algorithms [8, 3] using metaheuristics techniques. The preprocessing phase is needed to prepare the work for the remaining two stages. During this phase, exams are sorted following the most constrained variables first heuristic [9] and implicit constraints are discovered using a form of transitive closure. During the construction stage a complete feasible solution is found using a variant of Tabu search along with conflicts dictionary to reduce cycling. In the enhancement phase a chosen metaheuristic is used. Once a solution can no longer be improved or reaches an idle state, another metaheuristic kicks in and used. The following metaheuristics are considered: Hill Climbing (HC) [10, 11], Simulated Annealing (SA) [12] and our updated version of the extended "Great Deluge" algorithm [8] which improves on the one proposed in [3].

In order to evaluate the performance of the different phases of our proposed approach we conducted several experiments on instances taken from the ITC 2007 benchmarking datasets [13]. The results are very promising and competitive with the well known ETP solvers. The rest of the paper is structured as follows. In the next section we will introduce the problem we are tackling. Section 3 presents our proposed solving approach. Experimental tests evaluating our solving method are then reported in Section 4. Finally, concluding remarks and future works are listed in Section 5.

2 Problem Description

2.1 Problem Formulation

Following the common formulations to the exam timetabling [14, 15] we model this problem as a constraint problem including the following.

Variable. Each exam is modeled as a problem variable defined over a domain of all possible assignments to that exam. An assignment is composed of a time period and a room.

Room Constraint. Exams are constrained by rooms seating capacity.

Student Constraint. This constraint prevents a student from being scheduled for more than one exam during a given time period.

Order constraint. This constraint is about exam ordering and precedence between two or more exams.

Same Duration Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in the same time slot.

Different Duration Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in different time slots.

Same Room Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in the same room.

Different Room Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in different rooms.

2.2 Penalty Function

The penalty function is a problem dependent generic function to calculate the total cost/value of a given solution. Each soft constraint involves a single or multiple resources and violating it has its own penalty value that should be set in the problem description. The total penalty value of any solution is the sum of penalties of all violated soft constraints in the whole exam problem. Penalties correspond to violating soft constraints including the following.

1. Two exams in a row.
2. Two exams in the same day.
3. Mixed durations where two or more exams are taking place in the same room but have different durations.
4. Room penalty where using certain rooms implies specific penalty to discourage scheduling exam to them.
5. Period penalty where assigning exam to certain periods implies specific penalty.

3 Proposed Solving Approach

Our proposed solving approach consists of the following three main phases. A pre-processing phase followed by a construction and an enhancement phases.

3.1 Pre-processing phase

The difficulty of any exam timetabling depends on three factors; the number of students that enroll in it, direct student conflicts in that exam and its scheduling priority constraint, if any, among all exams scheduling. Following the idea of most constrained variables first [9], exams with most scheduling difficulty are scheduled first. The reason for this is that if scheduled late, they would most likely increase the potential of the un-assignment process of other exams that violate some constraints which eventually causes a backtracking. The pre-processing phase consists of following two stages.

1. Exam timetabling problem collections ordering.
2. The discovery of un-specified (implicit) hard constraints.

In the following we provide the details for each stage.

3.1.1 Problem collections ordering

In this stage a process takes place for the different collections that the exam problem consists of. These collections are exams, rooms, periods and students. Exams and students are usually large collections and pre-ordering those leads to a better performance and efficient results during search. In [16, 17] two of the well-known common techniques have been proposed to describe the ordering of exams based on difficulty criteria preceding their assignment to time slots. Our approach is slightly different from these techniques. It depends on a different concept revolving around our knowledge that large exam timetabling problems contain large exams, students and resources collections, and enhancing the way that we retrieve and lookup any element in these collections is a key in any efficient search algorithm. Indeed, the time complexity of looking up or retrieving an element from unsorted large collection is $O(n)$ whereas the time complexity of the same process in a sorted collection is $O(\log n)$.

Our approach of collections ordering pre-processing stage involves the creation of four ordered collections at the time of building problem variables, values and facts collections based on the efficient Microsoft .NET framework which implements the quicksort algorithm. It is worth mentioning that quicksort makes $O(n \log n)$ comparisons in average to sort n items. Prior to our decision on whether to perform this stage or not, we thought of two issues; the time needed to build large sorted collections and the time required to lookup or retrieve any element in these collections. After examining the different types of collections that Microsoft .NET framework provides, we made the decision to use Sorted Generic Lists for all variables, values and constraints collections. There are two reasons for that. First, we only need to build them once at the beginning of the problem modeling and hence we produce them in a sorted manner. This is the only time we spent to sort them. They also do not consume a lot of memory as other types of collections. In fact, they are the least memory consuming collection. The second reason is that after building any of the problem collections, we only need to do lookups which a generic list is good at and one of the fastest collections for that matter and its time complexity is $O(\log n)$. Table 1 shows the time complexity for adding an element and for looking up or retrieving an element from both unordered and ordered collections. Although, we include the cost of removing an element as in all large collections, as we only build any collection once and lookup or retrieve afterward and there is no need for removals. In the case of collections that need items removals we use hash sets as the cost of removing an item is considerably small.

Table 1. Time Complexity for Ordered and Unordered Collections

	Adding elt	Lookup/Retrieval	Removing elt
Unordered Collection	$O(1)$	$O(n)$	$O(n)$
Ordered Collection	$O(n)$	$O(\log n)$	$O(n)$

3.1.2 Discovery of implicit hard constraints

In this stage we have developed a technique to discover all hard constraints that were not explicitly defined in the problem. In any large COP problem that contains a large collection of variables, values and constraints, there is always the possibility of missing some of the hard constraints that depend on some of the declared ones. Our approach is to provide a pre-processing stage that discovers these unspecified constraints and add them to the problem constraints collection. In fact our goal is to add other constraints that should be known before assigning a value to a variable which in essence might eliminate some of the variables domain values and hence preventing a backtracking process, which would occur later on, if these additional constraints were not specified.

The exam timetabling problem usually contains the following three types of exam based constraints [14, 15].

Exam Ordering. Two or more exams scheduling should appear in a particular order. For example: exam 1 should take place after exam 2.
Exam Coincidence. Two or more exams should take place at the same time.
Exam Exclusion. Two or more exams should take place at different times.

This pre-processing stage is based on creating three full graphs for each type of these constraints where nodes represent the exams and edges are the hard constraints between exams. Then by traversing each graph, we try to discover same or different type of constraints between other exams in the same graph. The following are the four steps we use to achieve such discovery.

1. **Propagating ordering constraints that belong to concurrent exams.** For example, if a coincidence constraint declares that exam 1 must take place at the same time as exam 2 and another ordering constraint states that exam 2 must take place after exam 3. In this case, we need to add a new ordering constraint stating that exam 1 must take place after exam 3.
2. **Propagating ordering constraints by transitive closure.** For example, if exam 1 must be scheduled after exam 2 and exam 2 must be scheduled after exam 3 then this implies adding a new ordering constraint which states that exam 1 must be scheduled after exam 3.
3. **Propagating coincidence and exclusion constraints.** For example, if there is a coincidence constraint stating that exam 1 must take place in the same period as exam 2 and another distinct constraint stating that exam 2 must take place in a different period than exam 3 then a new exclusion constraint must be added to state that exam 1 and exam 3 must take place in different periods.
4. **Propagating the largest exam period of the same exam set to all other exams.** This happens when all exams are involved in the same coincidence constraint. For example, if there is a coincidence constraint involving three exams, exam 1, exam 2, and exam3, with a respective periods of 3 hours, 2 and half hours and 2 hours. Then all three exams anticipated periods should be updated to be equal to the largest (3 hours). It should be mentioned that any penalty cost that involves periods, when calculated, uses the original period and not the updated one.

3.2 Construction phase

In the construction phase a complete feasible solution is found using Tabu search metaheuristic along with conflicts dictionary to reduce cycling. Conflicts dictionary essentially is a dictionary data structure based consisting of a key and value and is used for its performance capability. Each entry in the conflicts dictionary represents a count for the number of conflicts that an assignment causes during search. In future search iterations, the entry with the highest counts are avoided and regarded as tabu. Utilizing Tabu search metaheuristics with conflicts dictionary can be further detailed as follows. As the search is only considered by variable and value selection criteria, the algorithm initially tries to find those variables that are most problematic to assign. Usually, a variable is randomly selected from unassigned variables that have the smallest domain size and less number of hard constraints. It then attempts to select the best value to assign to the selected variable using conflicts dictionary. A best value is one where its assignment improves the overall value of the solution. Also, any value that violates a fewer number of hard constraints is considered. In other words, when assigning a value to a given variable, the algorithm is looking to minimize the number of conflicting variables that need to be unassigned in order to reach or keep a solution feasible after assignment. A value is selected randomly if there is more than one value with such conditions. Soft constraints violations are totally ignored in this phase as they might affect the algorithm performance when searching for complete feasible solutions.

3.3 Enhancement phase

In the enhancement phase, a combination of three metaheuristics is employed and we can select just one, two or three out of theses metaheuristics. Whatever a metaheuristic is used, a local optimum is found. Once a solution can no longer be improved or reaches an idle state, another metaheuristic technique kicks in and is used. In our algorithm we used three of the well-known metaheuristics. These are Hill Climbing (HC) [10, 11], Simulated Annealing (SA) [12] and our Modified Extended Great Deluge (MEGD). MEGD is altered to allow some alternations of the bound that is imposed on the overall solution value. The search ends after a predetermined time limit has been reached. The best solution found within that limit is returned. Our MEGD is based on the Extended Great Deluge (EGD) [8] method which in turn is based on the original Great Deluge (GD). GD was introduced by Dueck [18] as a cure to SA requirement to find a cooling schedule for a particular instance of a given problem. GD algorithm starts with a "water level" equal to the initial solution value, and a preconfigured rate usually named "tolerance rate" to decrease that water level. The predetermined rate is the only parameter for this algorithm and this is one of this algorithm's advantages. GD accepts worsening solutions if the penalty cost is less than the water level. This latter is decreased by the pre-determined rate set for every iteration. Due to the advantage of using less parameters, GD has been used in several other implementations of metaheuristics.

The Extended Great Deluge (EGD) [8] has a construction phase followed by an improvement phase. The construction phase is applied using the existing adaptive ordering heuristic search method [19]. This latter ordering uses a weighted order list of the examinations which is to be scheduled based on soft constraints as well as the "difficulty to schedule" constraint. Once an exam is scheduled, its weight is increased based on the localized penalties it came across. The unscheduled examinations are given a considerably larger increase, based on a formulation that is based on the maximum general penalty encountered from [19]. The improvement phase starts when feasibility is achieved in the construction phase and tries to provide an improved solution. Unlike EGD, our approach is only concerned with the enhancement phase and it only tries to improve the overall value of the current feasible complete solution. Our approach is different from EGD as follows.

1. In the original GD, the tolerance value starts with the initial solution's value and decreases by a preconfigured rate. It tries to range within all neighbours of the current solution in each iteration. However, in our approach, tolerance rate ranges between values that are percentage of the current solution value; one above and one below. In our approach, we use two preconfigured values, namely tolerance lower bound and tolerance upper bound. Tolerance upper bound is a preconfigured value that defaults to $(108\%)^{iter_{idle}}$ of the initial solution. $iter_{idle}$ is a counter that starts with 1 and is incremented by 1 each time the tolerance rate is reset. Tolerance lower bound is also a preconfigured value that defaults to 92% of the initial solution. The tolerance decay rate is a predetermined rate that defaults to 99.99995%. At the beginning a tolerance rate t is assigned to a value of the initial solution. It is decreased by tolerance decay rate in each iteration. Likewise, in every iteration, a new neighbour is selected and tested against the current t and the best solution value. If it is better than either one of them, the current solution becomes the best solution and t is decayed by tolerance rate.
2. The second difference occurs at the time of taking the decision to reset the tolerance value t. Tolerance value t is reset as follows. t reaches the tolerance lower bound which as we discussed is equal to 92% (or predetermined value) of the best solution so far. We can as well reset t based on the last n (defaulted to 40) solutions if they happen to be consistent and carry the same value. This means that we are stuck in a local optimum and there is no need to complete the full cycle and reach the lower bound. Rather, we decrease the current tolerance decay rate by half the rate and restart.

Figure 1 presents the pseudo code of MEGD.

Neighbourhood selection variation is by far the most influential technique that affects rapid local search. Using more than one neighbourhood within a search provides a very effective technique of escaping from a local optimum. It is notable that if the current solution is in a local optimum in one neighbourhood, it might escape the local optimum, if assigned a different neighbourhood and can consequently be more improved using a good feasible approach. In exam timetabling, the neighbourhoods used in local search techniques largely involve moving some exams from their current time slot and/or rooms to a new time slot and/or rooms. Based on that, our implementation (corresponding to the function **selectNeighbour()** in Figure 1) uses the following seven neighbourhoods.

1. Exam Duration Move: selects a single exam randomly and move it to a different feasible time slot randomly.

```
Procedure Modified Extended Great Deluge (MEGD)
Begin Procedure
$t \leftarrow f(s) \Rightarrow$ initial tolerance boundary level, $t^* \leftarrow$ Decay Rate, should be $< 100\%$, default $= 99.99995\%$
$f \leftarrow f(s) \Rightarrow$ initial solution value, $iter_{idle} \leftarrow 0 \Rightarrow$ number of idle iteraions
$f_{Best} \leftarrow f(s), S_{Best} \leftarrow s$
$sol_{array}[\bar{n}] \leftarrow 0 \Rightarrow$ arrayof last $\bar{n}$ solutions, $n \leftarrow 0$
$t_{upper} \leftarrow$ Upper tolerance bound Rate, default $= 108\%$, $t_{lower} \leftarrow$ Lower tolerance bound Rate, default $= 92\%$
while (termination criteria not met) do
    $s^* \leftarrow$ selectNeighbour$(s, i)$
    $f^* \leftarrow f(s^*) \Rightarrow$ calculate current solution value
    if ($f^* \leq f$ or $f^* \leq t$)
        $s \leftarrow s^* \Rightarrow$ Accept,        $sol_{array}[n] \leftarrow s^*$
        $n \leftarrow n + 1$,                $f \leftarrow f^* \Rightarrow$ current solution value
        if ($f^* \leq f_{Best}$)   $S_{Best} \leftarrow s^*, f_{Best} \leftarrow f^*$
    endif
    $t \leftarrow t \times t^* \Rightarrow$ decrease boundary
    if($sol_{array}$ have $\bar{n}$ values and all are the same) $\Rightarrow$ Stuck in local optimum
        $iter_{idle} \leftarrow iter_{idle} + 0.5 \Rightarrow$ increase idle iterations by half
        $t \leftarrow (t_{upper})^{iter_{idle}} \times s \Rightarrow$ New Tolerance Boundary Level
        $sol_{array}[\bar{n}] \leftarrow 0 \Rightarrow$ reset last solutions values array
        $n \leftarrow 0$
    else
        $t_{level} \leftarrow (t_{lower})^{iter_{idle}+1} \times S_{Best} \Rightarrow$ Tolerance Boundary Level
        if ($t \leq t_{level}$)
            $iter_{idle} \leftarrow iter_{idle} + 1 \Rightarrow$ increase idle iterations
            $t \leftarrow (t_{upper})^{iter_{idle}} \times S_{Best} \Rightarrow$ New Tolerance Boundary Level
        endif
    endif
end while
return   $S_{Best}$
end procedure
```

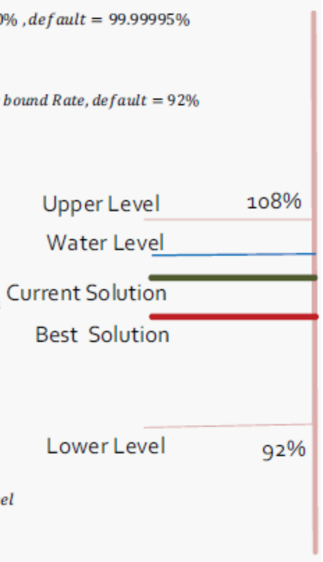


Fig. 1. Procedure Modified Extended Great Deluge (MEGD).

2. Exam Duration Swap Move: selects two exams randomly and swaps their assigned time slots.
3. Non Conflicting Assignment Move: selects an exam randomly and assigns it to a non-conflicting assignment (time slot and room) randomly.
4. Room Move: selects a single exam randomly and moves it to a different feasible room randomly.
5. Room Swap Move: selects two exams randomly and swaps their assigned rooms.
6. Exam Swap Move: selects two exams randomly and swaps their assignments (i.e. time slots and rooms).
7. Random Move: selects an exam randomly and assigns a new assignment to it randomly. The assignment consists of a room and time slot and might cause conflicts.

4 Experimentation

This section reports the experiments conducted on the well-known benchmarking datasets of the International Timetabling Competition (ITC 2007) [13]. This benchmarking datasets consists of 12 basic real world examination time tabling problems obtained from different anonymous universities around the world. The detailed properties of the 12 benchmark instances are summarized in Table 2.

Table 2. ITC 2007 Exam Track Benchmarking Datasets.

Instance #	# of students students	# of exams	# of rooms	Period & Room Hard Constraints	Constraints density	# of time slots
1	7,891	607	7	12	5.04%	54
2	12,743	870	49	14	1.17%	40
3	16,439	934	48	184	2.62%	36
4	5,045	273	1	40	14.94%	21
5	9,253	1,018	3	27	0.87%	42
6	7,909	242	8	23	6.13%	16
7	14,676	1,096	15	28	1.93%	80
8	7,718	598	8	21	4.54%	80
9	655	169	3	10	7.79%	25
10	1,577	214	48	58	4.95%	32
11	16,439	934	40	185	2.62%	26
12	1,653	78	50	16	18.21%	12

As we will see, our proposed approach is successful in competing with benchmarking results published in literature so far. We measure the general behaviour and performance of our implementation in the two different phases to solve the exam timetabling problem; construction phase and enhancement phases. We also compare our approach to the well known exam timetabling problem solvers. All the experiments are performed on an PC Intel Core 2-Duo 2.4 GHz processor with 8 GB of RAM.

4.1 Construction Phase Testing and Analysis

Our construction approach is based on Tabu Search with Dictionary Conflicts. We set our goal to get a complete feasible solution as fast as possible so that the enhancement phase can kick in and improve the overall solution value gradually. In order to measure the performance of Tabu with CD, we tested it against standard Tabu search and in both cases preprocessing phase is done prior to constructing complete solution. As known, standard Tabu algorithm prevents cycling by using a tabu list, which determines the forbidden moves. This list stores the most recently accepted moves. The inverses of the moves in the list are forbidden. Our approach differs in that we sum all the accumulated number of conflicts that a move caused rather than just moves which are considered as forbidden. We also implemented "Iteration Distance" which excludes entries that are far away from the current iteration based on configured setting for iteration distance. For the purpose of the construction phase testing, we selected dataset 4 as it has a high conflict density (14.94%) along with high number of students and exams which makes it as one of the toughest problem to solve in our benchmarking datasets.

During the process of building a complete feasible solution, we record solution value in every iteration along with its time. This testing is only concerned with the construction phase and so we set our testing to run for 10 times for each method of the selected dataset. Then we select the trial with the best solution value from the ten trials. We represent each point in the graph with the corresponding penalty cost monitored after every iteration along with its time. The last penalty cost is the cost of the first complete feasible solution and that is where the construction phases stops.

Figure 2 illustrates the full snapshot of the best trial for standard TS on dataset 1 while figure 3 shows the same pattern for TS with CD. Among 10 trials, using best run's solution value, although standard TS shows better complete solution (6041), it took 8.03 seconds and 1081 iterations to get it while TS with CD with 6803, took 4.21 and 672 iterations. Also, standard TS algorithm shows relatively higher number of fluctuations between lower penalty cost and higher ones where TS with CD seems to have gradually been building the complete solution with a minimum number of instability.

However, dataset 4 has shown a different pattern. Dataset 4 is one of the most constrained problems. The top chart of Figure 4 illustrates the full snapshot of the best trial for standard TS on dataset 4 while the bottom chart shows the same pattern for TS with CD. The top chart articulates how standard TS struggled with finding the less penalty cost solutions in contrary to TS with CD (bottom chart). Standard TS spent a total of 28.54 seconds (3281 iterations) to find a best complete solution, amongst 10 trials, with a penalty cost of 31133 while TS with CD took only 2.03 seconds (567 iterations) to find one with a penalty of 27633. That is a performance improvement of around 1400% with solution value improvement of 112%.

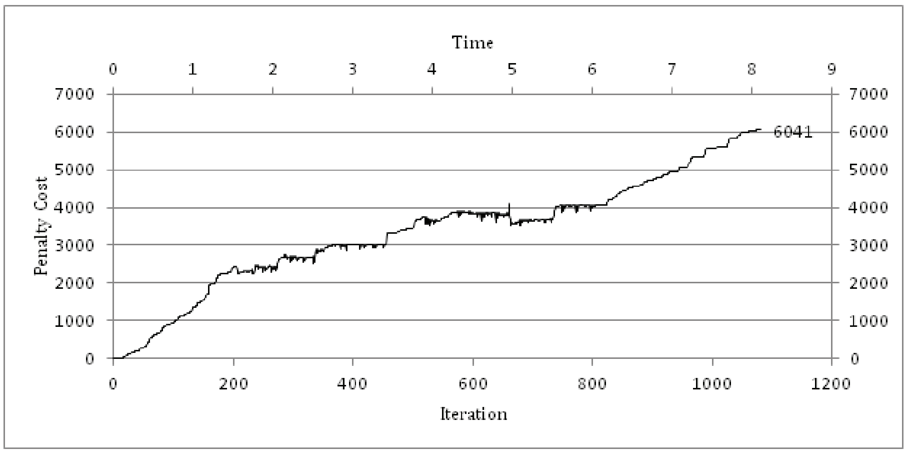


Fig. 2. Dataset 1 Construction Phase using Standard TS.

Dataset 5 is the least constrained problem with only 0.87% but with relatively high number of students and exams (9253 students and 1018 exams)

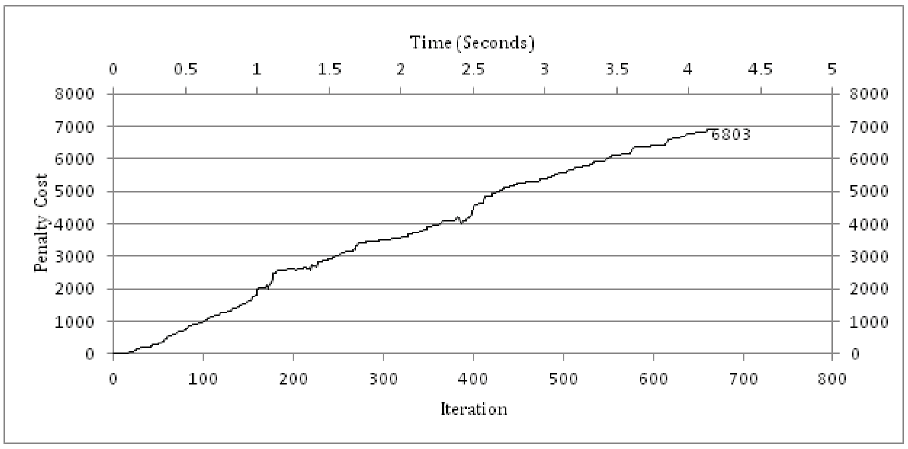


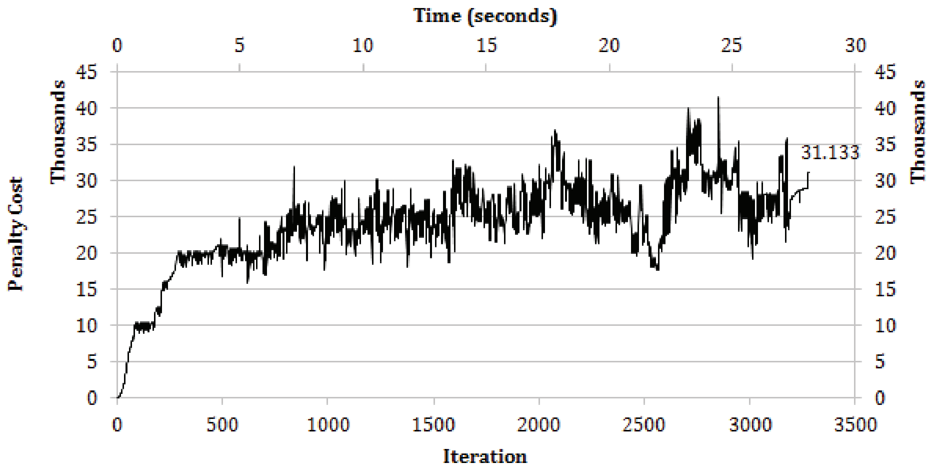
Fig. 3. Dataset 1 Construction Phase using TS with CD.

which leads us to think that it should be one of the easiest problems to solve. We can see that in the lack of any fluctuations between worse and better solutions in the graphs for the datasets in figure 5. Nonetheless, TS with CD algorithm performs slightly better than standard TS even though the problem itself tends to be easy to solve. In ten trials, standard TS obtained 6530, as a best solution value, in 2.58 seconds (1020 iterations) while TS with CD achieved 5030, as a best solution value, in 2.21 seconds (1050 iterations).

4.2 Enhancement Phase Testing and Analysis

We compare 4 methods labeled method 1, method 2, method 3 and method 4 and respectively corresponding to HC+SA, SA, EGD and our MEGD. All these methods use Tabu Search with Dictionary Conflicts in the construction phase. In addition, only methods 2, 3 and 4 have a preprocessing phase.

Figures 6, 7 and 8 shows the enhancement phase best solution distribution history for four methods against iteration in datasets 1, 6 and 8. From these figures, we can clearly notice that without preprocessing the first method tends to improve solutions values within a relatively short time and keeps improving almost very slowly. Another visible notice is that method 1 seems to use less number of iterations which suggests that it employs these iterations cycles either in backtracking or accessing non efficient collections. Method 2 which also uses SA starts enhancing a complete solution very early but then ends with slightly outperforming method 1. On the other hand, the last two methods, using GD flavours with preprocessing in place, spend some time to find the first improving solution after the first complete solution which also tends to be of, relatively, worse value than methods 1 and 2. This is due to the nature of great deluge algorithm which only accepts an improving solution. Also, a bad solution is accepted if its quality is less than (for the case of a minimization problem)



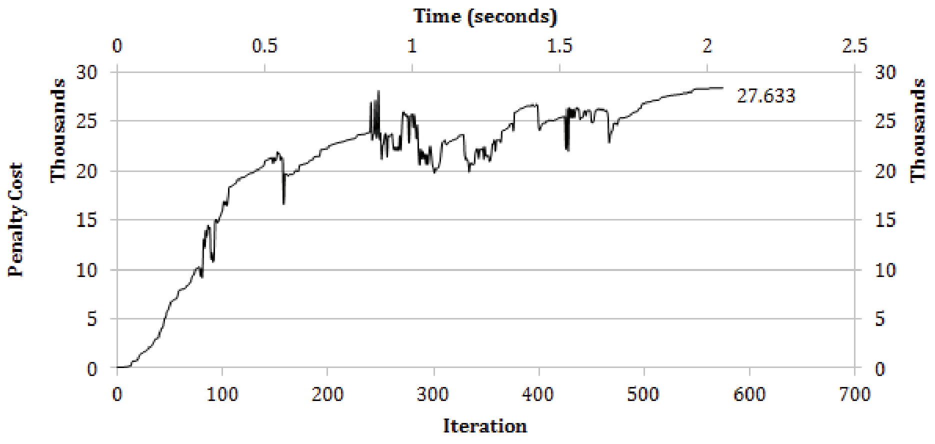
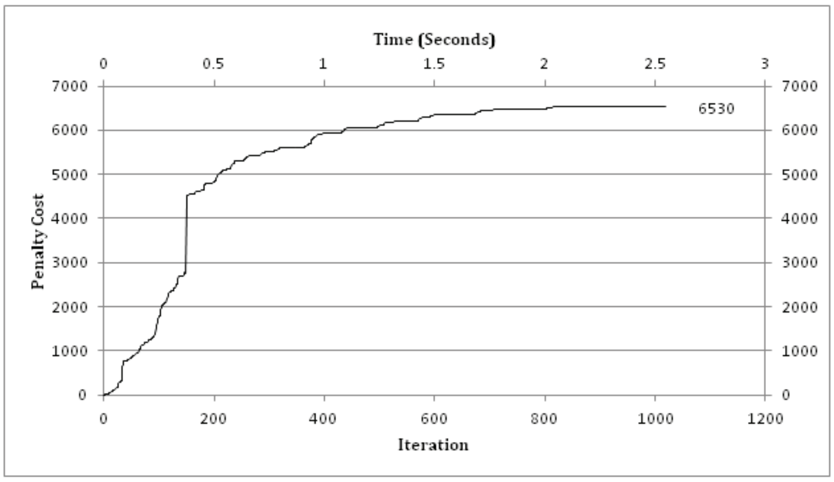


Fig. 4. Performance of the construction phase on dataset 4 with Tabu and Tabu together with conflicts dictionary.



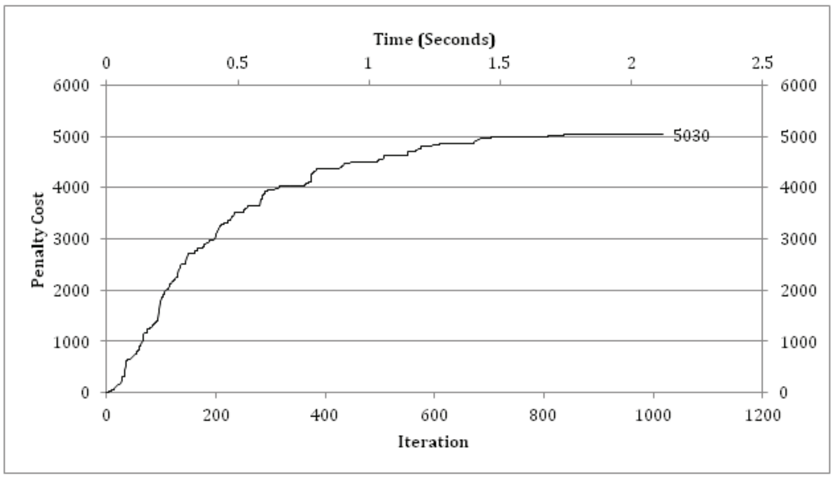


Fig. 5. Dataset 5 using Standard TS (top chart) and TS with CD (bottom chart).

or equal to an upper bound or ″level″ in which during the search process, the ″level″ is iteratively updated by a constant decreasing rate. It also means that, with the preprocessing phase in place, there will be more features. This means that there are more effort to satisfy more constraints but also gaining better performance when looking up the different collections in particular area as well as a more careful exploration. For the inclusion of preprocessing phase, our proposed search algorithm diversification of search to gather the whole search space proved the importance of finding the global minimum quickly. We also note that MEGD performs slightly better than EGD in 8 out of 12 of the datasets. Figure 6 illustrates that methods 3 and 4 were close in terms of results in achieving the best solution. This is also the case for method 1 and 2 although method 2 outperformed to some extent method 1. Method 3 reached a best solution value of 4185. The same pattern also appears in figures 7 and 8 where they show results for dataset 6 and dataset 8 where method 4 is marginally the winner in finding the best solution.

Generally, the algorithms might behave differently due to the different measurements enforced during the search process. However, the difference between SA, GD, EGD and MEGD algorithms lies in the acceptance criteria functionality that would make a difference on the limited solving time that was imposed on our benchmarking datasets. This might not be the case if we have relatively longer times for several hours or days as all these algorithms are based on the stochastic local search and there will always be the possibility of achieving good results.

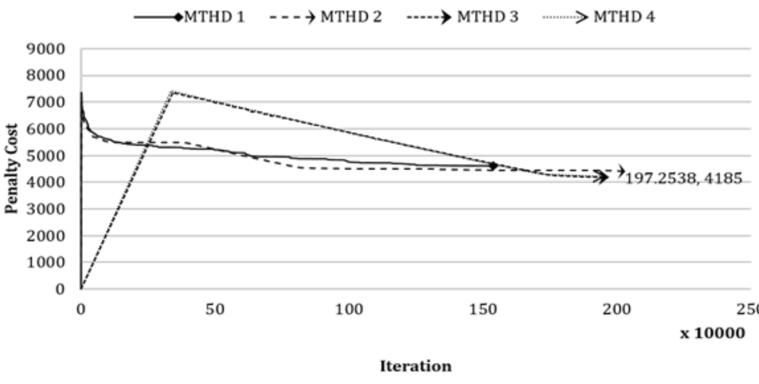


Fig. 6. Performance on dataset 1 of the enhancement phase.

4.3 Comparative Tests Results

On the basis of results obtained by both construction and enhancement phases, we decided to compare our four methods to the five well known ET solvers. Each of the datasets used in our testing phase has a previously discovered best known solution announced by ITC 2007. The five known solvers are the following finalists of the examination track of the competition.

1. Müller [20] implemented a constraint-based solver, which constructs a complete solution, followed by a hill climbing and a great deluge approach for improving the solution.
2. Gogos et al. [21] used a Greedy Randomized Adaptive Search Procedure, followed by simulated annealing, and integer programming.
3. Atsuta et al. [22] developed a constraint satisfaction solver combined with Tabu search and iterated local search.

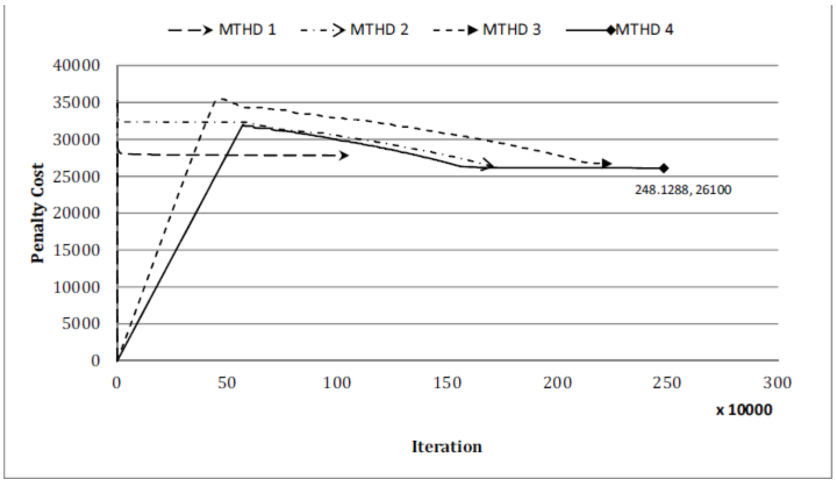


Fig. 7. Performance on dataset 6 of the enhancement phase.

4. De Smet [23] has his own solver, namely Drools Solver, which is a combination of Tabu search and the JBoss Drools rules engine used for the evaluation.
5. Finally, Pillay [24] used a heuristic method that is inspired by cell biology.

During experiment runs, we managed to achieve an outstanding 98% success in reaching complete feasible solution on all instances in all attempts. The remaining 2% were only in dataset 4 and 12.

For each method trials we performed 11 individual runs on each of the 12 competition instances, using the time limit specified by the competition benchmarking program as our stopping criteria, which equated to 362 seconds. The same timeout value on each machine is used for all of the 12 datasets. In all cases, we logged out all best solution values history along with times and iterations where these best solution values are discovered.

The settings of the algorithms have remained the same throughout the experiment for the purpose of going in line with ITC 2007 rules. One of our objectives in testing phase is to represent different algorithm variations that are composed of different algorithms and compare them to the performance of ITC 2007 results. The expectation was also set for the results to be reasonably comparable if not better than ITC 2007 exam track results.

Table 3 reports the comparative results including the best solution value (lowest penalty cost) and average of best solution values for each variant. When searching without preprocessing, performance degrades relatively to when using preprocessing phase. Only the first method did not use preprocessing and if we look first at the performance of its algorithms in comparison to ITC 2007

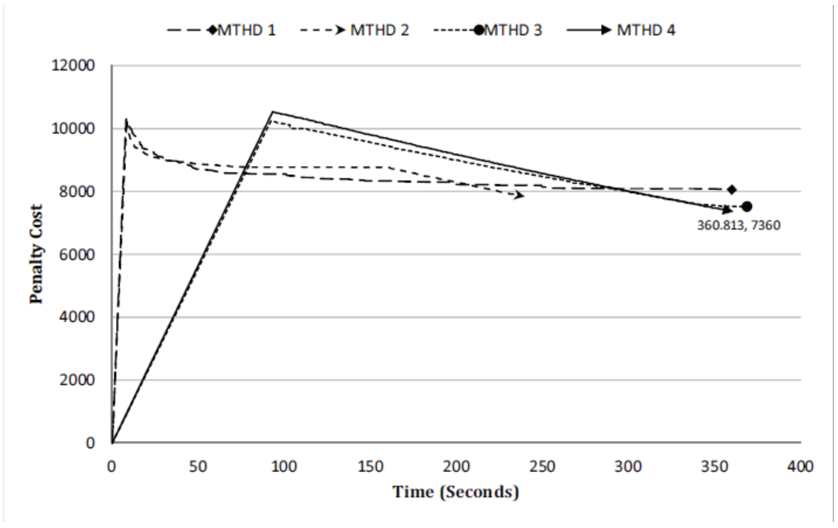
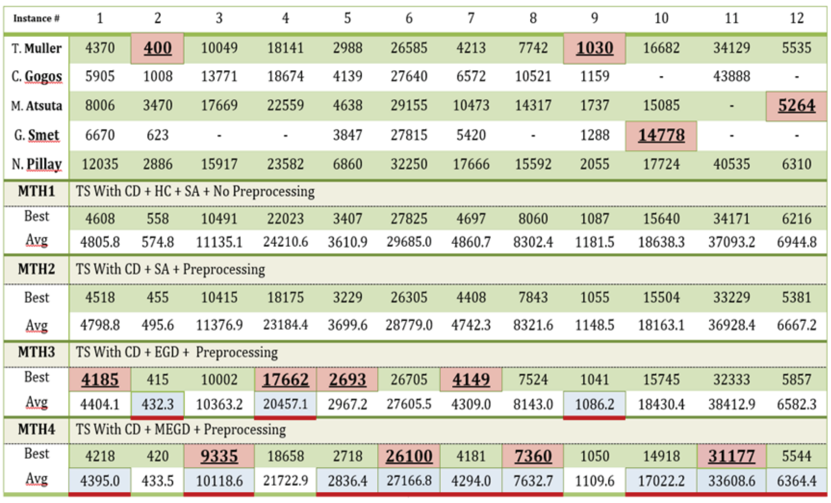


Fig. 8. Performance on dataset 8 of the enhancement phase.

results we will notice that it comes in the second place in 10 out of 12. This is the case for all datasets except datasets 10 and 12. TS with CD + HC+ SA with no preprocessing is the worst algorithm variant in our testing and it comes in the second place in most datasets in comparison to ITC 2007 results.

The other three algorithms variants performed better. Only when we used GD algorithm extensions (EGD and MEGD), we started to see results that overtake ITC 2007 results. Our approach gets 8 out of 12 datasets as best results. These results are split between EGD and MEGD evenly with 4 best results each.

In order to obtain a fair comparison, it is worth noticing that the performance loss is on average about 7% between SA with preprocessing and MEGD with preprocessing, whereas it is about 10% if the preprocessing phase is not implemented with SA. Moreover, one can also notice that the gap between the two methods becomes smaller with higher conflicts density problems, and that the behavior of the methods with pre-processing phase implemented is more stable with respect to SA with no preprocessing phase. All in all, EGD and MEGD performed much better than SA with preprocessing phase not to mention SA with no preprocessing. Finally, all of our methods performed well in comparison to ITC 2007 results in best solution values and in best average values.



Instance #	1	2	3	4	5	6	7	8	9	10	11	12
T. Muller	4370	**<u>400</u>**	10049	18141	2988	26585	4213	7742	**<u>1030</u>**	16682	34129	5535
C. **Gogos**	5905	1008	13771	18674	4139	27640	6572	10521	1159	-	43888	-
M. **Atsuta**	8006	3470	17669	22559	4638	29155	10473	14317	1737	15085	-	**<u>5264</u>**
G. **Smet**	6670	623	-	-	3847	27815	5420	-	1288	**<u>14778</u>**	-	-
N. **Pillay**	12035	2886	15917	23582	6860	32250	17666	15592	2055	17724	40535	6310
MTH1	TS With CD + HC + SA + No Preprocessing											
Best	4608	558	10491	22023	3407	27825	4697	8060	1087	15640	34171	6216
Avg	4805.8	574.8	11135.1	24210.6	3610.9	29685.0	4860.7	8302.4	1181.5	18638.3	37093.2	6944.8
MTH2	TS With CD + SA + Preprocessing											
Best	4518	455	10415	18175	3229	26305	4408	7843	1055	15504	33229	5381
Avg	4798.8	495.6	11376.9	23184.4	3699.6	28779.0	4742.3	8321.6	1148.5	18163.1	36928.4	6667.2
MTH3	TS With CD + EGD + Preprocessing											
Best	**<u>4185</u>**	415	10002	**<u>17662</u>**	**<u>2693</u>**	26705	**<u>4149</u>**	7524	1041	15745	32333	5857
Avg	4404.1	432.3	10363.2	20457.1	2967.2	27605.5	4309.0	8143.0	1086.2	18430.4	38412.9	6582.3
MTH4	TS With CD + MEGD + Preprocessing											
Best	4218	420	**<u>9335</u>**	18658	2718	**<u>26100</u>**	4181	**<u>7360</u>**	1050	14918	**<u>31177</u>**	5544
Avg	4395.0	433.5	10118.6	21722.9	2836.4	27166.8	4294.0	7632.7	1109.6	17022.2	33608.6	6364.4

Table 3. Comparative results.

5 Conclusion and Future Work

We presented our proposed approach to solve exam timetabling problem using four different metaheuristics search methods; tabu search, hill climbing, simulated annealing, and different flavours of great deluge. We also introduced a pre-processing phase to enhance the overall search process. A Tabu metaheuristic search method with conflict dictionary is proposed as a construction phase to achieve a partial or complete initial feasible solution. The tabu list does not contain operators or moves that are problem specific. It only needs to store the conflicted moves along with the accumulated number of conflicts it caused.

A modified extended great deluge heuristic search method is used during search to eliminate some of the time wasted in local optimum based on certain conditions. The selected heuristics perform in sequence to produce a good solution for the current state of the problem. The whole hybrid heuristics approach is configurable and able to manage and control its heuristics without having a domain pre-knowledge of the exam timetabling problem.

In the near future we will investigate advanced variables ordering heuristics [9] as weill as evolutionary techniques using a parallel architecture [25, 26]. We will also intend to explore path consistency algorithms [27] to discover temporal constraints in the preprocessing phase.

References

1. Qu, R., Burke, E., McCollum, B., Merlot, L., Lee, S.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling **12**(1) (2009) 55–90
2. Schaerf, A.: A survey of automated timetabling. Artificial Intelligence Review **13**(2) (2009) 86–127
3. McCollum, B., McMullan, P., Parkes, A., Burke, E., Abdullah, S.: An extended great deluge approach to the examination timetabling problem. MISTA 2009, Multidisciplinary International Scheduling Conference: Theory and Applications (2009) 20–69
4. Gogos, C., Alefragis, P., Housos, E.: An improved multi-staged algorithmic process for the solution of the examination timetabling problem. Annals of Operations Research **194**(1) (2012) 203–221
5. Abdullah, S., Turabieh, H.: On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems. Information Sciences **191**(0) (2012) 146 – 168 Data Mining for Software Trustworthiness.
6. Sabar, N., Ayob, M., Qu, R., Kendall, G.: A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence **37**(1) (2012) 1–11
7. Sabar, N., Ayob, M., Kendall, G., Qu, R.: Grammatical evolution hyper-heuristic for combinatorial optimization problems. Evolutionary Computation, IEEE Transactions on **17**(6) (Dec 2013) 840–861
8. McCollum, B., McMullan, P., Parkes, A.J., Burke, E., Abdullah, S.: An extended great deluge approach to the examination timetabling problem. in MISTA 2009, Multidisciplinary International Scheduling Conference: Theory and Applications, Dublin (2010) 20–69
9. Mouhoub, M., Jashmi, B.J.: Heuristic techniques for variable and value ordering in csps. [29] 457–464
10. Kendall, G., Hussin, N.M.: A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. Practice and Theory of Automated Timetabling **3616** (2005) 270–293
11. L. T. G. Merlot, N. Boland, B.D.H., Stuckey, P.J.: A hybrid algorithm for the examination timetabling problem. Practice and Theory of Automated Timetabling **2740** (2003) 207–231
12. Dowsland, K.A.: Simulate annealing. Modern heuristics techniques for combinatorial problems, ch. 2 (1995) 20–69
13. McCollum, B., McMullan, P.: The second international timetabling competition: Examination timetabling track. University of Nottingham, Queens University, Notingham, Belfast, Technical Report: QUB/IEEE/Tech/ITC2007/Exam/v4.0/17 2007 (2009)
14. E.K. Burke, J.N., Weare, R.: A memetic algorithm for university exam timetabling. Practice and Theory of Automated Timetabling **1153** (1996) 241–250
15. Chan, C.K., Gooi, H.B., Lim, M.H.: Co-evolutionary algorithm approach to a university timetable system. in The 2002 congress on evolutionary computation, Honolulu **2** (2002) 19461951
16. E. K Burke, B. MacCathy, S.P., Qu: Knowledge discovery in a hyperheuristic for course timetabling using case-based reasoning. Practice and theory of automated timetabling (PATAT'02) (2002)
17. M. W. Carter, G.L., Lee, S.Y.: Examination timetabling: algorithmic strategies and applications. Journal of the Operational Research Society **47** (1996) 373–383

18. Dueck, G.: New optimisation heuristics for the great deluge algorithm and the record-torecord travel. Journal of Computational Physics **104** (1993) 86–92
19. Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaption of heuristic orderings. Annals of Operations Research **129**(1-4) (2004) 107–134
20. Müller, T.: Itc2007 solver description: A hybrid approach. in Proceedings of the 7th international conference on the practice and theory of automated timetabling (2008)
21. E. Gogos, C.A., Housos, P.: Multi-staged algorithmic process for the solution of the examination timetabling problem. Practice and theory of automated timetabling (PATAT), 2008 (2008)
22. T. Atsuta, M.N., Ibaraki: An approach using a general csp solver. Technical report (2008)
23. Smet, G.D.: ITC2007 examination track: Practice and theory of automated timetabling. Technical report (2008)
24. Pillay, N.: A developmental approach to the examination timetabling problem. Technical report (2008)
25. Abbasian, R., Mouhoub, M.: An efficient hierarchical parallel genetic algorithm for graph coloring problem. [29] 521–528
26. Abbasian, R., Mouhoub, M.: A hierarchical parallel genetic approach for the graph coloring problem. Applied Intelligence **39**(3) (2013) 510–528
27. Mouhoub, M.: Analysis of Approximation Algorithms for Maximal Temporal Constraint Satisfaction Problems. In: The 2001 International Conference on Artificial Intelligence (IC-AI'2001), Las Vegas (2001) 165–171
28. Mouhoub, M.: Dynamic Path Consistency for Interval-based Temporal Reasoning. In: 21st International Conference on Artificial Intelligence and Applications(AIA '2003), ACTA Press (2003) 393–398
29. Krasnogor, N., Lanzi, P.L., eds.: 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011. In Krasnogor, N., Lanzi, P.L., eds.: GECCO, ACM (2011)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A Criteria Transformation Approach to Timetabling based on Non-Linear Parameter Optimization

Christian John · Dietmar Tutsch · Reinhard Möller · Thomas Lepich · Bernard Beitz

Abstract This paper presents a concept for timetabling based on a parameter optimization system for approximative numerical calculation of some parameter combination under soft and hard constraints. The concept uses a non-linear parameter optimization method with an iterative variation of parameters. The paper focuses on the transformation process to migrate problem-domain specific criteria into optimization-compatible objects suitable for a standardized parameter optimization procedure. A framework for use in other problem domains is presented. The method is applicable to a wide range of timetabling problems due to its dynamically parametrized restrictions. Timetabling is integrated in educational context in various university and scholastic scopes.

Keywords Timetabling · Non-Linear Parameter · Optimization

1 Introduction

Timetabling is used in a vast variety of applications, such as in examination, curricula and courses, scheduling and assigning rooms or in general time management applications. This paper describes a new concept for deriving algorithmic criteria from generalized timetabling ”stories”, the set of fuzzily given conditions and constraints of a timetabling task, in order to calculate appropriate time slots under complex conditions. The general idea is to use a

Christian John
Bergische Universität Wuppertal
Faculty E Rainer-Gruenter-Strasse 21
D-42119 Wuppertal
Building FC, Room 2.07
Tel.: +49-202-4391186
Fax: +49-202-4391944
E-mail: chjohn@uni-wuppertal.de

probabilistic, iterative algorithm avoiding the well-known issues and problems of commonly used timetabling algorithms. Thus, the transformation of general problem-describing "criteria" to "parameters", "constraints", and "conditions" in terms of optimization is in our special focus. Today's timetabling solutions are mostly based on "syntactic", "statistic", or "structural" approaches. There are several disadvantages like preprocessing tasks, stochastic methods with deterministic tasks, fuzziness, clipping, exhaustive search, etc. [1–5]. Our concept is to avoid most of these disadvantages and to develop a more general and extensible solution for timetabling. The conceptual idea is based on a non-linear parameter optimization method, as considered by many authors before, consisting of an objective or target function, restrictions and constraints, and a set of parameters describing the problem domain [6–14].

In addition to existing approaches of this kind, our focus is on conveniently pre-processing the conditions and constraints.

2 Foundation

While typical combinatorial problems can easily be described in form of spoken or written text, their solution depends on the qualified extraction of the criteria governing the problem. Thinking of well-known optimization algorithms, we have to provide parameters, boundary conditions, constraints and an appropriate evaluator to be used as target or objective function. To glue these two sides together, we have to translate some informal criteria description into a syntactically correct formal expression with scalar parameters and explicit boundary conditions such as restrictions and constraints. These expressions can then be used to iteratively find a best-fitting solution - to achieve a goal, meaning to minimize or maximize the evaluation function based on some set of parameters while taking the boundary conditions into account [15,16].

Here the problem arises to transform problem-specific criteria descendent from some real-life problem domain into terms of optimization. Regarding the problem-specific criteria as issues or dimensions in an n-dimensional euclidean space, we have to find an isomorphic transformation of the form:

$$\Pi \approx \Omega$$

with Π being the domain-specific problem space and Ω being the optimization search space.

Fig. 1 shows this bijective transformation with the problem space criteria C_j and the search space parameters P_i and restrictions R_{j-i}.

This transformation mirrors the need for numerical solution of a non-linear system of equations. Several specialized approaches can be found to solve specific optimization problems (Fig. 2). The most general and non-limiting approach is to think of optimization in terms of *non-linear parameter optimization*.

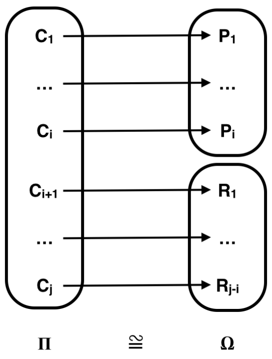


Fig. 1 Bijective criteria transformation

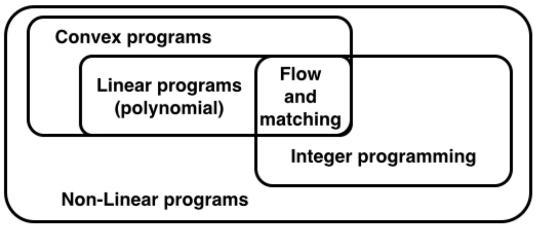


Fig. 2 Optimization problem classes (corresponding to [16])

Non-linear parameter optimization works by an iterative variation of parameters [16–19]. The variation may follow a stochastic and/or deterministic approach and normally comprises some sort of step-width control. While a parameter combination describes the problem domain of the specific given problem, the target function is expressed as *f(parameters)* and gives an estimation value for the grade of optimum fitting and thus the approach of the solution in question. Boundary conditions act as constraints that limit the valid parameter combinations. The optimum may be described as a minimum or maximum of the target function [20–22].

In general, parameter optimization in terms of a target function can be expressed as

$$X = \{x \in \Omega | \forall x' \in \Omega : f(x) \geq f(x') | f : \Omega \to \mathbb{R}, \geq \in \{<, >\}\}$$

with

Ω	the search space
$\geq \in \{<, >\}$	the comparative relation
$X \subseteq \Omega$	the set of global optima

where x is the n-dimensional set of scalar parameters with values fitting into the n-dimensional search space Ω and X is the best-fitting parameter set as the result of the optimization. We are free in the decision to choose the minimum or maximum approach for optimization, here we have chosen the minimum approach.

The iterative variation of parameters can be done in a stochastic or a deterministic manner or a mix of both. Deterministic parameter variation tries to find the best way towards the optimum in search by taking target function differentiations into account, while for stochastic parameter variation the need for a continuously differentiable target function does not arise. Typical representations for deterministic optimization are any forms of gradient procedures, whereas several stochastic procedures have been developed in the past, the evolutionary procedure being one of the most prominent ones.

The decision for deterministic or stochastic procedures in general or for some gradient procedure or evolutionary procedure in detail will have some impact on the overall performance of the optimization, but the main concepts are the same in both cases, so the decision for one of these algorhithms does not infringe the general timetabling procedure described below.

3 Timetabling as a Multi-Criteria Story

Encountering a real-life timetabling problem we are normally confronted with a diffuse informal description of conditions and goals to be met with the solution. We call this the "timetabling story". The first solution step is the need for semantically understanding the story details, rather than an algorithmic question. The meaning of understanding the timetabling story is to identify the events, limitations, desires and the time line. Then we transform these characteristic parts into formal definitions of prarameters, restrictions, constraints, the target function and the definition area. Some parts (see Fig. 3) have to be configured before the optimization is started.

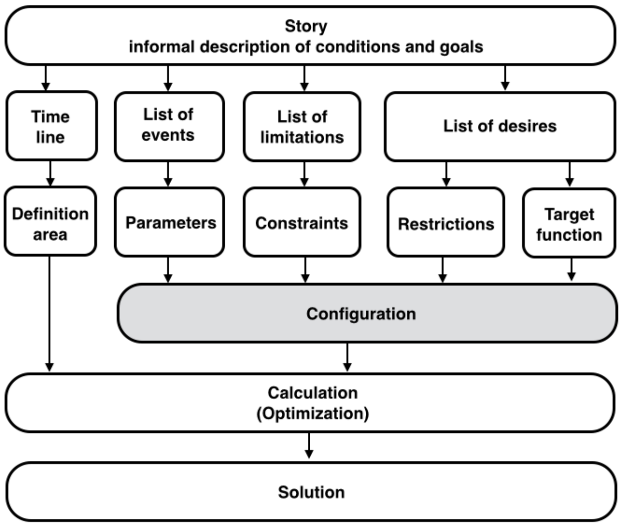


Fig. 3 Timetabling story transformation

Giving a definition of the term *timetabling*, we think of timetabling as some kind of time management for events, determining ideal time slot combinations or structured schedule based on a time line and restricted to a time interval. A timetable consists of distinct time slots, each having a start and end time, often denoted as "arrival" and "departure", and they need to be arranged or combined on the time line. The timeline is given as a one-dimensional definition region of time, restricted by a distinct start and end timestamp. Unfortunately the given time space is perforated by unavailable periods, e.g. weekends, holidays and night times. The remaining free time spaces then can be filled with the time slots.

While these aspects are concerning the timeline as a whole, more criteria have to be derived from the story, concerning inner relations and dependencies between time slots. Time slots may have a prescribed order, some or all time slots must not overlap, distance preferences may occur as well as daytime preferences.

Additionally, in almost all cases we have to ensure that all time slots find accommodation in the timeline because time slots must not be skipped.

In this paper, our goal is to create a generalized approach to all of these timetabling issues - or at least to a vast majority of them. Looking at the timetabling tasks in a more abstract way, we can name a number of general principles that can be found in all of the above-mentioned conditions:

– definition of time line, start and end time
– definition of blocked time spans
– recognition of free time spans
– observing relations, distances and dependencies
– observing allowed or forbidden overlapping
– observing orders and preferences.

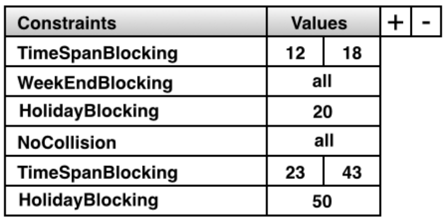
These principles lead to the generalized criteria:

– time and duration
– relations and grouping
– delay and distance
– order and preferences
– overlapping

Other criteria may arise as well. Thus timetabling can be described as a multi-criteria problem. While some standard approach might think of criteria in terms of unilateral, bilateral and multilateral event relations, our solution handles these criteria classes (and any other criteria) in a unified manner. Even real-world knowledge may be integrated into this approach by defining customized optimization conditions.

Our generalized timetabling solution is based on a non-linear parameter optimization procedure (as described above), while the different criteria are implemented as sets of parameters and restrictions.

Fig. 4 shows the schematic user-interface for the "configuration" part in Fig. 3 where arbitary criteria may be added and parametrized by selecting predefined criteria classes from a given criteria catalogue and assigning appropiate values.



Constraints	Values		+	-
TimeSpanBlocking	12	18		
WeekEndBlocking	all			
HolidayBlocking	20			
NoCollision	all			
TimeSpanBlocking	23	43		
HolidayBlocking	50			

Fig. 4 Schematic User-Interface for Configuration and value assignment

Fig. 5 shows an example of a specific dynamically configured non-linear parameter optimization application with a combination of several different criteria principles, a target function, a set of restrictions and parameters. Thus the concept yields the following advantages: a dynamic selection of criteria, parametrizable constraint values, reusable restrictions, and enabling or disabling of conditions. As a result, we obtain a highly reconfigurable general solution system.

While this is the conceptual approach, the question arises, how to implement the named timetabling principles in a real-life dynamically configurable software application, allowing for adding any principle known and upcoming.

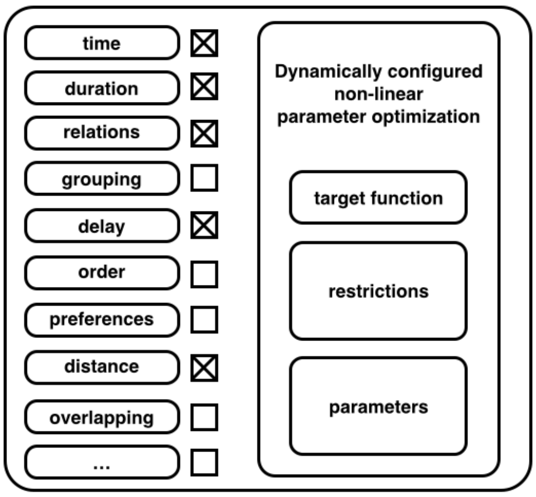


Fig. 5 Dynamically configured non-linear parameter optimization

4 Dynamic Multi-Criteria Transformation in OOP

After describing the conceptual structure of our problem domain the implementation starts with the technical realization in form of a three-layer OOP system (see Fig. 6).

In the top-down approach for analyzing the timetabling problem domain described above we have generalized typical timetabling questions, including event relations and real-world knowledge, into abstract criteria. The Object-Oriented Programming (OOP) design principle is the best-fitting paradigm to implement generalizable problem structures with concrete algorithm implementations. Following the standard OOP nomenclature, object classes are ruled by interfaces to encapsulate implementation details and realize inheritance and polymorphy.

We use OOP with classes and interfaces to implement the basic non-linear parameter optimization [16, 21] as our general problem solver and we provide interfaces to embed concrete parameters, restrictions, constraints, and target functions to meet the above-mentioned timetabling requirements.

Following this flexible and expandable design, we have structured our solution into a three-layer scheme, of which the first two layers contain the predefined parts of the software solution, while the third layer is the place for the implementation details of the particular problem (which, in our case, is the timetabling problem). This expandable layer has to be implemented individually with program code to match the problem's requirements.

4.1 First layer: Optimization

This is the basic layer forming the general foundation for our solution: the optimization layer. It generalizes the non-linear parameter optimization, defining the interfaces *IOptimizer* (for the optimizing algorithm as such) and a suitable *IVariator* (implementing the parameter variations and step-width control). The *IOptimizer* interface in particular is the hook for the real optimization algorithm's implementation. In general, we can differentiate between stochastic and deterministic optimization algorithms. In our implementation, a simple gradient program represents the deterministic algorithm class, where we decided to use a simple evolution program as representation of a stochastic algorithm. The variators have been developed accordingly. The optimizers in this basic layer may be used "as-is" by the concrete optimization task, but more sophisticated optimizers could be hooked in if estimated necessary, promising possible performance gains.

4.2 Second layer: Interface

The second layer is situated between the optimization and the real-life task. It defines the interfaces for linking the other layers together. The interface layer provides several pre-conditions for the real-life task implementation to be fulfilled.

– *IParameter*: As described above, parameter optimization is equivalent to finding the best-fitting parameter combination according to a given evaluation or target function. The *IParameter* interface defines the abstract concepts for optimization parameters to be integrated into an optimization procedure.
– *IConstraint*: A constraint works as a boundary condition, hard-restricting parameter values to represent a valid solution by ensuring parameters to be contained in a valid range of values. Parameters violating a constraint must not be used in further calculations, they will be rejected and a new parameter variation will be triggered.
– *IRestriction*: A restriction is similar to a constraint, but it is a weak constraint: it is allowed to be violated, but violation will be punished with bad target values. Ideally, a restriction is implemented in a way such that violating the restriction will force the optimization converge against the optimum or at least against better values, expressed in parameter combinations. It will not enforce a new parameter set. Complying the restriction is a positive quality statement for the result - a valid and potentially better parameter combination.
– *ITarget*: This is the target function, evaluating and rating the current parameter combination. The target is the central place for defining the optimization goal. It is important to state that for deterministic algorithms the target function must be continuously differentiable, at least locally, i. e. it must be possible to calculate a numeric differentiation for the current parameter combination. Non-deterministic and stochastic algorithms do not rely on this precondition of differentiability.
– *ITask*: This is the overall container defining the concrete problem and binding the other parts together.

4.3 Third layer: Real-life task

This layer is the concrete implementation of all problem-specific parameters, constraints, restrictions as well as the target function (see Fig. 7). To build up a new problem we first define a Task and identify and implement the following details:

– The target function as implementation of *ITarget*
– Identify the problem's parameters and implement as set of *IParameter*, while initializing with estimated start parameter values
– Identify the constraints and implement as IConstraint subclasses
– Identify the restrictions and their penalty behavior, implemented as set of *IRestriction*

Now we can start the calculation. The optimization will begin with the start parameter set and iteratively vary the parameters and rate the current result by evaluating the target function. The calculation will stop as soon as some termination condition is met. Using the evolutionary procedure [22], termination may simply be defined in terms of iteration steps, thus increasing the number of steps can give the chance for better results while deteriorating the runtime needed. In a more general approach, optimization procedures may have more sophisticated termination conditions in form of convergence criteria, promising more effective termination conditions. In this case, convergence has to be defined in the context of the problem domain. For deterministic optimization procedures, the target function's differentiability can help finding convergence criteria while running the risk of being stuck in some local optimum. For non-deterministic and stochastic procedures, the local optimum problem is reduced, but convergence criteria must be defined in some supplementary way.

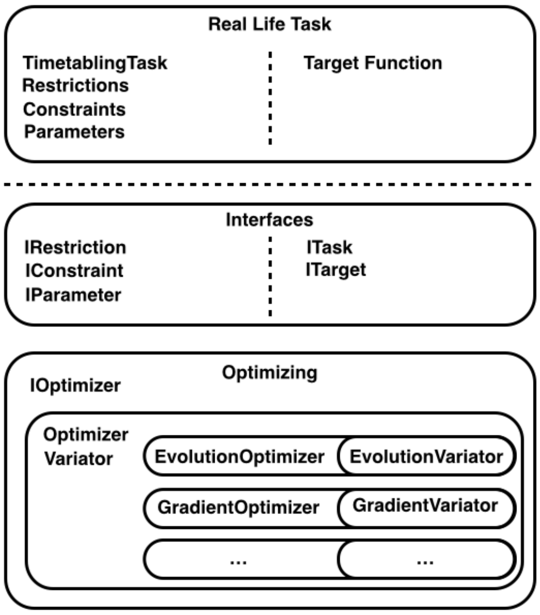


Fig. 6 Full three layer scheme

The whole three-layer scheme is outlined in Fig. 6, showing the OOP transformation of the dynamic multi-criteria timetabling algorithm.

4.4 Multi-Criteria Framework

We have seen that for solving a concrete task using this generalized OOP and optimization approach at least some programming is involved, implementing the scheme's interfaces appropriately. To facilitate this we have created a framework outlined below (see Fig. 7).

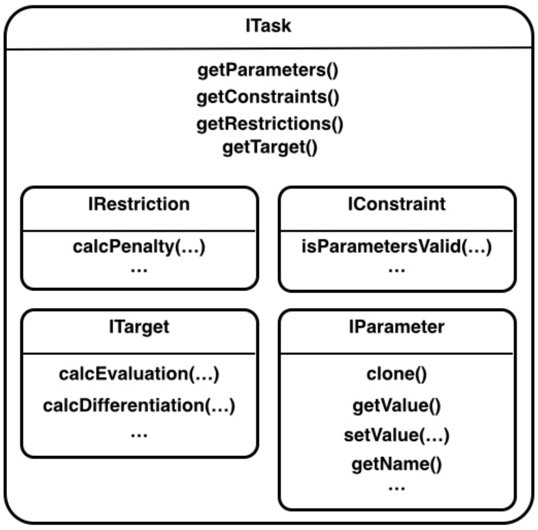


Fig. 7 Class Model

5 Scaling Effects in Transformation

Transforming the n-dimensional problem domain criteria into an optimization-based numerical solution, the problem of different parameter dimensions in terms of measure arises. In general, we can distinguish parameters in the dimensions

– time and date
– order and relation
– length and angle
– force, mass and weight
– color and brightness
– elasticity
– temperature
– and potentially many other dimensions.

Especially for timetabling, the time and order dimensions will become important. Both types of parameters have different definition ranges and cannot be handled equally without normalization. It is clear that those parameters must be submitted to normalization using their specific definition range. As a result, all normalized parameters may be treated equivalently in variation during optimization. We apply normalization in a strictly local manner, such that the *IParameters* internally encapsulate their true values while parameter variations apply normalized variation steps to them.

While *IConstraints*, if violated, decline our parameter set as a whole, *IRestrictions* provide a calculated penalty value representing the grade of non-compliance of the corresponding condition. A simple restriction may calculate its penalty value using a linear function. The linear function's slope can be used as a weight factor for restrictions with different significance. More sophisticated penalty functions may be formulated as exponential functions.

As a result, we have to observe that

- parameters have to be normalized according their definition range
- restrictions have to be provided with an appropriate rating factor.

6 Example: Examination Planning at BUW

Paradigmatically, we show the conceptual approach to probabilistic timetabling for our institute at BUW Bergische Universität Wuppertal.

In the examination period we set our exams time slots, taking into consideration all constraints and relations between exams.

First we have to define a start-time constraint and an end-time constraint reflecting the examination period as a whole. Then we add more constraints:

- No exam should be written on a weekend - a *WeekendConstraint* arises
- Furthermore no exam should be written in an inconvenient time, for example in the night or too early in the morning - a *NightConstraint*
- Holidays are inconvenient as well as exam date - giving a *HolidayConstraint*

The constraints' goal is to find valid exam dates - our parameter set - in the resulting free time slots. Fig. 8 shows the constraints for the given timeline, limited to the constraints declared. Zooming into the timeline focusses a section of the timeline, in this example it is a typical week. Further zooming leads into the monday section. Obviously, applying all constraints gives a valid time span between 8:00 AM and 4:00 PM. In this idle time period on monday the algorithm might place five exams belonging to three different exam groups and thus partially overlapping. As we only consider the constraints at this point, the exam time slot distribution shown is valid, but not yet necessarily optimal. Further optimization steps will variate the time slot positions in the gaps of idle times, until eventually the optimum distribution is found.

7 OOP Transformation Framework

The OOP transformation implies the identification of the parameters describing this example task, the constraints and restrictions to be applied and the target function definition.

Running the optimization procedure, it will hopefully give a solution in form of the best-fitting parameter combination for our target function after n iterations. Here it would be a parameter set of t_i for the start of each time span. The number n of iterations needed to find the solution depends on several conditions: the arbitrary distribution of the exams start points, the kind of step-width control, the local-vs.-global optimum bias, and the selected fitting tolerance of the target function (the latter is equivalent to the algorithm's termination condition).

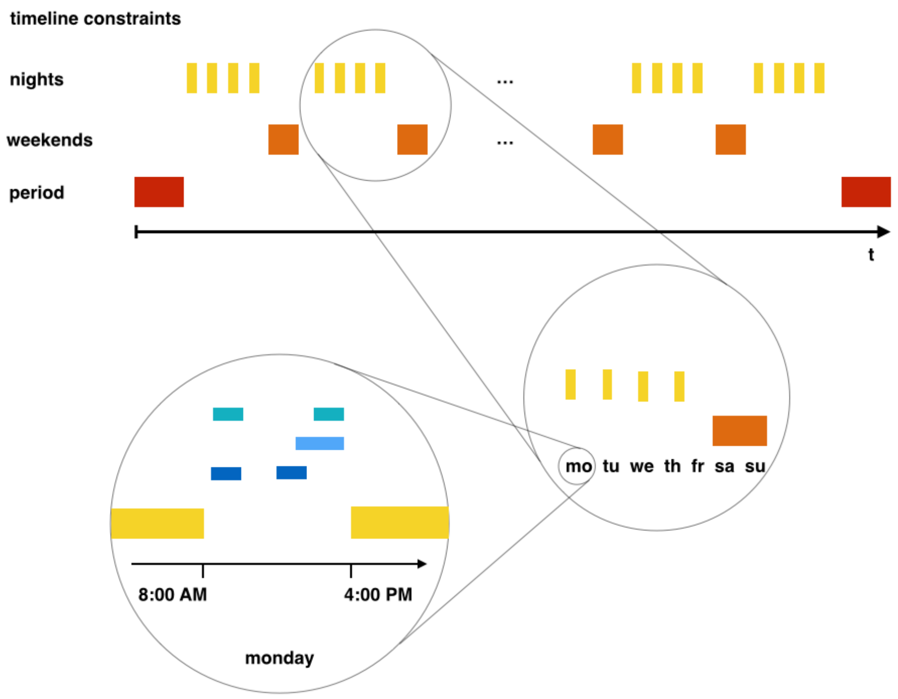


Fig. 8 Overview of the timeline constraints and free time spans

The number n of iterations can be estimated as being far lower than the steps we would need for an exhaustive search or any Monte-Carlo approach, but we can expect performance loss in comparison to conventional, direct timetabling algorithms. The performance loss will be compensated by the ability to dynamically parametrize and alter the set of constraints and restrictions.

8 Constraints and Restrictions

According to the previous examination timetabling example we limit the timeline by the period's start and end time defining these limits as two constraints: a *startConstraint* for the start or departure and an *endConstraint* for the end or arrival. The idea of constraints is to describe general conditions for a problem forcibly limiting the allowed parameter values. In this case, we have two boundary constraints because the time region has two limiting sides. E.g., the left side constraint checks the current parameter t against the region limit on the left side (see implementation: Listing 1). The result of these tests mark the actual given parameters to be valid or invalid. In case of invalid parameters the optimization procedure will trigger a new parameter variation, after which the parameters will again be submitted to constraint checks. In case of valid parameters the target function will be evaluated using the current parameter set.

```
public class StartConstraint implements IConstraint
{
        public boolean isParametersValid(ParamSet params)
        {
                for (IParameter : params)
                        if (p.value < startTime)
                                return false;
                return true;
        }
}
```

Listing 1 Start constraint

While we understand constraints to be mandatory limitations for the problem's parameters, we describe restrictions to be weak constraints. Violating a restriction results in a bad target function value due to calculated penalty values from the restriction. In the context of our example the condition "two exams have to be a distance of seven days apart" would be a restriction (see implementation: Listing 2).

```
public class DistanceRestriction extends TimeRestriction
{
        private double dist;
        private int i1;
        private int i2;
        private double factor = 1.0;

        public DistanceRestriction(double factor, double dist,
                                   int i1, int i2)
        {
                super(factor);
                this.dist = dist;
                this.i1 = i1;
                this.i2 = i2;
        }

        public double calcPenalty(ParamSet params)
        {
                double t1 = params.get(i1).value;
                double t2 = params.get(i2).value;
                double delta = Math.abs(t1-t2);
                if (delta >= dist)
                        return 0.0;
                return factor * (dist - delta);
        }
}
```

Listing 2 Restriction for distance check

9 Target Function

Regarding the requirements described so far, our target function must fulfill two goals:

1. penalty values must lead back into the allowed time slots
2. when all restrictions are fulfilled, the time-span distribution must be submitted to a rating function, so that the optimization can converge towards the desired ideal distribution.

While goal (1) is handled by the restriction's penalty function values, fulfillment of goal (2) depends on programming the rating. The actual rating is formulated in terms of "distribute all time spans equally", "prefer gathering the time spans at the beginning of the period" or any other goal for the specific problem given.

While the start conditions

– set of start parameters
– start step width of the variation (i.e. search radius)

are crucial for a converging algorithm, the target function must reflect the optimization goal by rating the current parameter values according to the selected goal.

10 Dynamic configuration

The perception we have experienced by working with the timetabling is that a principal solution appears to be generalizable but specific timetabling tasks need specific ways of solution in detail. So we decided to provide a dynamically configurable solution process to represent the solution details.

Fig. 8 shows a configurable timetabling task and appropriate target function with optional settings for management of constraints and restrictions; we can turn on and off constraints and restrictions as needed, and in case of a missing condition, we can simply define a new constraint and/or restriction.

According to Fig. 9, adding a holiday would result in adding a new time-locking constraint for this day. As an example for a completely new condition we could introduce the idea of two time spans requiring to be in a defined order. In this case we implement a new *IRestriction* class and insert it into the timetabling task.

Applying the described procedure to real-life timetabling problems, short-term occurrences (e.g. teacher's illness) are omitted because they are not predictable.

11 Implementation

The software has been implemented in Java using the Eclipse RCP framework. The RCP framework and its plug-in mechanism allow to conveniently

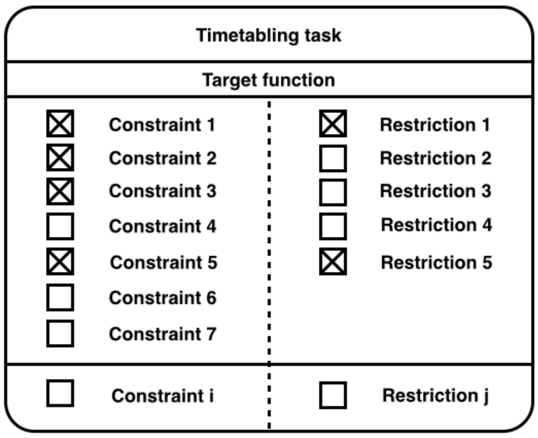


Fig. 9 Dynamic configuration of algorithm

expand the software for adapting to the current specific problem domain by implementing the basic framework interfaces. The core optimization algorithm is implemented as a basic plug-in (OSGi bundle) using the well-known evolutionary procedure. For performance enhancement purposes other optimization procedures may be implemented as well, no matter if following the stochastic or the deterministic approach. Any constraints, restrictions and target functions can be realized as additional plug-ins, thus making the whole package very flexible and expandable for other optimization domains.

12 Real-Life Applications

The main aspect for this work descended from the need for examination planning. Different kinds of exam constraints and exam restrictions lead to a non-linear parameter optimization approach. Before we built up this system, the planning was performed manually and with rather high error rate. The idea arose to create an automatic exam planning system and avoiding error rates.

We used the optimized exam dates for an extended application for room management. This includes a room list together with the room seat capacities and assigns exams to rooms, taking into consideration the expected numbers of participants.

After calculating the exam dates and appropriate rooms we provided this informations for another project: We developed an application for helping students and visitors navigating through and locating at the Wuppertal University Campus. Our students will use their smartphone and its integrated camera to take a picture from their current environment, and a server-based application will identify their position and locate them on the campus. Together with the timetabling information, students and exam candidates can be guided to their target exam rooms using smartphones and web browsers. This procedure is an extension of our eCampus project [eCampus] [23].

13 Conclusion

A new, flexible and configurable framework for transforming timetabling criteria stories was presented, using standardized condition classes for an iterative approach via non-linear parameter optimization. The concepts for transforming the different timetabling criteria into objects to be applied in an optimization procedure were shown and a simple framework for creating optimization-oriented subclasses and end-user applications has been presented. Scaling effects for parameters and restrictions have been discussed. The method uses constraints and restrictions to integrate arbitrary boundary conditions to represent timetabling specifics for a wide range of applications. An example demonstrated the basic ideas behind the transformation method and discussed some oncoming problems and their solutions.

References

1. Tomáš Müller, Real-life Examination Timetabling, in MISTA 2013 - Proceedings of the 6th Multidisciplinary International Scheduling Conference, 2013.
2. Tomáš Müller, H. Rudová, Real-life Curriculum-based Timetabling In PATAT 2012 - Proceedings of the 9th international conference on the Practice And Theory of Automated Timetabling, 2012.
3. Rui Li, Michael T.M. Emmerich et al.: Mixed Integer Evolution Strategies for Parameter Optimization. MIT 2013.
4. Alberto Colorni , Marco Dorigo , Vittorio Maniezzo, A Genetic Algorithm To Solve The Timetable Problem, Computational Optimization and Applications Journal, 2013
5. Tanguy Lapègue, Damien Prot, Odile Bellenguez-Morineau, A Tour Scheduling Problem with Fixed Jobs: use of Constraint Programming, Practice and Theory of Automated Timetabling, 2012.
6. Moritz Mühlenthaler, Rolf Wanka, Fairness in Academic Course Timetabling, Practice and Theory of Automated Timetabling, 2012.
7. Gilles Pesant, A Constraint Programming Approach to the Traveling Tournament Problem with Predefined Venues, Timetabling, Practice and Theory of Automated Timetabling, 2012.
8. Benny Raphael, Ian F. C. Smith, Engineering Informatics: Fundamentals of Computer-Aided Engineering, Second Edition, John Wiley and Sons, 2013
9. Wil Michiels, Emile Aarts,Jan Korst, Theoretical Aspects of Local Search, Springer 2007
10. Francesca Rossi; Peter Van Beek; Toby Walsh, Handbook of constraint programming. Elsevier 2006
11. Burke E.K., Landa Silva J.D., Soubeiga E., Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, In Meta-heuristics: Progress as Real Problem Solvers, Selected Papers from the 5th Metaheuristics International Conference (MIC 2003), pp 129-158, 2005
12. Kalyanmoy Deb, Multi-Objective Optimization Using Evolutionary Algorithms, Wiley Paperback 2009
13. Kaisa Miettinen, Nonlinear Multiobjective Optimization, Springer 2012
14. Matthias Ehrgott, Multicriteria Optimization, Springer 2010
15. Christian John, Reinhard Möller, A Probabilistic Approach to Pattern-Matching Based on a Dynamic Rule-Driven System, 2013 IEEE GHTCE, Shenzhen 18.11.2013
16. Christos H. Papadimitriou, Kenneth Steiglitz, Combinatorial Optimization - Algorithms and Complexity. 1998 Dover, New Jersey
17. Jörn Schmidt, Christina Klüver, Jürgen Klüver: Programmierung naturanaloger Verfahren. Vieweg+Teubner. Wiesbaden 2010.
18. Juraj Hromkovic. Randomisierte Algorithmen. Teubner. Wiesbaden 2004.

19. Rolf Wanka. Approximationsalgorithmen. Teubner. Wiesbaden 2006.
20. Christian John, Thomas Lepich, Bernhard Beitz, Reinhard Möller, Dietmar Tutsch, A Probabilistic Approach to Pattern-Matching Based on Non-Linear Parameter Optimization, 2014 IEEE WCCAIS ICCIS, Sousse 17.02.2014
21. Walter Alt, Nichtlineare Optimierung, 2011, Vieweg+Teubner Verlag, Wiesbaden
22. Karsten Weicker: Evolutionre Algorithmen.Vieweg, Berlin 2007.
23. http://www.gds.uni-wuppertal.de/gds/service/veranstaltungen/projekt-icampus.html (2013-07-02,15:00)

KHE14: An Algorithm for High School Timetabling

Jeffrey H. Kingston

Abstract This paper presents work in progress on KHE14, an algorithm for high school timetabling. Most of KHE14's components have been published previously, so are described only briefly. A few components, notably many of the augment functions called by the ejection chain algorithm, are new, so are described in detail. The paper includes experiments on the standard XHSTT-2013 data set, performed in February 2014.

Keywords High school timetabling · Ejection chains

1 Introduction

High school timetabling is one of the three major timetabling problems found in academic institutions, along with university course timetabling and examination timetabling. Automated methods for its solution have been studied from the early days of computers [19] to the present day [15].

An XML format called XHSTT was introduced recently to represent real instances and solutions of the high school problem [8,16]. XHSTT was used in the Third International Timetabling Competition [18], the first competition to include high school timetabling. An XHSTT data set called XHSTT-2013 is currently being promoted as a benchmarking standard [17]. It contains 24 instances from real high schools in 11 countries around the world.

This paper presents work in progress on KHE14, an algorithm for high school timetabling, with experiments on XHSTT-2013 made in February 2014.

KHE14 is built on the author's KHE high school timetabling platform [12], and distributed with it. It has many parts, developed by the author in a series of papers over the last ten years [6,7,9–11]. All this cannot be repeated here. So although this paper describes KHE14 completely, it does so only at a high

J. Kingston
School of Information Technologies, The University of Sydney, Australia
E-mail: jeff@it.usyd.edu.au

level. Details are explained only when they are new and significant; otherwise they are just mentioned, with a reference to the papers just cited, or to the KHE documentation [12], which has full details. The main innovations are in Sect. 7, where new repairs for several kinds of defects are given.

Sect. 2 gives a brief specification of the problem. Sects. 3–7 present the components of KHE14, with experiments related to the components. Sect. 8 brings the components together into the full KHE14 algorithm and contains experiments that evaluate it generally.

All experiments use the XHSTT-2013 data set [16], as downloaded on 4 February 2014, and were performed on the author's desktop machine, an Intel i5 quad-core running Linux. Individual solutions were produced single threaded; multiple solutions for one instance were produced in parallel. The DK-HG-12 instance is not tested, because it is not realistic; it has 411 demand defects (Sect. 4) before solving starts (connected with resources Student539 and R36, for example). Instances NK-KP-05 and NK-KP-09 are not tested because their run times are inconveniently long at present. KHE14 does solve these instances, but it takes approximately one hour on each.

2 Problem specification

This paper uses the XHSTT specification of the high school timetabling problem. XHSTT instances contain four parts: the *cycle*, which is the chronological sequence of times that may be assigned to events; *resources*, which are entities that attend events (teachers, rooms, students, or classes, where classes are groups of students who mostly attend the same events); *events*, which are meetings, each of fixed duration (number of times), and containing any number of *event resources*, each specifying one resource that attends the event; and *constraints*, which specify conditions that solutions should satisfy, and the penalty costs to impose when they don't.

XHSTT currently offers 16 constraint types (Table 1). Whatever its type, each constraint may be marked *required*, in which case it is called a *required* or *hard* constraint, and its cost (a non-negative integer) contributes to a total called the *infeasibility value* in XHSTT, and the *hard cost* here. Otherwise the constraint is called *non-required* or *soft*, and its cost contributes to a different total called the *objective value* in XHSTT and the *soft cost* here.

A solver assigns starting times to events, except for *preassigned* events (events whose starting time is given by the instance), trying to minimize first hard cost and then soft cost. It may also be required to split events of long duration into smaller events, called *sub-events* in XHSTT and *meets* in KHE and in this paper. And it may be required to assign resources to unpreassigned event resources: often rooms, occasionally teachers, never (in practice) students or classes. A full specification appears online [8]; further details are given as needed throughout this paper.

Table 2 gives some idea of the instances of the XHSTT-2013 data set. They vary greatly in difficulty, in ways that such a table cannot fully capture.

Table 1 The 16 constraints, with informal definitions, grouped by what they apply to.

Event constraints	
Split Events constraint	Split event into limited number of meets
Distribute Split Events constraint	Split event into meets of limited durations
Assign Time constraint	Assign time to event
Prefer Times constraint	Assign time from given set
Spread Events constraint	Spread events evenly through the cycle
Link Events constraint	Assign same time to several events
Order Events constraint	Assign times in chronological order
Event resource constraints	
Assign Resource constraint	Assign resource to event resource
Prefer Resources constraint	Assign resource from given set
Avoid Split Assignments constraint	Assign same resource to several event resources
Resource constraints	
Avoid Clashes constraint	Avoid clashes involving resource
Avoid Unavailable Times constraint	Make resource free at given times
Limit Idle Times constraint	Limit resource's idle times
Cluster Busy Times constraint	Limit resource's busy days
Limit Busy Times constraint	Limit resource's busy times each day
Limit Workload constraint	Limit resource's total workload

Table 2 The number of times, teachers, rooms, classes (groups of students), individual students, and events in the instances of XHSTT-2013. There are 24 instances altogether.

Instance	Times	Teachers	Rooms	Classes	Students	Events
AU-BG-98	40	56	45	30		387
AU-SA-96	60	43	36	20		296
AU-TE-99	30	37	26	13		308
BR-SA-00	25	14		6		63
BR-SM-00	25	23		12		127
BR-SN-00	25	30		14		140
DK-FG-12	50	90	68		279	1120
DK-HG-12	50	100	70		523	1471
DK-VG-09	60	46	52		163	928
ES-SS-08	35	66	4	21		225
FI-MP-06	35	25	25	14		280
FI-PB-98	40	46	34	31		387
FI-WP-06	35	18	13	10		172
GR-H1-97	35	29		66		372
GR-P3-10	35	29		84		178
GR-PA-08	35	19		12		262
IT-I4-96	36	61		38		748
KS-PR-11	62	101		63		809
NL-KP-03	38	75	41	18	453	1156
NL-KP-05	37	78	42	26	498	1235
NL-KP-09	38	93	53	48		1166
UK-SP-06	25	68	67	67		1227
ZA-LW-09	148	19	2	16		185
ZA-WD-09	42	40		30		278

3 Timetabling structures

KHE evaluates constraints continuously as the solution changes during solving, using efficient incremental methods, and makes the resulting costs available to solvers, which use them to guide the solve as usual. If requested, KHE can also add structures to the solution which ensure that violations of some constraints cannot occur, and it can add other structures which encourage *regularity*: patterns of assignment that make timetables more uniform. Regularity has no direct effect on cost, but it may make good solutions easier to find [11].

KHE14's first, *structural* phase, is mainly devoted to adding the structures explained in this section. These are all optional as far as the KHE platform is concerned; KHE14 chooses to use them, but other algorithms need not.

KHE14 does not use any information that could be called metadata. For example, sets of times may be identified in XHSTT as days, but KHE14 does not use that information. Nor does it treat student resources (say) differently merely because they are called students. Instead, it examines which resources are preassigned, which sets of times and resources appear in constraints, and so on, taking its cues from the structure alone.

Many elements of the instance influence KHE14's structures: avoid clashes constraints (constraining events which share a preassigned resource to be disjoint in time), time preassignments, link events constraints, split events and distribute split events constraints, spread events constraints (influencing how many meets events split into), prefer times constraints, prefer resources constraints, and avoid split assignments constraints. These are taken in decreasing cost order; each either influences the structures, or is ignored if inconsistent with previous elements. The KHE documentation [12] explains in detail how they affect the result and how they interact. It would take too long to repeat that here. Instead, the following explains the structures that emerge.

Courses are sets of events during which the same students meet the same teacher to study the same subject. Spread events constraints may be present to encourage a course's meets to spread evenly through the cycle, and avoid split assignments constraints may be present to encourage those meets to be assigned the same teacher (if not preassigned) or room.

XHSTT offers a spectrum of ways to define courses. At one extreme, the exact set of events required is given. For example, if a Mathematics course needs to occur five times per week in events of durations 2, 1, 1, 1, and 1, then five events with these durations would be given, along with split events constraints which ensure that each event produces one meet. At the other extreme, a single event of the total duration required is given, along with split events and distribute split events constraints which say how that total may be split into meets. In the Mathematics example, a single event of duration 6 would be given, along with constraints saying that meets of duration 1 or 2 are required. This handles a situation frequently found in real instances, where the total duration is fixed, but how it is to be split up is more flexible.

The structural phase splits events into meets whose durations depend on the parts of the instance listed above, and groups the meets into sets that KHE calls *nodes*. One node contains the meets of one course, at least to begin with. The structural phase creates nodes heuristically, as follows. Meets derived from the same event go into the same node. When two events contain the same preassigned resources and are connected by a spread events or avoid split assignments constraint, they are taken to belong to the same course, so their meets also go into the same node. Grouping meets into nodes does not constrain their assignments, but it acts as a hint to solvers that the meets should be assigned times together, and opens the door to various methods of promoting regularity, which work with nodes, not meets.

Link events constraints, specifying that certain events should be assigned the same times, give rise to a different structure. KHE allows one meet to be assigned to another instead of to a time, meaning that any time assigned to the other meet is in fact assigned to both. The structural phase makes assignments of meets to other meets which ensure that link events constraints cannot be violated. Meets assigned to other meets are not included in nodes, which (by convention) tells solvers that their assignments should not be changed.

Assigning one meet to another supports *hierarchical timetabling*, in which a timetable for a few meets is built and later incorporated into a larger one. This promotes regularity, so the structural phase spends time searching for useful hierarchical structures, as described in [6,7].

Each meet contains a set of times called its *domain*. Only times from its domain may be assigned to a meet. KHE14 chooses domains based on prefer times constraints. The duration of a meet also affects its domain: a meet of duration 2 cannot be assigned the last time in the cycle as its starting time, and so on. KHE represents domains both as bit sets, for efficient assignability testing, and as lists of times, for efficient iteration over all legal assignments.

A meet contains one *task* for each event resource in the event that it is derived from. Each task is a demand for one resource at each of the times the meet is running, either preassigned (the usual case for student and class tasks) or not (the usual case for room tasks). Unpreassigned tasks specify the type of resource required (teacher, room, etc.), and there are usually prefer resources constraints which encourage the solution to assign a specific kind of resource, such as a Mathematics teacher or a Science laboratory.

Each task contains a set of resources called its *domain*. Only resources from its domain may be assigned to a task. KHE14 chooses domains based on prefer resources constraints.

Avoid split assignments constraints, which specify that certain tasks should be assigned the same resources, are handled structurally by KHE14. One of the tasks is chosen to be the *leader task*, and the others are assigned it instead of a resource, meaning that whatever resource is assigned to the leader task is to be considered as assigned to them too.

The XHSTT specification says that hard constraint violations, while permitted, should be few in good solutions, but that soft constraint violations are normal and to be expected [8]. So additional structures must be used with caution, especially when derived from soft constraints. KHE14 uses an heuristic strategy: it includes them at first, but removes them towards the end, so

Table 3 Encouraging regularity between forms: -RF and +RF denote without it and with it. KHE14 uses +RF. In all tables in this paper, columns headed C: contain solution costs. Hard costs appear to the left of the decimal point; soft costs appear as five-digit integers to the right of the point. The minimum costs in each row are highlighted. Columns headed T: contain run times in seconds. All tables and graphs (including captions) were generated by KHE and incorporated unchanged. They can be regenerated by any user of KHE.

Instance	C:-RF	C:+RF	T:-RF	T:+RF
AU-BG-98	**9.00583**	12.00491	17.1	25.4
AU-SA-96	**4.00015**	16.00014	35.1	72.2
AU-TE-99	**1.00158**	4.00124	0.8	3.4
BR-SA-00	1.00090	**1.00057**	0.7	0.8
BR-SM-00	30.00123	**29.00093**	4.4	6.3
BR-SN-00	5.00249	**4.00243**	1.6	2.8
DK-FG-12	0.02370	**0.02248**	94.0	180.9
DK-HG-12				
DK-VG-09	**12.03206**	12.03349	393.9	368.4
ES-SS-08	0.02367	**0.01362**	4.4	15.6
FI-MP-06	**0.00118**	3.00132	1.5	8.0
FI-PB-98	**0.00051**	6.00039	1.7	6.3
FI-WP-06	0.00086	**0.00078**	1.7	1.3
GR-H1-97	**0.00000**	**0.00000**	0.5	5.9
GR-P3-10	4.00078	**2.00088**	16.6	22.7
GR-PA-08	**0.00029**	0.00040	2.4	6.9
IT-I4-96	0.00602	**0.00494**	4.6	8.1
KS-PR-11	0.00160	**0.00150**	10.7	77.1
NL-KP-03	**0.03825**	0.04774	86.4	193.1
NL-KP-05				
NL-KP-09				
UK-SP-06	0.00196	**0.00102**	14.0	29.5
ZA-LW-09	29.00000	**26.00000**	16.9	18.9
ZA-WD-09	**5.00000**	9.00000	6.3	24.2
Average	**4.00681**	5.00660	34.0	51.3

that later repair operations are not limited by them. The original constraints are not forgotten: even when violations are allowed, they are still penalized.

The author has used a structural phase like the one described here for many years [6, 7]. KHE14's version, described briefly here and fully in the KHE documentation [12], is more robust than its predecessors: it resolves conflicting requirements using priorities as explained above, and it takes full account of all interactions between requirements.

Testing the effectiveness of adding structures that encourage regularity is complicated by the fact that there are several kinds of regularity and several ways to encourage it [11], not all of which can be disabled at present. Table 3 investigates regularity between forms. For example, if the classes of the Year 11 form attend English for 6 times per week in meets of durations 2, 1, 1, 1, and 1, and the classes of the Year 12 form attend Mathematics for 6 times per week in meets of the same durations, then encouraging regularity between forms encourages these two courses (or others with the same meet durations) to be simultaneous. The results show no clear advantage in cost, and a clear disadvantage in running time. It is too soon to abandon regularity between forms, but the evidence of Table 3 is tending against it.

4 The global tixel matching

A timetabling problem is a market in which resources are demanded by events and supplied to them. The unit of supply is one resource at one time, called a *supply tixel*. The term 'tixel' has been coined by the author by analogy with the 'pixel', one cell of a graphical display.

Each event demands a number of tixels of certain types. For example, a typical event called *7A-English*, in which class *7A* studies English for 6 times per cycle, demands 18 tixels: six tixels of class resource *7A*, six tixels of teachers qualified to teach English, and six of ordinary classrooms. This event is said to contain 18 *demand tixels*.

The market is represented by an unweighted bipartite graph. Each demand tixel is a node; each supply tixel is a node. An edge joins demand tixel d to supply tixel s when s may be assigned to d. For example, a demand tixel demanding class resource *7A* would be connected to the supply tixels for resource *7A* (one for each time in the cycle). A demand tixel demanding an English teacher would be connected to each supply tixel of each English teacher.

Each demand tixel requires only one supply tixel. Each supply tixel can be assigned to only one demand tixel, otherwise there would be a timetable clash. Accordingly, a set of assignments is a *matching* in this graph: a set of edges such that no two edges share an endpoint. There is an efficient algorithm for finding a maximum matching (one with as many edges as possible) [14].

There may be many maximum matchings, but they all fail to assign supply tixels to the same number of demand tixels, and since that number is the important thing, it is convenient to pretend that there is just one maximum matching. The author calls it the *global tixel matching*. The important number is a lower bound on the number of unassigned demand tixels in any solution, given the decisions already made. The matching defines an assignment which maximises the number of tixels assigned, but it is not useable directly, because it violates many constraints.

When a meet is assigned, the sets of edges connected to its demand tixels (their *domains*) shrink. For example, the six tixels demanding resource *7A* in the meets of event *7A-English* are initially connected to all the supply tixels for *7A* (one for each time of the cycle), but after times are assigned, each becomes associated with a particular time, and is connected to just the supply tixel for *7A* at that time. Tixel domains also change when the domain of a meet or task is changed. KHE keeps them up to date automatically.

Use of the global tixel matching is optional. KHE14 installs it during its structural phase and retains it until the end. It uses it in two ways. First, while times are being assigned, each unmatched demand tixel adds hard cost 1 to the total cost, guiding the solver away from assignments that would lead to problems later. For example, assigning 6 Science meets to some time, demanding 6 Science laboratories then, will be penalized if there are only 5 Science laboratories, even though no room assignments are made. Second, while resources are being assigned, only assignments that do not increase the number of unmatched demand tixels are permitted (until the end, when a last-ditch attempt is made to assign any remaining unassigned tasks). The rationale is that unmatched demand tixels lead inevitably to defects, and if their number is allowed to increase, there is little hope of reducing it again.

Additional demand tixels are added based on hard unavailable times, limit busy times, and limit workload constraints. For example, if teacher Smith is limited to at most 7 busy times out of the 8 times on Monday, then one demand tixel demanding Smith at a Monday time is added. This section is adapted from [10], which contains much more detail: how these additional tixels are defined, how to implement the matching efficiently, and so on.

5 Polymorphic ejection chains

Like most timetabling solvers, KHE14 first constructs, then repairs. Most of the repair work is done by *ejection chains*. An ejection chain is a sequence of one or more *repair operations* (also called *repairs*), which are often simple operations such as moves and swaps. The first repair removes one *defect* (a specific fault in the solution) but may introduce another; the next repair removes that defect but may introduce another; and so on. A key point is that the defects that appear as a chain grows are not known to have resisted attack before. It might be possible to repair one of them without introducing another, bringing the chain to a successful end.

Ejection chains are not new. They are the augmenting paths of matching algorithms, and they occur naturally to anyone who tries to repair a timetable by hand. They were brought into focus and named by Glover [3], in work on the travelling salesman problem. In timetabling, they have been applied to nurse rostering [2], resource assignment [10], and time repair [4, 5, 11].

A key insight of [11] is that ejection chains are naturally *polymorphic*: each defect along one chain can have a different type from the others, calling for a correspondingly different type of repair. Thus, any number of types of defects, and any number of types of repairs, can be handled together. In KHE, there is one defect type for each constraint type, representing one specific point in the solution where a constraint of that type is not satisfied, plus one defect type representing one specific unmatched demand tixel in the global tixel matching.

An ejection chain algorithm uses a set of functions, one for each kind of defect, called *augment functions* since they are based on the function for finding an augmenting path in bipartite matching [14]. An augment function is passed a specific defect of the type it handles, and it tries a set of alternative repairs on it. Each repair removes the defect, but may create new defects. If no significant new defects appear, the augment function terminates successfully, having reduced the solution cost. If one significant new defect appears (one whose removal would reduce the solution cost below its value when the chain began; it may cost more than the previous defect), it makes a recursive call to the appropriate augment function for that defect in an attempt to remove it. In this way a chain of coordinated repairs is built up. If the recursive call does not succeed in improving the solution, or was not attempted because two

Table 4 Effectiveness of variants of KHE14's ejection chain algorithm. Each pair of characters represents one complete restart of the algorithm: a digit denotes a maximum chain length (u means unlimited); + denotes allowing entities to be revisited along one chain, and - denotes not allowing it. KHE14 uses 1+,u-. Other details as previously.

Instance	C:u-	C:1+,u-	C:1+,2+,u-	T:u-	T:1+,u-	T:1+,2+,u-
AU-BG-98	9.00571	12.00491	**8.00500**	**22.6**	25.4	24.4
AU-SA-96	**14.00024**	16.00014	17.00027	81.1	**72.6**	76.2
AU-TE-99	2.00130	4.00124	**2.00090**	3.9	3.5	**3.1**
BR-SA-00	1.00072	1.00057	**1.00054**	1.0	**0.8**	1.5
BR-SM-00	**25.00135**	29.00093	27.00135	10.1	**6.4**	10.8
BR-SN-00	4.00252	**4.00243**	5.00246	3.7	2.9	**2.0**
DK-FG-12	0.02390	**0.02248**	0.02347	**160.8**	180.2	302.9
DK-HG-12						
DK-VG-09	**12.03206**	12.03349	12.03498	544.1	**375.8**	756.8
ES-SS-08	1.02081	**0.01362**	1.02639	17.1	**16.0**	43.1
FI-MP-06	4.00120	3.00132	**3.00117**	12.8	**7.8**	9.0
FI-PB-98	5.00051	6.00039	**4.00056**	5.5	6.3	**4.7**
FI-WP-06	**0.00049**	0.00078	0.00079	1.5	1.3	**1.2**
GR-H1-97	**0.00000**	**0.00000**	**0.00000**	6.2	**5.9**	5.9
GR-P3-10	2.00047	2.00088	**0.00009**	37.5	22.6	**12.4**
GR-PA-08	**0.00035**	0.00040	0.00040	**6.2**	7.2	6.9
IT-I4-96	1.01111	**0.00494**	0.00512	**7.8**	8.1	8.0
KS-PR-11	**0.00130**	0.00150	0.00135	77.6	**77.0**	78.2
NL-KP-03	**0.03707**	0.04774	0.04547	232.2	**197.8**	226.2
NL-KP-05						
NL-KP-09						
UK-SP-06	2.00144	**0.00102**	0.00182	46.8	**30.0**	34.9
ZA-LW-09	**25.00000**	26.00000	26.00000	**18.1**	20.2	18.7
ZA-WD-09	11.00000	**9.00000**	**9.00000**	26.5	24.0	**18.9**
Average	5.00678	**5.00660**	5.00724	63.0	**52.0**	78.4

or more significant new defects appeared, the augment function undoes the repair and continues with alternative repairs. It could try to remove a whole set of new defects, but that would rarely succeed in practice.

The main loop of the algorithm repeatedly iterates over the defects of the solution, or over a subset of them that it is expedient to target, and calls the appropriate augment function on each. It terminates when one complete pass over all defects yields no reduction in solution cost.

KHE offers two methods for preventing the tree of repairs searched by an augment function from growing to exponential size: either the length of the chains is limited to at most some fixed constant, or else it is unlimited, but entities visited while searching for one chain are marked, and revisiting them is prohibited, limiting the size of one search to the size of the solution.

Table 4 investigates these options for limiting the search. KHE14's choice has the lowest average cost and run time, but there is no clear signal.

Entities are marked in a way that prevents repairs on other chains, or further along the same chain, from targeting that entity, while allowing any number of alternative repairs of the entity to be tried where it is marked. Only the first entity changed by each repair is marked. Other entities changed by it are neither checked for being marked nor marked. This is important, since one

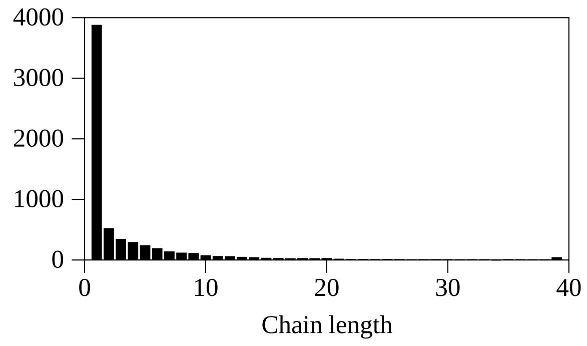


Fig. 1 For each chain length, the number of successful chains of that length found during time repair, over all tested instances of XHSTT-2013. There were 6322 successful chains altogether, and their average length was 3.5. All successful chains of length greater than 39 are shown as having length 39. The longest successful chain had length 140.

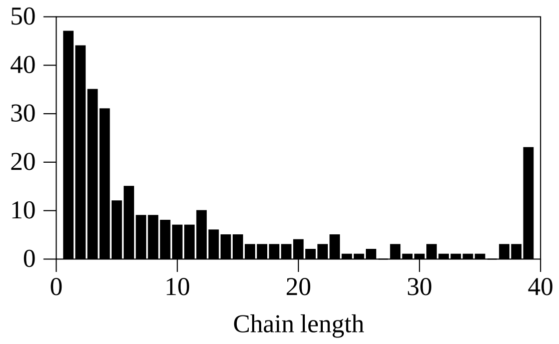


Fig. 2 For each chain length, the number of successful chains of that length found during resource repair, over all tested instances of XHSTT-2013. There were 321 successful chains altogether, and their average length was 11.5. All successful chains of length greater than 39 are shown as having length 39. The longest successful chain had length 105.

of them is likely to be targeted next. As long as at least one entity is marked by each repair, the size of the search will be limited as desired.

KHE14 makes two kinds of calls to the ejection chain algorithm: *time repair* calls, which repair time assignments, and *resource repair* calls, which repair resource assignments. Fig. 1 shows how long successful time repair chains are, and Fig. 2 does the same for resource repair. Most are short, but some are long. The average length of resource repair chains is surprisingly high. It was shown in [11] that chain lengths tend to increase as the algorithm progresses.

The text of this section is adapted from [11], which also explains how ejection chains are implemented in KHE. The user writes one augment function for each defect type, which iterates over the alternative repairs, applying and unapplying each in turn. KHE supplies the main loop, chaining together of individual repairs, testing for success, and dynamic dispatch by defect type.

6 Repair operations

A *repair operation*, or just *repair*, is a change to the solution made in the hope of removing a specific defect. This section presents the repairs used by KHE14's ejection chain algorithms. A detailed description of two unusual repairs is given first, followed by a brief description of the full set. Sect. 7 explains how they are used to repair the various kinds of defects.

Node swaps. A node is a set of meets, grouped together because they make up one course (Sect. 3). Suppose two nodes have meets of the same durations (one of duration 2 and four of duration 1, say). A node swap swaps the starting times of corresponding meets in those nodes.

Pairs of swappable nodes are common, since it simplifies planning if many courses have equal duration, and courses of equal duration often split into meets of equal durations. Nodes are only swapped when they have the same preassigned resources, so swapping avoids introducing clashes involving those resources. Swapping nodes has some advantages over swapping meets: it preserves regularity, and tends not to create new spread events defects.

Kempe meet moves. These begin with the move of a meet from its current time t_1 to some other time t_2. If that causes clashes between preassigned resources at t_2, the other meets involved in the clashes are moved to t_1, any clashes produced by those moves cause more meets to be moved to t_2, and so on until there are no new clashes and no more moves.

When a Kempe meet move succeeds, the result is usually a simple move or swap. Having a single operation which could turn out to be either is convenient, since it allows a solver to try moving a problem meet to each time t_2, whether the affected resources are free then or not.

Node swaps and Kempe meet moves are implemented more generally than described here, including support for hierarchical timetabling and preserving regularity. Kempe meet moves can swap meets of different durations when they are adjacent in time. A full description appears in [11], except for one recent improvement, in how regularity is preserved [12].

Turning now to the full list of repair operations, let a *variable* be a meet or a task, considered as an entity requiring a time or resource to be assigned to it. An *assignment* is a change to a variable from unassigned to assigned. A *move* is a change to a variable from one assignment to a different assignment. An *unassignment* is a change to a variable from assigned to unassigned.

When the change is an assignment or move, the new value of the variable is likely to create conflicts (timetable clashes) with other variables. There are at least four ways to handle these conflicts. The *basic* way is to do nothing, leaving it to the ejection chain algorithm to notice the resulting defects and try to repair them. The *ejecting* way is to unassign conflicting variables. This will be better than the basic way if it produces a single defect (an assign time or assign resource defect) rather than several defects whose common cause may not be clear to the ejection chain algorithm. The *swap* way, applicable only to moves, is to move the conflicting variables in the opposite direction. The

Kempe way, also applicable only to moves, is to continue swapping back and forth to remove conflicts, as explained above for Kempe meet moves.

Ignoring unassignment, which seems to be not useful alone, this makes six operations altogether: *basic assignment*, *ejecting assignment*, *basic move*, *ejecting move*, *swap*, and *Kempe move*. Applying them to meets and tasks gives twelve operations. KHE14 uses most of them, plus two operations which are sets of these ones treated as a unit: node swaps and *ejecting task-set moves*, which are sets of ejecting task moves to a common resource.

An ejecting move is a Kempe move that ends early, as soon as the variables to be moved in the opposite direction are unassigned. It often makes sense to first try a Kempe move, then fall back on an ejecting move; this is similar to trying a particular reassignment of the unassigned variables first. The term *Kempe/ejecting move* refers to a sequence of one or two repairs, first a Kempe move, then an optional ejecting move with the same parameters. The ejecting move is omitted when the Kempe move (successful or not) does not try to move anything back in the opposite direction, since the two repairs are identical then.

Kempe meet moves are useful because instances often contain preassigned class resources which are busy for all or most of the cycle. Moving a meet containing such a resource practically forces another meet to move the other way, so it makes sense to get on and do it. Kempe task moves are less useful because they apply to unpreassigned resources, such as teachers and rooms, which are less constrained. Ejecting moves seem more appropriate for them.

Table 5 investigates various combinations of Kempe, ejecting, and basic meet moves. The results here are clear: Kempe meet moves are helpful, and the Kempe/ejecting combination is better than the Kempe/basic alternative when run time is included in the comparison.

These repair operations are not original to this paper. The author has used node swaps and Kempe meet moves before [11], and ejecting task moves [10], but not ejecting meet moves.

7 Repairing defects

This section explains how the ejection chain algorithm repairs defects using the repair operations of Sect. 6. For each kind of defect, this section defines a set of repairs. The augment function for that kind of defect applies the first of these repairs, calls a KHE function to test for success and recurse, then either returns 'success' immediately or unapplies that repair and tries the next, and so on, returning 'no success' when all repairs are tried without success.

KHE has objects called *monitors*, each monitoring one point of application of one constraint, or one demand tixel in the global tixel matching. Each monitor contains a cost. When some part of the solution changes, KHE notifies the monitors affected by that part, and they revise their evaluation and perhaps change their cost. Any cost changes are reported and cause the overall solution cost to change. For example, when a time is assigned to a meet, any affected

Table 5 Kempe, ejecting, and basic moves during time assignment. Where the main text states that Kempe meet moves are tried, K means to try them and X means to omit them. Where it states that ejecting meet moves are tried, E means to try them and B means to try basic meet moves instead. KHE14 uses KE. Other details as previously.

Instance	C:KE	C:KB	C:XE	C:XB	T:KE	T:KB	T:XE	T:XB
AU-BG-98	12.00491	19.00390	**11.00490**	19.00473	26.5	39.5	18.8	41.9
AU-SA-96	16.00014	**4.00013**	25.00106	34.00060	73.9	417.6	52.3	112.2
AU-TE-99	**4.00124**	9.00133	8.00172	11.00153	3.5	10.1	4.0	3.3
BR-SA-00	**1.00057**	3.00075	1.00081	6.00066	0.8	0.6	1.3	0.3
BR-SM-00	29.00093	28.00093	32.00120	**26.00084**	6.4	1.5	4.8	1.2
BR-SN-00	**4.00243**	7.00258	10.00267	11.00231	2.8	1.8	4.5	0.8
DK-FG-12	0.02248	**0.01893**	0.02359	0.02565	192.9	370.0	122.5	88.1
DK-HG-12								
DK-VG-09	12.03349	**12.03077**	12.03424	14.03683	384.5	698.4	297.6	133.8
ES-SS-08	**0.01362**	0.02662	0.01771	0.04053	15.8	30.4	13.0	15.0
FI-MP-06	3.00132	**1.00123**	2.00107	6.00120	7.5	9.7	8.3	8.1
FI-PB-98	6.00039	**2.00173**	7.00022	6.00144	6.3	9.9	7.4	5.9
FI-WP-06	0.00078	1.00069	**0.00036**	3.00102	1.2	3.4	2.8	1.6
GR-H1-97	**0.00000**	**0.00000**	2.00000	2.00000	5.9	5.9	6.2	6.0
GR-P3-10	2.00088	**0.00074**	12.00150	1.00105	22.7	9.0	59.2	7.0
GR-PA-08	0.00040	**0.00035**	0.00043	2.00111	5.7	6.4	8.1	5.5
IT-I4-96	**0.00494**	1.00711	0.00605	0.00651	8.4	7.6	8.0	5.9
KS-PR-11	0.00150	0.00294	**0.00149**	2.01257	77.6	82.1	79.6	80.2
NL-KP-03	0.04774	0.04111	**0.03340**	0.04635	205.5	922.5	327.0	161.8
NL-KP-05								
NL-KP-09								
UK-SP-06	**0.00102**	2.00084	5.00324	12.00608	28.9	146.5	176.7	28.7
ZA-LW-09	**26.00000**	30.00000	29.00000	31.00000	18.0	13.7	18.2	15.1
ZA-WD-09	**9.00000**	11.00000	9.00001	17.00000	23.6	34.9	88.0	10.9
Average	**5.00660**	6.00679	7.00646	9.00909	53.3	134.4	62.3	34.9

assign time and prefer times monitors are notified, and they change their cost accordingly. Concretely, a defect is a monitor whose cost is non-zero.

Monitors may be *grouped*: joined into sets treated as single monitors whose cost is the sum of the individual costs. KHE14's grouped monitors have the same kind and monitor the same thing in reality (examples appear below), and are repaired by repairing any one member of the group. Monitors may also be *detached*: fixed to cost 0 regardless of their true cost. Grouping and detaching are used to prevent the algorithm from being confused by apparently distinct defects which really point to the same problem. Such defects could cause a chain to end when there is a worthwhile repair to continue with.

(One application of ejection chains to timetabling [4] found a way to exploit a coarser grouping than any used here: it groups all defects related to one resource. The elements of such a group may have different types, so it does not make sense to perform repairs specific to any one type. Instead, all moves of a meet to which the resource is assigned to a time when the resource is free are tried. From those moves which introduce at most one new conflict, the 20 best are selected and used as the set of repairs for the group.)

As mentioned earlier, there are two categories of calls on the ejection chain repair function: *time repair* calls, which repair the assignments of times to meets, and *resource repair* calls, which repair the assignments of resources to tasks. Each category has its own augment functions, and its own way of grouping and detaching monitors. Both categories are defined below.

Demand defects. These are cases of unmatched demand tixels in the global tixel matching (Sect. 4), usually indicating that the demand for some set of resources at some time exceeds their supply then. The defect is not really the one unmatched tixel, but rather the whole set of demand tixels that contribute to the excess demand. Given the nondeterminism of the global tixel matching, any one of these could be reported as the defect. Repair operations use KHE functions to visit them all, and, in effect, repair the set, not the individual.

During time repair, demand monitors lying within tasks of the same meet are grouped, so that multiple demand defects that can be repaired by moving that meet are perceived as a single defect. Defects are handled by trying all repairs that move any of the meets contributing to the excess demand away from the problem time. Kempe/ejecting meet moves are tried first, then node swaps between nodes with similar preassigned resources. For example, 6 Science meets running simultaneously when there are only 5 Science laboratories will generate one demand defect which will cause repairs to be tried that move any of the 6 meets away from the problem time. Simple clashes also produce demand defects, and are repaired in the same way.

Demand monitors derived from the same avoid unavailable times, limit busy times, or workload limit monitor are grouped. If any of these are involved in a demand defect (if they contribute to the excess demand), then the repairs just given are not well targeted, because they could move a contributing meet to a different time within the times whose limit has been exceeded, or swap a meet back into those times. So different repairs are tried in that case: ejecting meet moves that move any of the contributing meets away from the times whose limit has been exceeded. For example, if teacher Smith is preassigned to tasks which give him 8 busy times on Monday when he is limited to 7, the demand monitor derived from this limit will contribute to the excess demand, causing ejecting moves to be tried that move meets preassigned to Smith away from Monday. Similarly, a demand monitor derived from an avoid unavailable times monitor will cause ejecting moves away from the unavailable times.

During resource repair, demand defects are handled differently. They do not contribute to the cost of the solution, and they are not repaired. Instead, ejection chains which increase their number are not applied (except right at the end, when a last-ditch attempt is made to assign any remaining unassigned tasks). The reason for this is as follows.

At the start of each call to the resource repair ejection chain algorithm, the number of demand defects is equal to what it was at the end of time assignment. This is because no resource assignments are accepted which increase this number, and resource assignments which decrease it are impossible, since a resource assignment reduces the domains of demand nodes, reducing the choice of matchings. (If resource repairs changed meet assignments, that could reduce the number of demand defects; but it is not likely to, because similar changes were tried during time repair, at a point when fewer resources were assigned so more choices were open.) Repair of demand defects is not attempted, then, because it is doomed to failure. This also explains why chains which increase the number of demand defects are not applied: such an increase would be almost impossible to reverse later. This argument is from [10].

Split events defects and distribute split events defects. These are cases of events split into too few or too many meets, or into meets whose durations are not wanted. Event splitting is handled by the structural phase (Sect. 3), and these defects are ignored during time and resource repair.

Assign time defects. These are cases of meets not assigned a time. There are usually none when time repair begins, because the initial time assignment usually assigns a time to every meet; but ejecting meet moves create them. They are handled during time repair only, by trying all ejecting meet assignments to times in the meet's domain. Kempe meet moves are not possible (there is no original time). Assign time monitors whose events are joined by link events constraints handled structurally are grouped; they monitor the same thing.

It is important to assign a time to every meet, so assign time defects are also treated during time repair like demand defects are treated during resource repair: ejection chains may unassign meets temporarily as they go, but no chain is applied which, in the end, increases the total cost of assign time defects.

Prefer times defects. These are cases of meets assigned unwanted times: for example, a Physics meet that prefers a morning time but is assigned an afternoon time. They are handled during time repair only, by trying all Kempe/ejecting meet moves of the meet to a preferred time. Prefer times monitors whose events are joined by link events constraints handled structurally are grouped when they request the same times.

Spread events defects. These are cases where the meets of a course are spread unevenly through the cycle. For example, two might occur on Monday, and none on Thursday. They are handled during time repair only, by trying all Kempe/ejecting meet moves which remove the defect (in the example, all moves of a Monday meet to a Thursday time). Spread events monitors whose events are joined by link events constraints handled structurally are grouped.

Link events defects. These are cases where events which should occur at the same time do not. Like event splitting, event linking is handled structurally (Sect. 3), and these defects are ignored during time and resource repair.

Order events defects. These are cases where events should appear in a particular time order, but they don't. KHE14 ignores these defects, because there are no order events constraints in the XHSTT-2013 data set. However, it is easy to find repairs that remove them, so there will be no problem in handling them in future. Order events monitors whose events are joined by link events constraints handled structurally are grouped.

Assign resource defects. These are cases where a task is not assigned a resource. They are handled during resource repair, by trying all ejecting task assignments to resources in the task's domain. Assign resource monitors whose tasks are joined by avoid split assignments constraints handled structurally are grouped while those structures are present.

Prefer resources defects. These are cases where a task is assigned a resource it does not prefer: for example, the room task of a Science meet assigned an ordinary classroom instead of a Science laboratory. They are handled during resource repair, by trying all ejecting task moves to the preferred resources. Kempe task moves are possible, but (as discussed above) they seem unlikely to be useful and have not been tried. Prefer resources monitors whose tasks are joined by avoid split assignments constraints handled structurally are grouped while those structures remain in place, if they request the same resources.

Avoid split assignments defects. These are cases where tasks are assigned different resources, when they want the same resource. For example, a Music event split into five meets, four taught by Smith and one taught by Brown, is an avoid split assignments defect, also called a *split assignment.*

Structures which prohibit split assignments are present for most of KHE14, but near the end they are removed and split assignments are constructed (they are usually better than nothing). These split assignments are repaired by taking each pair of resources assigned to the tasks, finding the subset of the tasks assigned those resources, and trying each ejecting task-set move of that subset to a resource from the domain of one of them. These are the smallest repairs capable of reducing cost, which is prudent, given the difficulty.

A promising alternative kind of repair, not yet implemented, is to pick one of the resources currently assigned to some of the tasks, whose workload limit permits it to be assigned to all of them, and try to move the meets of the other tasks to times when that resource is free. Previous phases would need to ensure that split assignments were concentrated in meets that demand few resources, making them more likely to be movable.

Avoid clashes defects. These are cases where a resource attends two meets at the same time. During time repair, avoid clashes monitors are detached: demand monitors do their job and more. During resource repair, they are handled by trying all ejecting task moves of the clashing tasks to resources in their domains. There is no confusion with demand defects, because of the way that demand monitors are handled during resource repair (see above).

Avoid clashes monitors for resources of the same type are grouped when they are derived from the same constraint, all the event resources of their type are preassigned, and the resources are preassigned to the same events, so follow the same timetable. The saving can be significant: in the NL-KP-03 instance, for example, there are 453 resources representing individual students, but only 297 groups, or 285 when link events constraints are taken into account.

Avoid unavailable times defects. These are cases where a resource attends a meet at a time when it is unavailable. An avoid unavailable times constraint is the same as a limit busy times constraint whose set of times is the unavailable times, and whose `Maximum` attribute is 0. So these defects are handled as described below for limit busy times defects, including grouping.

Limit idle times defects. These are cases where a resource's timetable contains an idle time: a time when the resource is not *busy* (attending an event), but is busy earlier that day and later. During time repair, they are handled by taking each meet assigned to the resource which occurs at the beginning or end of one of its days, and trying each ejecting move of it to a time that reduces the number of idle times and does not cause clashes. A move to any non-clashing time between the first and last busy times on any day is acceptable. If the meet is adjacent to an idle time, then moving it far away will remove that idle time, so it may also be moved to just before the first busy time of any day, and just after the last busy time of any day, provided that first or last busy time is not one when the meet itself is currently running. Limit idle times monitors are grouped in the same way as avoid clashes monitors. During resource repair, limit idle times defects are ignored; repairing them then is future work.

Cluster busy times defects. These are cases where a resource is busy on too few or too many days. During time repair, if the problem is too few days, then all ejecting moves are tried which move a meet from a day in which it is not the only meet to a day in which it is. If the problem is too many days, then ejection chains struggle, because cost is reduced only when a day becomes completely free, which may require several meet moves. The present repair tries all ejecting moves of meets which are alone in their day to days when other meets are present. Some defects can be repaired in this way, but many cannot; they are future work, as are repairs during resource repair. Cluster busy times monitors are grouped in the same way as avoid clashes monitors.

Limit busy times defects. These are cases where a resource is overloaded or underloaded during some set of times, typically one day. For example, teacher Jones might expect to be busy for only at most 7 of the 8 times on any day; anything more is a limit busy times defect.

Break each limit busy times monitor into two monitors, one monitoring underloads and the other overloads. During time repair, overload monitors that give rise to demand monitors that do all their work are detached. Other overload monitors and all underload monitors are not. They are handled by trying all ejecting meet moves of one of the resource's meets from inside the set of times to outside it, or vice versa, depending on whether the defect is an overload or an underload.

During resource repair, all these monitors are attached. There is no confusion with demand monitors, as explained above for avoid clashes defects. Overloads are handled by trying all ejecting task moves which unassign the resource from any meet running during the set of times. Underloads are ignored; repairing them during resource repair is future work. Limit busy times monitors are grouped in the same way as avoid clashes monitors.

Limit workload defects. These are similar to limit busy times defects whose times are the whole cycle, and are handled in the same way, including grouping.

Repairs targeted at specific defects are rare in the timetabling literature. About half of the above is previous work [10, 11] and half is new. Disentangling new from old would be tedious, but this paper is the first to tackle anything approaching a complete set of defect types with polymorphic ejection chains.

Which augment functions are most effective? Finding a good measure of effectiveness is not easy. For example, virtually any defect can be removed if enough mayhem is visited on its surroundings, so success in removing defects, taken in isolation, is a poor measure. One simple approach, not claimed to be

Table 6 Effectiveness of time augment functions. For each time augment function, the number of calls to the function, the number of successful calls, and the ratio of the two as a percentage, over all tested instances of XHSTT-2013. Only non-zero rows are shown.

Augment function	Total	Successful	Percent
Assign time	3770681	12890	0.3
Spread events	440720	3723	0.8
Ordinary demand	177379	797	0.4
Workload demand	8439	63	0.7
Avoid unavailable	48428	367	0.8
Limit idle	847642	3447	0.4
Cluster busy	253699	333	0.1
Limit busy	652935	737	0.1

Table 7 Effectiveness of resource augment functions. For each resource augment function, the number of calls to the function, the number of successful calls, and the ratio of the two as a percentage, over all tested instances of XHSTT-2013. Only non-zero rows are shown.

Augment function	Total	Successful	Percent
Assign resource	500023	2032	0.4
Avoid splits	6946	182	2.6
Avoid clashes	94	0	0.0
Limit busy	24523	133	0.5
Limit workload	170855	1343	0.8

perfect, is to say that a call on an augment function is effective when it returns `true`, meaning that it lies on a chain that improved the solution, and ineffective when it returns `false`. The ratio of effective to effective plus ineffective calls, expressed as a percentage, measures the effectiveness of the function.

Table 6 presents the number of calls on each time repair augment function used by KHE14 when solving XHSTT-2013, and their effectiveness, measured as just described. Table 7 does the same for resource repair. Interpretation is problematical, but it seems, for example, that the time repair functions for cluster busy times and limit busy times defects are relatively ineffective.

Which repairs are most effective? Again, finding a good measure is not easy. For example, on any given defect one kind must be tried first, and this gives it more opportunities to both succeed and fail than the others. Again, a simple approach is used: the successful calls on a given augment function are attributed to the repairs that caused the successes.

Table 8 is like Table 6 except that it contains one row for each kind of repair of each kind of time defect, measured in this way. Some of the results are quite interesting: the tiny number of successful node swaps, for example. The corresponding results for resource repairs are omitted, since each resource repair augment function tries just one kind of repair operation.

8 The algorithm

This section describes the KHE14 algorithm at a high level. An implementation is available online (function `KheGeneralSolve2014` of [12]).

Table 8 Effectiveness of time repair operations. For each time augment function and repair operation, the number of calls on that repair operation made by that augment function, the number of successful calls, and the ratio of the two as a percentage, over all tested instances of XHSTT-2013. Only non-zero rows are shown.

Augment function : Repair operation	Total	Successful	Percent
Assign time : Basic meet assignment	3	3	100.0
Assign time : Ejecting meet assignment	3770675	12884	0.3
Assign time : Basic meet move	3	3	100.0
Spread events : Basic meet move	1	0	0.0
Spread events : Ejecting meet move	254101	249	0.1
Spread events : Kempe meet move	186618	3474	1.9
Ordinary demand : Basic meet move	27	2	7.4
Ordinary demand : Ejecting meet move	121656	64	0.1
Ordinary demand : Kempe meet move	54942	726	1.3
Ordinary demand : Node swap	754	5	0.7
Workload demand : Ejecting meet move	8439	63	0.7
Avoid unavailable : Ejecting meet move	48428	367	0.8
Limit idle : Ejecting meet move	847642	3447	0.4
Cluster busy : Ejecting meet move	253699	333	0.1
Limit busy : Ejecting meet move	652935	737	0.1

KHE14 proceeds in *phases* (major steps). First comes the *structural phase.* It constructs an initial solution with no time or resource assignments, converts resource preassignments (in the instance) into resource assignments (in the solution), adds additional structure as described in Sect. 3, and adds the global tixel matching as described in Sect. 4.

Next comes the *time assignment* phase, which assigns a time to each meet. It has been described fully elsewhere [6, 7, 11]; here is an overview. For each resource to which a hard avoid clashes constraint applies it builds one *layer*, the set of all nodes containing meets preassigned that resource. After discarding layers that are redundant because they are subsets of other layers, and sorting so that (heuristically) the most difficult layers come first, it assigns times to the meets of each layer in turn. The algorithm for assigning times to the meets of one layer is heuristic and complex. It tries for regularity with previously assigned layers, and exploits the fact that the meets of one layer should not overlap in time, by maintaining a minimum-cost matching of meets to times.

A node may lie in several layers, if its meets contain several preassigned resources. Such a node is handled with the first layer it lies in, and the result is not changed when assigning subsequent layers. So when a layer's turn comes to be assigned, all its nodes may be already assigned. Such layers are skipped.

After each unskipped layer is assigned, an ejection chain repair (Sect. 5) is applied. Its main loop is targeted at the defects of the layer just assigned, but its recursive calls may spread into earlier layers. After all layers have been assigned and repaired, another ejection chain repair is carried out, targeted at the entire time assignment. Then the structures which enforce regularity in time are removed and yet another ejection chain time repair is run.

Next come the *resource assignment* phases, one for each type of resource (teacher, room, etc.). These phases are sorted heuristically so that the most

Table 9 Effectiveness of KHE14 and KHE14x8. Details as previously. Different solutions to one instance vary in run time, so finding eight solutions on a quad-core machine often takes more than twice as long as finding one.

Instance	C:KHE14	C:KHE14x8	T:KHE14	T:KHE14x8
AU-BG-98	12.00491	**4.00524**	24.9	43.8
AU-SA-96	16.00014	**6.00006**	72.2	172.3
AU-TE-99	4.00124	**2.00140**	3.6	7.6
BR-SA-00	1.00057	**1.00051**	0.8	1.9
BR-SM-00	29.00093	**22.00129**	6.4	15.2
BR-SN-00	**4.00243**	**4.00243**	2.9	6.7
DK-FG-12	0.02248	**0.02046**	175.5	404.0
DK-HG-12				
DK-VG-09	12.03349	**12.03257**	373.1	919.4
ES-SS-08	0.01362	**0.01287**	14.8	31.1
FI-MP-06	3.00132	**0.00125**	7.7	16.4
FI-PB-98	6.00039	**1.00024**	6.1	12.9
FI-WP-06	0.00078	**0.00041**	1.2	5.4
GR-H1-97	**0.00000**	**0.00000**	6.2	13.2
GR-P3-10	2.00088	**0.00006**	22.8	51.8
GR-PA-08	0.00040	**0.00021**	7.2	19.5
IT-I4-96	0.00494	**0.00197**	7.9	20.1
KS-PR-11	0.00150	**0.00116**	76.4	173.3
NL-KP-03	0.04774	**0.03919**	190.4	698.7
NL-KP-05				
NL-KP-09				
UK-SP-06	0.00102	**0.00056**	30.3	88.2
ZA-LW-09	26.00000	**16.00000**	21.7	34.5
ZA-WD-09	9.00000	**6.00000**	25.0	50.1
Average	5.00660	**3.00580**	51.3	132.7

difficult come first. In practice, teachers are assigned first (if needed), then rooms; students and classes are not assigned, since they are all preassigned, and so were assigned during the structural phase.

Each resource assignment phase has three parts. In the first part, which in practice assigns most tasks, violations of avoid split assignments and prefer resources constraints are prohibited structurally, and assignments that increase the number of demand defects (Sect. 4) are rejected. A resource assignment algorithm is called that tries to assign a resource to each unpreassigned task of the current type. If there are avoid split assignments constraints, a *resource packing* algorithm which follows a bin packing paradigm is used. Otherwise a constructive heuristic is used, more effectively than usual because of the guidance provided by the global tixel matching. These algorithms are described in detail in [10], where resource packing was found to be the best of three plausible resource assignment algorithms for teacher assignment. This first part ends with a call on the ejection chain resource repair algorithm, targeted at the event resource and resource defects of the current type.

The second part of the phase is only carried out for those types of resources whose event resources are subject to avoid split assignments constraints. It removes structures that prevent split assignments, finds split assignments for unassigned tasks (using a construction heuristic specialized for them), and

Table 10 Event defects in the solutions produced by KHE14x8. Each column shows the number of defects of one kind of event constraint. A dash indicates that the instance contains no constraints of that type. The columns appear in the same order as the rows of Table 1.

Instance	SS	DS	AT	PT	SE	LE	OE
AU-BG-98	0	0	0	0	30	0	-
AU-SA-96	0	0	0	0	6	0	-
AU-TE-99	0	0	0	-	8	0	-
BR-SA-00	0	0	0	0	0	-	-
BR-SM-00	0	0	4	0	2	-	-
BR-SN-00	0	0	0	0	0	-	-
DK-FG-12	-	-	0	-	41	0	-
DK-HG-12							
DK-VG-09	-	-	0	-	81	0	-
ES-SS-08	0	-	0	-	88	-	-
FI-MP-06	0	-	0	0	0	-	-
FI-PB-98	0	-	0	0	1	-	-
FI-WP-06	0	-	0	-	0	-	-
GR-H1-97	-	-	0	-	0	0	-
GR-P3-10	0	-	0	0	0	0	-
GR-PA-08	-	-	0	-	5	0	-
IT-I4-96	0	-	0	0	0	-	-
KS-PR-11	0	0	0	0	0	-	-
NL-KP-03	0	-	0	0	0	0	-
NL-KP-05							
NL-KP-09							
UK-SP-06	-	-	0	-	0	0	-
ZA-LW-09	0	-	3	0	-	0	-
ZA-WD-09	-	-	1	0	0	0	-
Total	0	0	8	0	262	0	

calls ejection chain repair again. Then it tries two VLSN search algorithms [1, 13] which sometimes find small improvements. One rearranges the resource assignments within a given set of times using min-cost flow, the other unassigns and optimally reassigns pairs of resources [9]. The details are in [12] as usual; they are omitted here because these algorithms are peripheral to the main thrust of this paper, which is already overlong.

The third part is a last-ditch attempt to assign as many of the remaining unassigned tasks as possible. It removes all structural prohibitions, removes the prohibition on increasing the number of unmatched demand tixels, and calls the ejection chain repair algorithm a third time.

The final *cleanup phase* carries out some minor tidying up. Whenever two meets derived from the same event have ended up adjacent in time, this phase merges them into one when that is possible and reduces cost. It also unassigns tasks and meets when that reduces cost.

Table 9 shows the overall performance of KHE14 and its variant KHE14x8, which runs KHE14 8 times in parallel and keeps a best solution. Random numbers are not used; instead, each run is given a different *diversifier* (a small fixed integer). It is used in several places, to vary the starting point of list traversals, and to break ties. For example, ejection chain algorithms sort their initial defects by decreasing cost; the diversifier influences the order of defects

Table 11 Event resource and resource defects produced by KHE14x8. Details as previously.

Instance	AR	PR	AS	AC	AU	LI	CB	LB	LW
AU-BG-98	2	0	42	2	0	-	-	18	0
AU-SA-96	1	0	0	3	0	-	-	0	0
AU-TE-99	0	0	12	2	0	-	-	1	0
BR-SA-00	-	-	-	1	0	8	2	-	-
BR-SM-00	-	-	-	11	1	8	11	-	-
BR-SN-00	-	-	-	4	0	16	16	-	-
DK-FG-12	0	0	-	0	0	56	113	64	-
DK-HG-12									
DK-VG-09	0	0	-	2	-	52	47	32	-
ES-SS-08	0	0	-	0	5	-	0	0	-
FI-MP-06	-	-	-	0	10	20	-	6	-
FI-PB-98	-	-	-	0	0	12	-	0	-
FI-WP-06	-	-	-	0	-	13	-	8	-
GR-H1-97	-	-	-	0	0	-	-	-	-
GR-P3-10	-	-	-	0	0	0	-	3	-
GR-PA-08	-	-	-	0	0	7	-	0	-
IT-I4-96	-	-	-	0	6	31	1	2	-
KS-PR-11	-	-	-	0	0	54	-	0	-
NL-KP-03	0	0	-	0	7	334	15	51	-
NL-KP-05									
NL-KP-09									
UK-SP-06	0	-	-	0	0	25	-	-	-
ZA-LW-09	-	-	-	1	-	-	-	-	-
ZA-WD-09	-	-	-	3	0	-	-	-	-
Total	3	0	54	29	29	636	205	185	0

of equal cost. These solutions are available from the KHE web page [12]. An analysis of the remaining defects appears in Tables 10 and 11.

9 Conclusion

No overall conclusion can be drawn yet: KHE14 is the author's first solver to address such a wide range of constraint types; it is work in progress and has not reached its full potential; and solutions to XHSTT-2013 made by other solvers were not available at the time of writing. Individual solutions are available, but comparing with them ignores an essential requirement of a good solver, namely robustness over many instances, so is deliberately not done here.

The author is currently exploring ideas for improvements in all phases of KHE14. Recently, he modified the structural phase to restrict meet domains to discourage cluster busy times defects. This produced a solution to instance NL-KP-03 with 9 cluster busy times defects and cost 0.02804, much better than the 15 and 0.04774 reported in the tables. The algorithm that assigns times to the meets of one layer was designed without regard to cluster busy times and limit idle times constraints, so it needs revision. That should make solving the Dutch instances run faster, since at present they are overwhelming the time repair algorithm with hundreds of limit idle times defects.

References

1. R. Ahuja, Ö. Ergun, James B. Orlin, and A. Punnen, A survey of very large-scale neighbourhood search techniques, Discrete Applied Mathematics, 123, 75–102 (2002)
2. Kathryn A. Dowsland, Nurse scheduling with tabu search and strategic oscillation, European Journal of Operational Research, 106, 393–407 (1998)
3. Fred Glover, Ejection chains, reference structures and alternating path methods for traveling salesman problems, Discrete Applied Mathematics, 65, 223–253 (1996)
4. Peter de Haan, Ronald Landman, Gerhard Post, and Henri Ruizenaar, A case study for timetabling in a Dutch secondary school, Practice and Theory of Automated Timetabling VI (Springer Lecture Notes in Computer Science 3867), 267–279, (2007)
5. Myoung-Jae Kim and Tae-Choong Chung, Development of automatic course timetabler for university, Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT'97), 182–186 (1997)
6. Jeffrey H. Kingston, A tiling algorithm for high school timetabling, Practice and Theory of Automated Timetabling V (Springer Lecture Notes in Computer Science 3616), 208–225 (2005)
7. Jeffrey H. Kingston, Hierarchical timetable construction, Practice and Theory of Automated Timetabling VI (Springer Lecture Notes in Computer Science 3867), 294–307 (2007)
8. Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, URL `http://www.it.usyd.edu.au/~jeff/hseval.cgi` (2010)
9. Jeffrey H. Kingston, Timetable construction: the algorithms and complexity perspective, Annals of Operations Research, DOI 10.1007/s10479-012-1160-z (2012)
10. Jeffrey H. Kingston, Resource assignment in high school timetabling, Annals of Operations Research, 194, 241–254 (2012)
11. Jeffrey H. Kingston, Repairing high school timetables with polymorphic ejection chains, Annals of Operations Research, DOI 10.1007/s10479-013-1504-3
12. Jeffrey H. Kingston, KHE web site, `http://www.it.usyd.edu.au/~jeff/khe` (2014)
13. Carol Meyers and James B. Orlin, Very large-scale neighbourhood search techniques in timetabling problems Practice and Theory of Automated Timetabling VI (Springer Lecture Notes in Computer Science 3867), 24–39 (2007)
14. Christos. H. Papadimitriou and Kenneth Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall (1982)
15. Nelishia Pillay, An overview of school timetabling research, PATAT10 (Eighth international conference on the Practice and Theory of Automated Timetabling, Belfast, August 2010), 321–335 (2010)
16. Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, Gerhard Post, David Ranson, and Henri Ruizenaar, An XML format for benchmarks in high school timetabling, Annals of Operations Research, 194, 385–397 (2012)
17. Gerhard Post, XHSTT web site, `http://www.utwente.nl/ctit/hstt/` (2011)
18. Gerhard Post, Luca Di Gaspero, Jeffrey H. Kingston, Barry McCollum, and Andrea Schaerf, The Third International Timetabling Competition, PATAT 2012 (Ninth international conference on the Practice and Theory of Automated Timetabling, Son, Norway, August 2012), 479–484 (2012)
19. G. Schmidt and T. Ströhlein, Timetable construction—an annotated bibliography, The Computer Journal, 23, 307–316, (1980)

A two-phase heuristic and a lexicographic rule for improving fairness in personnel rostering

Komarudin · Marie-Anne Guerry · Pieter Smet ·
Tim De Feyter · Greet Vanden Berghe

Abstract A fair allocation of workload to people is crucial for securing job satisfaction. Researchers have introduced numerous objectives and algorithms to represent and improve fairness in personnel rostering problems. These approaches should not ignore the roster quality that is influenced by personnel rostering constraints, such as maximum working times, minimum rest times, etc. The present paper proposes a new fairness objective and an effective two-phase heuristic for optimizing rosters, taking into consideration the established personnel rostering constraints and the fairness. The new fairness objective is based on a lexicographic rule that offers a beneficial trade-off between roster quality and fairness. The new heuristic is tested on real world data and the results show that fair rosters can be obtained without significantly decreasing the roster quality.

Keywords Personnel rostering · roster quality · fairness · two-phase heuristic · lexicographic evaluation

1 Introduction

Personnel rostering aims to produce a timetable for personnel that satisfies the coverage requirements in a predefined time period. Furthermore, the generated timetable should meet a variety of contractual and personal constraints. Generally, each constraint is specified with a certain weight, denoting its penalty value when violated. The *individual penalty* refers to the sum of penalties that can be associated with one member of personnel. The objective of the problem is to improve the

Komarudin · Marie-Anne Guerry
Vrije Universiteit Brussel, Department of Business Technology and Operations
E-mail: {komarudin, maguerry}@vub.ac.be

Tim De Feyter
KU Leuven, Department of Business Management Research
E-mail: tim.defeyter@kuleuven.be

Pieter Smet · Greet Vanden Berghe
KU Leuven, Department of Computer Science, CODeS
E-mail: {pieter.smet,greet.vandenberghe}@cs.kuleuven.be

roster quality by minimizing the total penalty (see e.g. Smet et al (2013)). This kind of personnel rostering problem arises in several domains, including health care, transportation, and security (Ernst et al, 2004).

In addition to the roster quality, Stolletz and Brunner (2012) advocate to also consider fairness of work assignment. Fairness can be seen as the degree to which individual penalties are balanced over all employees. An unfair roster is likely to result in larger differences in e.g. workload among employees, which could induce job dissatisfaction (Larrabee et al, 2003). It is thus important that a solution has both a high roster quality and a high fairness.

Several approaches have been proposed to improve fairness of personnel rosters. Bard and Purnomo (2005) consider fairness by specifying an upper limit on the individual penalty for each employee, determined by the individual penalty that he/she collected in the previous planning period. This approach does not actually balance work assignments over all employees in a certain period. Rather it allows to compensate for a low quality roster (i.e. low individual penalty) from the previous planning period.

Chiaramonte and Chiaramonte (2008) represent fairness by a ratio that is based on the standard deviation of the individual penalties. Similarly, Stolletz and Brunner (2012) use the range (the difference between the maximum and the minimum) of the individual penalties to balance the number of working times and work assignments over all employees. Several other objectives have been defined for improving both the roster quality and the fairness. Smet et al (2012b) compare three objectives: (1) the maximum individual penalty, (2) the absolute deviation of the individual penalties, and (3) the range of the individual penalties. Martin et al (2013) in addition introduce the sum of squared penalties.

The objectives defined by Smet et al (2012b) are generalizations of Chiaramonte and Chiaramonte (2008)'s and Stolletz and Brunner (2012)'s objectives, which can be used to improve fairness over different rostering periods. They are not restricted to specific constraints only, but they can be used to balance penalties for all types of constraints associated with an individual employee. They provide a general way to extend the established personnel rostering models with a fairness aspect. Furthermore they have the advantage that they preserve the value of weights that represent the degree of importance of the rostering constraints.

All previous studies consider fairness by aggregating the individual penalties. In contrast, the current paper introduces a new objective that employs a different methodology. The new objective is represented by a vector of all individual penalties. It facilitates high fairness through beneficial penalty trade-off between employees. Moreover, the new objective does not strive to level the individual penalties over all employees, since this is not always appropriate due to the heterogeneous nature of a workforce in terms of contractual and personal constraints (Komarudin et al, 2013).

In Section 2, we describe the personnel rostering problem in general and discuss examination criteria of the roster quality and the fairness. Martin et al (2013) compared different objectives in an optimization algorithm using Jain's fairness index (Jain et al, 1984). This index may not suitable since it depends on the magnitude of individual penalties. We show that if the difference between two individual penalties is kept the same, the index is likely to improve as the individual penalties increase. A good Jain's index value is thus not always the result of a fair roster, but it can also be caused by high individual penalties.

Therefore, similar to Bard and Purnomo (2005), we suggest to use three examination criteria that provide quantitative information for comparing rosters: (1) the total penalty, (2) the standard deviation of the individual penalties, and (3) the maximum individual penalty. These criteria provide a comprehensive assessment of a roster since they examine the roster quality, the fairness, and the most unattractive individual rosters. We will use these examination criteria to compare the different objectives.

In Section 3, we discuss several local search moves that are used for solving a personnel rostering problem with a fairness objective. These local search moves were introduced by Smet et al (2012b) and used by Martin et al (2013). When optimizing fairness, such moves may hinder the exploration for finding high quality rosters, since the use of a fairness objective may ignore a local search move that improves the roster quality but does not improve the fairness. For example, when the maximum individual penalty is used as the objective, a local search move that does not decrease the maximum individual penalty is likely to be rejected.

In order to deal with the contradictory objectives, we propose a two-phase heuristic. The first phase ignores the fairness and uses only the total penalty as the objective of the heuristic. In the second phase, one of the fairness objectives is used in order to obtain balanced work assignments. This approach allows the heuristic to reach high quality rosters in the search space before considering fairness.

In Section 4, we describe and analyze a series of computational experiments in order to investigate the effectiveness of the two-phase heuristic compared to the one-phase heuristic. Moreover, the roster quality and the fairness resulting from the experiments with different objectives are compared. The heuristics are applied to the personnel rostering model of Bilgin et al (2012) and they are evaluated on data from six hospital wards (Smet et al, 2012a).

Finally, the conclusion and suggestions for further research are discussed in Section 5.

2 The personnel rostering model and three examination criteria

The personnel rostering model is based on the work by Smet et al (2012a). A number of objectives that can be used for improving fairness are presented. We also discuss the limitations of the objectives and then propose a new objective that has several advantages. Furthermore, we explain three examination criteria for assessing the roster quality and the fairness.

2.1 The personnel rostering model

The personnel rostering problem has been formulated in several models (Ernst et al, 2004), which vary in formulation of the objective and the rostering constraints. This paper considers the model of Smet et al (2012a) which is formulated in a general way such that it can take into account a large set of rostering constraints. Due to this flexibility in modeling, it has been implemented in several Belgian hospitals.

The personnel rostering problem is defined as the problem of assigning personnel to shifts, subject to hard and soft constraints. The hard constraints define the feasibility of a solution while the soft constraints determine the quality of a solution. The hard and soft constraints of Smet et al (2012a)'s model are summarized in Table 1. A weight is associated with each soft constraint, denoting its penalty value for a violation. The objective of the problem is to satisfy all hard constraints and to minimize the total penalty.

Table 1 Hard and soft constraints of Smet et al (2012a)'s personnel rostering model

Hard constraints	Soft constraints
Single assignment per day	Coverage requirements
Assignment of defined requirements only	Training requirements
No overlapping assignments	Collaboration restriction
Assignment requires specific skill type	Rest time between two consecutive assignments
Fixed assignments	Skill type priorties
	Absence requests
	Counter restrictions on assignments
	Specific series pattern assignment

Smet et al (2012a) divide rostering constraints in two categories: the horizontal constraints and the vertical constraints. A horizontal constraint corresponds to a specific employee. Satisfaction of a horizontal constraint solely depends on the employee's roster and not on other employees. Meanwhile, a vertical constraint is defined according to general characteristics that apply to a group of employees. Satisfaction of a vertical constraint is subject to the shift assignments of the employees in the group. For example, satisfaction of a training requirement constraint depends on the shift assignments of the trainer and the trainee. In Table 1, the first three soft constraints (coverage requirements, training requirements and collaboration restriction) are vertical constraints, while the other soft constraints are horizontal constraints.

Assume the personnel is indexed $i = 1 \dots n$, the individual penalty $P_{H,i}$ is the sum of penalties from the horizontal constraints of employee i. The vertical penalty P_V is the sum of penalties from the vertical constraints. The calculation of P_V and $P_{H,i}$ follows the formulation of Smet et al (2012a). When we only consider the roster quality, the objective of the personnel rostering problem is to minimize the total penalty P^{WS} (defined in Eq. 1) while obeying all hard constraints.

$$\begin{aligned} &\text{minimize } P^{WS}; \\ &\text{with } P^{WS} = P_V + \textstyle\sum_{i=1}^{n} P_{H,i}; \end{aligned} \tag{1}$$

2.2 The fairness objectives

Eq. 1 only optimizes the roster quality but does not pay attention to improving fairness. In the literature, four alternative objectives have been presented to take into account the roster quality as well as the fairness (Smet et al, 2012b; Martin et al, 2013). They are:

1. The vertical and the maximum individual penalty P^{Max} (Eq. 2),

2. The total penalty and the absolute deviation of the individual penalties P^{Dev} (Eq. 3),
3. The total penalty and the range of individual penalties P^{Error} (Eq. 4), and
4. The sum of squared penalties P^{SS} (Eq. 5).

$$P^{Max} = P_V + n. \max_{i\in\{1..n\}} P_{H,i}; \tag{2}$$

$$P^{Dev} = P^{WS} + \sum_{i=1}^{n} |P_{H,i} - \frac{1}{n}\sum_{i=1}^{n} P_{H,i}|; \tag{3}$$

$$P^{Error} = P^{WS} + n.(\max_{i\in\{1..n\}} P_{H,i} - \min_{i\in\{1..n\}} P_{H,i}); \tag{4}$$

$$P^{SS} = \sqrt{(P_V)^2 + \sum_{i=1}^{n}(P_{H,i})^2}; \tag{5}$$

Objective functions 2-5 target improvements both in terms of roster quality and fairness. Minimizing P^{Max} expresses minimization of the worst individual penalty. In this way, the other individual penalties cannot be higher than the worst one. Minimizing P^{Dev} implies improving the roster quality and minimizing the individual penalty differences among employees. On the other hand, P^{Error} improves fairness by decreasing the difference between individual penalties among employees. The quadratic expression in P^{SS} prevents one individual penalty to be too high compared to other.

It should be noted that objective P^{SS} modifies the surface of the solution space of the original objective P^{WS} to a large extent, due to the quadratic operation. This effect is noteworthy since the relative importance of the constraints changes. Martin et al (2013), therefore, introduced weights for P_V and $P_{H,i}$ to reformulate P^{SS}. Many trial-and-error experiments may be needed to find appropriate weights that can restore the importance degrees of the constraints. The surface of the solution space also changes for P^{Max}, P^{Dev} and P^{Error}. However, the effect will not be as large as it is for P^{SS}, since in the former objectives, the weights of the constraints are preserved. That is, P_v and $P_{H,i}$ are included instead of $(P_v)^2$ and $(P_{H,i})^2$.

In order to provide a more accurate representation of the fairness, we introduce a new objective: P^{Lexi}. This objective is similar to the decomposition fairness model for course timetabling problems proposed by Mühlenthaler and Wanka (2013). P^{Lexi} is defined as a permutation of all individual penalties, sorted in a non-increasing order (represented in Eq. 6).

$$P^{Lexi} = (P'_{H,1}, P'_{H,2}, .., P'_{H,n}) \ s.t. \ P'_{H,1} \geq P'_{H,2} \geq ... \geq P'_{H,n} \tag{6}$$

For P^{Lexi}, fairness can then be defined as follows.

Definition 1 Roster p is considered to be more fair than roster q if $P^{Lexi}(p) <^L P^{Lexi}(q)$, i.e. $P^{Lexi}(p)$ is lexicographically smaller than $P^{Lexi}(q)$.

In other words, we say that roster p is more fair than roster q if the first non-zero component of the vector $P^{Lexi}(p) - P^{Lexi}(q)$ is negative.

The objective P^{Lexi} offers several advantages. Similar to the use of P^{Max}, the use of P^{Lexi} minimizes the maximum individual penalty. In fact, the individual penalty of each employee is minimized when P^{Lexi} is used since all individual penalties are represented. This is realized by minimizing the next maximum individual penalty in stages over all employees. The individual penalty for all employee is minimized step by step, starting with the employee having the maximum penalty up to the employee having the minimum penalty. This behavior differs from the behavior of P^{Dev} that tries to minimize the individual penalty differences.

Minimizing P^{Lexi} involves a trade-off mechanism that has an intuitive meaning. A trade-off between two employees is said to be beneficial if it results in a situation that is better for both employees. For example, moving workload from employee i to employee j is beneficial if $max\{P_{H,i}, P_{H,j}\}$ becomes smaller. That is, two employees can help to alleviate of each other's workload as long as the maximum of their individual penalties decreases. In addition, the objective P^{Lexi} does not favor negative trade-off. This is in contrast with P^{Dev} or P^{Error} that can favor to increase an individual penalty without decreasing other individual penalties.

The main disadvantage of P^{Lexi} is that it is only suitable for rostering problems without vertical soft constraints, since P_V is not considered in Eq. 6. If there are vertical soft constraints, P^{Lexi} could be used by adding a new hard constraint to the model which specifies an upper bound on the vertical penalty P_V. A less restrictive approach would be to incorporate P_V in P^{Lexi}, as is shown in Eqs. 7-8. Eq. 7 combines the objectives P^{WS} and P^{Lexi} and prioritizes the former against the latter. Eq. 8 combines the objectives P^{WS} and P^{Lexi} and tries to find a good balance between the roster quality and the fairness.

$$P^{ModLexi1} = (P^{WS}, P'_{H,1}, P'_{H,2}, .., P'_{H,n}) \ s.t. \ P'_{H,1} \geq P'_{H,2} \geq ... \geq P'_{H,n} \tag{7}$$

$$\begin{aligned} P^{ModLexi2} &= (P'_{H,1} + P^{WS}, P'_{H,2} + P^{WS}, .., P'_{H,n} + P^{WS}) \\ &\qquad s.t. \ P'_{H,1} \geq P'_{H,2} \geq ... \geq P'_{H,n} \end{aligned} \tag{8}$$

P^{Lexi} should be applied with care. Objective P^{Lexi} in a rostering problem with vertical hard constraints can produce a higher total penalty than the objective P^{WS}. When optimizing P^{Lexi}, a beneficial trade-off may increase the total penalty P^{WS}.

2.3 Criteria to compare the roster quality and the fairness

The objectives P^{Max}, P^{Dev}, P^{Error}, and P^{SS} integrate the roster quality and the fairness in their respective way. Comparing the qualities of several rosters based on these objectives can produce biased results. Furthermore, the objectives P^{Lexi}, $P^{ModLexi1}$, $P^{ModLexi2}$ can be used to obtain the order of the fairness of two or more rosters. However, it cannot provide quantitative differences in roster quality. Similar to Bard and Purnomo (2005), we suggest to calculate three examination criteria that can be used for comparing the quality of two or more rosters:

1. The total penalty P^{WS}
2. The standard deviation of the individual penalties $\sigma(P_H)$
3. The maximum individual penalty $P_{H,max} = \max\limits_{i\in\{1..n\}} P_{H,i}$

with $P_H = \{P_{H,1}, ..., P_{H,n}\}$.

The first examination criterion corresponds to the roster quality, the second one represents the fairness, and the third relates to the most unattractive individual rosters.

Contrary to Martin et al (2013), we do not recommend Jain's index to compare the quality of rosters based on the following consideration. Let $J(p)$ be Jain's index of roster p, and let the individual roster quality of employee i in roster p be $P_{H,i}(p)$. $J(p)$ can be calculated as shown in Eq. 9. When the difference between individual penalties decreases, Jain's index increases. Therefore, a high value of $J(p)$ indicates high fairness. The maximum value of $J(p)$ is one, which indicates that all employees receive the same individual penalty.

$$J(p) = \frac{(\sum_{i=1}^{n} P_{H,i}(p))^2}{n.\sum_{i=1}^{n}(P_{H,i}(p))^2} \tag{9}$$

Unfortunately, Jain's index can also be improved by increasing the individual penalties while keeping the differences fixed, as stated in Proposition 1. The proposition compares two rosters that have individual penalties differing by the same amount $d > 0$. In this situation, both rosters have the same value of $\sigma(P_H)$. However, the roster with the higher total penalty has a better Jain's index. Therefore, a high value of Jain's index may not always refer to high fairness since it may also be the result of an increase of the individual penalties. Jain's index is still suitable when the total horizontal penalty is fixed, or when the objective is to be maximized.

Proposition 1 *Suppose there are two rosters p and q. If $J(p) < 1$ and $P_{H,i}(q) = P_{H,i}(p) + d$, $\forall i \in \{1, .., n\}, d > 0$, then $J(q) > J(p)$.*

Proof Substituting $P_{H,i}(q) = P_{H,i}(p) + d$ into Eq. 9, results in

$$J(q) = \frac{(\sum_{i=1}^{n} P_{H,i}(p))^2 + M}{n.\sum_{i=1}^{n}(P_{H,i}(p))^2 + M}$$

with $M = 2nd\sum_{i=1}^{n} P_{H,i}(p) + n^2d^2$. Since $d > 0$ and $J(p) < 1$, then $J(q) > J(p)$. □

3 The algorithm

We implemented a tabu search algorithm for solving the rostering problems. Tabu search is a well known local search heuristic that has a mechanism to prevent the search from returning to already visited solutions. Algorithm 1 outlines the procedure. The algorithm starts from a random feasible solution, which is improved by exploring neighborhoods through local search moves. The local search moves used for the current personnel rostering problem are Bilgin et al (2012):

– Make or delete an assignment of an employee at a specific day

– Change a shift assignment of an employee at a specific day
– Change a skill assignment of an employee at a specific day
– Swap shift assignments of two employees at a specific day

The algorithm accepts the best neighboring solution that improves upon the best solution obtained so far. If there is no such solution, it accepts the best neighboring solution that results from a non-tabu move. A local search move is considered non-tabu if it does not match with any element in the tabu list. The tabu list keeps record of the characteristics of the recently executed moves. Specifically, it keeps track of the task assignments that have been changed by performing a move. In each record, four variables are maintained: the day index, the employee index, the skill assignment, and the shift assignment. In case a local search move involves two employees, two records are saved in the tabu list.

A greedy local search is applied to every new best solution found in order to intensify the search.

```
Input: Tabu search parameters, initial solution $n^0$;
Best solution := initial solution;
Current solution := initial solution;
while stopping condition not met do
    Generate $k$ neighboring solutions from the current solution;
    if A new best solution found then
        Improve the new best solution through greedy local search;
        Update best solution and current solution;
        Update tabu list;
    end
    else if Non-tabu solution(s) found then
        Update current solution with best non-tabu solution;
        Update tabu list;
    end
end
Output: best solution;
```

Algorithm 1: The tabu search algorithm

The local search moves from Bilgin et al (2012) may not be suitable for minimizing P^{Max}. The value of the objective P^{Max} can be decreased by decreasing P_V and/or $P_{H,max}$. In this way, local search moves that do not decrease P_V or $P_{H,max}$ are not useful in the search. Meanwhile, improving individual rosters that do not correspond to $P_{H,max}$ can be beneficial at later iterations when these individual rosters become the worst ones.

For minimizing P^{Dev}, the local search moves from Bilgin et al (2012) are also not always effective. Consider a situation where $P_{H,i} = P_{H,j}, \forall i, j \in \{1..n\}$. In this situation, the roster can be improved by decreasing P_V. Decreasing one or two individual penalties (assume that $n > 4$) can only make P^{Dev} increase since the absolute deviation increases more than the gained improvement. This situation can indicate the search is trapped in a local optimum. In other words, a local search move that makes an individual penalty to deviate largely from the average is usually not beneficial for improving P^{Dev}. Similarly, a heuristic minimizing P^{Error} can face the same issue.

Considering that the local search moves may not be effective to minimize P^{Max}, P^{Dev} and P^{Error}, we propose a two-phase approach. The first phase optimizes objective P^{WS} while the second phase uses one of the fairness objectives. The first phase is intended to obtain a roster with few constraint violations. The second phase aims to improve the fairness. In order to show the advantage of a two-phase heuristic, we tested the different algorithmic configurations shown in Table 2.

Table 2 Overview of algorithms

Heuristic method	Type	Objective function
MinWS (base method)	one-phase	P^{WS}
OP-MinMax	one-phase	P^{Max}
OP-MinDev	one-phase	P^{Dev}
OP-MinError	one-phase	P^{Error}
OP-MinSS	one-phase	P^{SS}
OP-MinModLexi1	one-phase	$P^{ModLexi1}$
OP-MinModLexi2	one-phase	$P^{ModLexi2}$
TP-MinMax	two-phase	first phase P^{WS}, second phase P^{Max}
TP-MinDev	two-phase	first phase P^{WS}, second phase P^{Dev}
TP-MinError	two-phase	first phase P^{WS}, second phase P^{Error}
TP-MinSS	two-phase	first phase P^{WS}, second phase P^{SS}
TP-MinModLexi1	two-phase	first phase P^{WS}, second phase $P^{ModLexi1}$
TP-MinModLexi2	two-phase	first phase P^{WS}, second phase $P^{ModLexi2}$

4 Experimental results

4.1 Experimental setup

The algorithms are evaluated using instances based on data from six wards (Smet et al, 2012a). Table 3 provides an overview of the instance characteristics. Note that we simplified the problem by omitting the continuity constraints that consider assignments before and after the rostering period.

Table 3 Instance characteristics of Smet et al (2012a)

No	Ward	Abbr.	No of skill types	No of shift types	No of employees	No of days
1	Emergency	Em	4	27	27	28
2	Geriatrics	Gr	2	9	21	28
3	Meal Preparations	MP	2	9	32	31
4	Psychiatry	Ps	3	14	19	31
5	Reception	Re	4	19	19	42
6	Palliative Care	PC	4	23	27	91

The tabu search algorithm was parametrized with a tabu list length of 1000 and a maximum neighborhood size of 1000. If a new best solution is found, 1000 greedy local search iterations are executed. A time limit of 600 seconds is imposed. It should be noted that the computation time for each phase in the two-phase heuristic, is half of the maximum computation time. For each problem instance, five repeated runs were executed. The algorithm was coded in C++ and the experiments were performed on a PC with a 2.7 GHz Intel processor operating on Windows 7.

In the following sections, two experiments are analyzed. Section 4.2 discusses the results for the base experiment, while Section 4.3 presents results for the slack experiment.

4.2 Base experiment

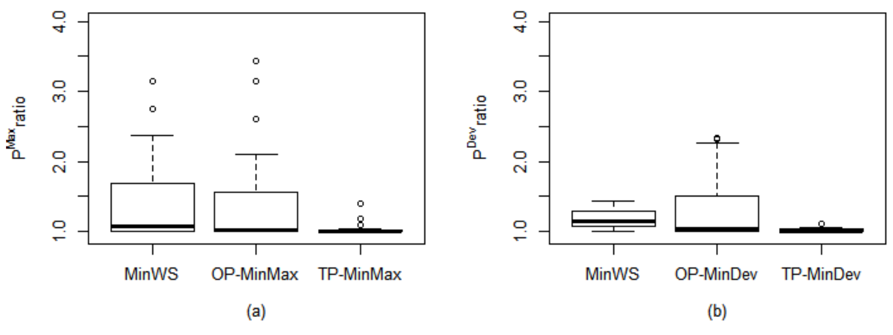
The first experiment evaluates the effectiveness of the two-phase approach and the one-phase approach. The instances are solved using one of the heuristic methods from Table 2. First, we compare the resulting objective values obtained with the two approaches. Then, the resulting rosters are compared in terms of the three examination criteria discussed in Section 2.3.

The objective values obtained with the one-phase approach (OP) and two-phase approach (TP) are shown in Figures 1-2. The horizontal axis corresponds to different heuristic methods. The objective values are obtained by evaluating the final rosters produced by each heuristic method according to the respective fairness objective. The vertical axis in Figure 1 is a ratio obtained by dividing the objective value by the minimum one for each problem instance. This allows to collect several problem instances that have different magnitudes of the weight values into one figure. In Figure 2, the vertical axis represents an index that is obtained as follows. For each problem instance, the vectors P^{Lexi} of the rosters are sorted using $<^L$ (lexicographic sorting). Then, the index represents the position of a roster in the sorted list.

When comparing MinWS with the heuristic methods with a fairness objective, the latter generally perform better. Figure 1 shows that the medians of the ratios obtained with MinWS are always higher than those obtained with the heuristic fairness methods. Figure 2 shows that the median of the index obtained using TP-MinModLexi1 is slightly higher than the one obtained with MinWS. Both methods try to optimize the objective P^{WS}, which is the first criterion when lexicographically sorting the results. TP-MinModLexi1 has the advantage that the range of the results is significantly smaller than the range obtained with MinWS.

The results show that the two-phase heuristics always perform better than the one-phase heuristics, when optimizing P^{Max}, P^{Dev}, P^{Error}, $P^{ModLexi1}$ and $P^{ModLexi2}$. There is no difference between the one-phase and two-phase heuristics when applying P^{SS}.

Now, we compare the performance of the heuristics from Table 2 using the three examination criteria discussed in Section 2.3. Computational results are presented in Tables 4-6. The values of P^{WS}, $P_{H,max}$ and $\sigma(P_H)$ are the averages over five runs. The last two rows of each table summarize the performance of each heuristic. *# with 5% gap* denotes the number of instances for which the heuristic obtained results within 5% from the best result. Results that differ no more than 5% from the best of all heuristics are indicated in bold. The *average ratio* represents the average difference between the best obtained P^{WS}, $P_{H,max}$, $\sigma(P_H)$ and the results obtained with heuristics with a fairness objective.



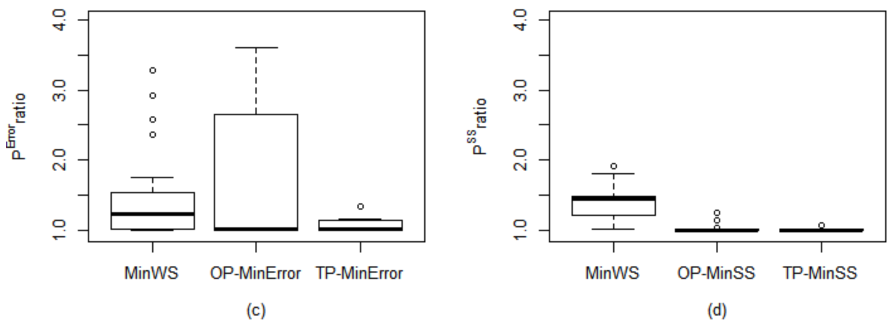


Fig. 1 Comparison of heuristics (a) MinWS, OP-MinMax and TP-MinMax, (b) MinWS, OP-MinDev and TP-MinDev, (c) MinWS, OP-MinError and TP-MinError, (d) MinWS, OP-MinSS and TP-MinSS

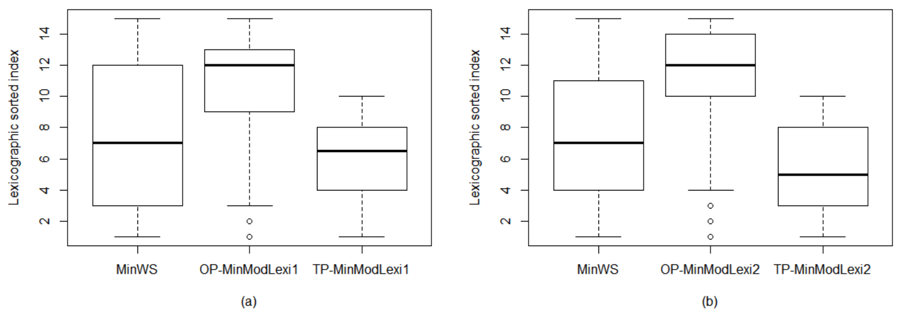


Fig. 2 Comparison of heuristics (a) MinWS, OP-MinModLexi1 and TP-MinModLexi1, (b) MinWS, OP-MinModLexi2 and TP-MinModLexi2

With respect to P^{WS}, the heuristics MinWS, TP-MinDev, TP-MinError, TP-MinModLexi1 and TP-MinModLexi2 all produce good results; i.e. four out of six instances within 5% of the best. This is to be expected, since their objectives all contain P^{WS}. P^{Max} only considers the total vertical penalty and the worst individual penalty. Improving P^{Max} can be achieved by moving (if possible) the vertical penalty to the non-worst individual penalty. This explains the results of TP-MinMax, which deviate largely from the results of MinWS. The objective used in TP-MinSS is different from P^{WS} because of the quadratic operation. Regarding $P_{H,max}$ and $\sigma(P_H)$, most two-phase heuristics with fairness objectives achieve better results than MinWS.

In general, the one-phase heuristics with fairness objectives perform worse than MinWS for all three examination criteria. In Section 3, we argued that the use of fairness objectives can result in ineffective search. P^{Max} and P^{Error} guide the search mainly based on the local search moves that reduce the maximum individual penalty. P^{Dev} has the disadvantage that it does not accept an improvement of one or two individual penalties that largely deviate from the average of all individual penalties.

Except for TP-MinSS, the two-phase heuristic generally performs better than the one-phase heuristic on the three examination criteria, as can be noted by their lower average ratio values. Among the two-phase methods, TP-MinSS produces the worst results for all three examination criteria, because, as mentioned in Section 2.2, the objective surface of the solution space can be quite different.

Table 4 Comparison of P^{WS} results obtained by different heuristics

Instance	MinWS	MinMax Heu.		MinDev Heu.		MinError Heu.		MinSS Heu.		MinModLexi1 Heu.		MinModLexi2 Heu.	
		OP	TP	OP	TP	OP	TP	OP	TP	OP	TP	OP	TP
Em	11,683	306%	112%	130%	**99%**	112%	**101%**	220%	229%	151%	**100%**	138%	**100%**
Gr	13,051	145%	146%	119%	115%	128%	124%	111%	108%	107%	**102%**	110%	**105%**
MP	22,724	163%	171%	**100%**	**100%**	**102%**	**102%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
Ps	9,432	239%	101%	177%	97%	168%	142%	202%	204%	**91%**	**91%**	109%	**94%**
Re	28,238	156%	132%	108%	**100%**	**100%**	**100%**	110%	**102%**	108%	**101%**	108%	**101%**
PC	147,710	243%	123%	**100%**	**100%**	**100%**	**100%**	**101%**	**102%**	107%	**100%**	**104%**	**100%**
# with 5% gap	6	0	0	2	4	3	4	2	3	2	6	2	6
Average ratio	100%	209%	131%	122%	**102%**	118%	111%	141%	141%	111%	**99%**	112%	**100%**

Table 5 Comparison of $P_{H,max}$ results obtained by different heuristics

Instance	MinWS	MinMax Heu.		MinDev Heu.		MinError Heu.		MinSS Heu.		MinModLexi1 Heu.		MinModLexi2 Heu.	
		OP	TP	OP	TP	OP	TP	OP	TP	OP	TP	OP	TP
Em	326	414%	**68%**	247%	**68%**	363%	**71%**	362%	370%	255%	**69%**	222%	79%
Gr	2,429	31%	32%	47%	48%	34%	**26%**	61%	55%	85%	86%	55%	61%
MP	3,827	**57%**	**57%**	**58%**	**58%**	**57%**	**57%**	**58%**	**58%**	73%	66%	**58%**	**58%**
Ps	400	260%	**100%**	370%	**100%**	410%	**100%**	285%	281%	**100%**	**100%**	282%	**100%**
Re	1,520	**104%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
PC	24,320	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
# with 5% gap	6	3	5	3	5	3	6	3	3	3	4	3	4
Average ratio	100%	161%	76%	154%	79%	177%	76%	161%	161%	119%	87%	136%	83%

Table 6 Comparison of $\sigma(P_H)$ results obtained by different heuristics

Instance	MinWS	MinMax Heu.		MinDev Heu.		MinError Heu.		MinSS Heu.		MinModLexi1 Heu.		MinModLexi2 Heu.	
		OP	TP	OP	TP	OP	TP	OP	TP	OP	TP	OP	TP
Em	72	448%	83%	264%	**74%**	329%	90%	296%	321%	336%	83%	266%	83%
Gr	720	36%	37%	34%	35%	42%	**31%**	51%	51%	79%	76%	59%	61%
MP	825	69%	68%	**38%**	**38%**	65%	68%	**38%**	**38%**	51%	45%	**38%**	**38%**
Ps	140	149%	105%	269%	99%	217%	**54%**	142%	150%	92%	92%	217%	98%
Re	496	**92%**	102%	98%	100%	100%	100%	103%	109%	97%	100%	**95%**	99%
PC	5,283	127%	107%	**83%**	**83%**	102%	102%	**82%**	**82%**	88%	88%	88%	88%
# with 5% gap	6	1	0	2	3	0	2	2	2	0	0	2	1
Average ratio	100%	153%	84%	131%	72%	143%	74%	119%	125%	124%	81%	127%	78%

Overall, TP-MinDev and TP-MinError are shown to perform best as they can balance P^{WS}, $P_{H,max}$ and $\sigma(P_H)$. TP-MinDev results in an increase of 2 percentage points over MinWS for P^{WS}, but it manages to reduce $P_{H,max}$ and $\sigma(P_H)$ with 21 and 28 percentage points, respectively. TP-MinError results in an increase of 11 percentage points over MinWS for P^{WS}, while $P_{H,max}$ and $\sigma(P_H)$ improve with 24 and 26 percentage points, respectively. However, it should be noted that for some instances, the results of TP-MinDev and TP-MinError differ largely from the results of MinWS. TP-MinDev produces a P^{WS} value 15 percentage points higher than MinWS for the Geriatrics ward, and TP-MinError produces P^{WS} 42 percentage points higher than MinWS for the Psychiatry ward.

TP-MinModLexi1 and TP-MinModLexi2 present a more balanced result, i.e. these methods are capable of producing both rosters with few violations and high fairness. For P^{WS}, the values are comparable to those produced by MinWS, however, for $P_{H,max}$ and $\sigma(P_H)$, the lexicographic heuristics produce significantly lower values. TP-MinModLexi1 improves 13 and 19 percentage points on $P_{H,max}$ and $\sigma(P_H)$, while TP-MinModLexi2 improves 17 and 22 percentage points on $P_{H,max}$ and $\sigma(P_H)$.

4.3 Slack experiment

The second experiment compares the fairness resulting from the two-phase heuristic methods with different objectives. First, the algorithms are modified such that they behave similar to goal programming, i.e. the value of the objective P^{WS} obtained in the first phase is added as a hard constraint to the model for the second phase. This new constraint ensures that new solutions in the second phase are only accepted if their total penalty P^{WS} is less than or equal to the value obtained in the first phase, allowing some level of slack β. Three values for β are considered: 0%, 5%, and 10%. In the second phase, one of the following objectives is optimized: P^{Max}, P^{Dev}, P^{Error}, P^{SS} or P^{Lexi}. Algorithm 1 is still used in both phases. Note that in the experiments, the first phase is only run once (for each problem instance, each replication) such that the second phase always starts from the same initial solution.

Table 7 shows that the final total penalty P^{WS} for the two-phase heuristic is indeed always within $(100 + \beta)$% of MinWS. TP-MinDev and TP-Minerror seemingly do not make much use of the allowed slack as their P^{WS} values are close to 100%. TP-MinLexi on the other hand, takes much more advantage of the allowed slack.

In general, the results show that the additional slack can help the heuristics to improve fairness. An increase of P^{WS} is usually accompanied by a decrease of $P_{H,max}$ and $\sigma(P_H)$. This behavior is true for all the two-phase heuristics except for the heuristics with objective P^{SS}.

Figures 3-5 show the heuristics ordered by fairness. Similar with Figure 2, the order index in Figures 3-5 is obtained by lexicographically sorting ($<^L$) the vectors P^{Lexi}. The results show that TP-MinLexi generally produces vectors P^{Lexi} that are lexicographically smaller than the ones produced by other heuristics. This shows that TP-MinLexi minimizes the penalty of each employee by allowing a positive trade-off among them, which is a significant advantage and makes this heuristic suitable for practical applications.

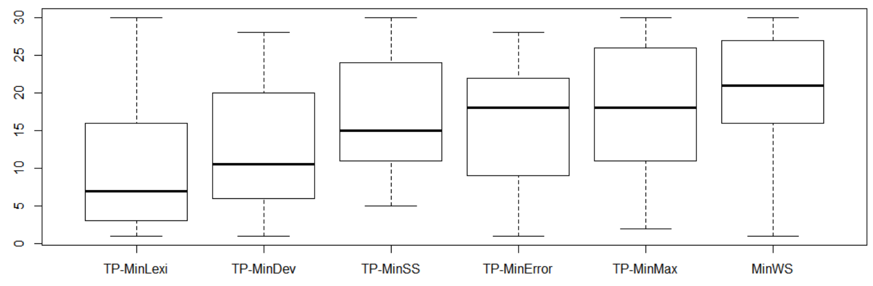


Fig. 3 Comparison of two-phase heuristics with 0% slack

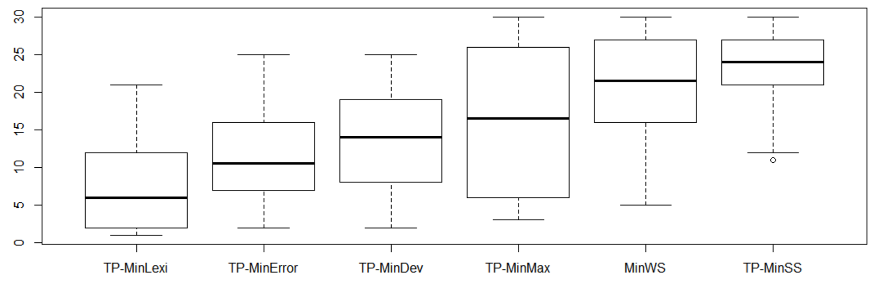


Fig. 4 Comparison of two-phase heuristics with 5% slack

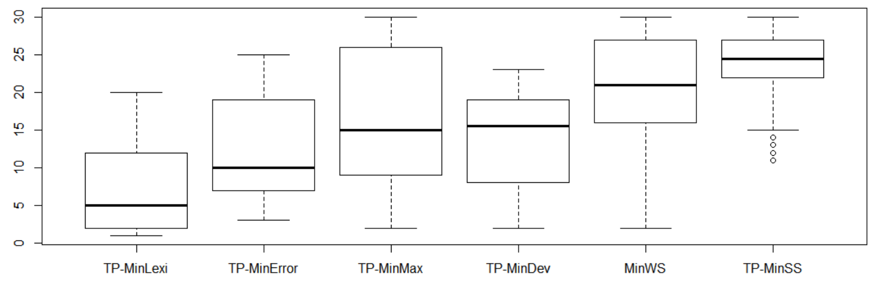


Fig. 5 Comparison of two-phase heuristics with 10% slack

Table 7 Comparison of P^{WS}, $P_{H,max}$ and $\sigma(P_H)$ results obtained by two-phase heuristics and the slack approach

Index	Slack	MinWS	TP-MinMax	TP-MinDev	TP-MinError	TP-MinSS	TP-MinLexi
P^{WS}	0%	100.0%	99.9%	99.0%	99.3%	99.6%	99.7%
	5%	100.0%	102.8%	99.3%	100.2%	102.4%	104.4%
	10%	100.0%	105.4%	100.9%	100.9%	103.5%	108.6%
$P_{H,max}$	0%	100.0%	87.5%	87.3%	85.0%	98.6%	85.1%
	5%	100.0%	83.5%	86.2%	81.0%	118.7%	83.5%
	10%	100.0%	78.4%	82.8%	77.5%	121.8%	76.3%
$\sigma(P_H)$	0%	100.0%	92.8%	79.4%	89.3%	88.4%	80.2%
	5%	100.0%	91.7%	75.9%	86.5%	107.6%	77.5%
	10%	100.0%	88.5%	74.2%	86.0%	117.6%	74.8%

5 Conclusion and future research

The present paper introduced methodologies to improve fairness in personnel rostering. First, a new lexicographic objective was described. Second, a two-phase heuristic approach, which makes use of the lexicographic evaluation, was presented. Finally, an extension of the two-phase approach was introduced which allows for some slack on the total penalty in order to enable fairness improvements in the second phase.

Computational experiments have been analyzed to identify the effectiveness of the new contributions. Three examination criteria from the academic literature were used to asses the roster quality and the fairness. The computational results showed that fair rosters can be produced, without significantly decreasing the roster quality.

This research can be extended in several ways. We argued that the existing local search moves may not be effective for optimizing the objectives representing fairness. A chain of local search moves as described by Burke et al (2013), may be beneficial. An algorithm that directly optimizes the three examination criteria in an aggregated manner can be investigated. Possible directions may be to consider a multi-objective approach or several iterative phases.

References

Bard JF, Purnomo HW (2005) Preference scheduling for nurses using column generation. European Journal of Operational Research 164(2):510–534

Bilgin B, De Causmaecker P, Rossie B, Vanden Berghe G (2012) Local search neighbourhoods for dealing with a novel nurse rostering model. Annals of Operations Research 194(1):33–57

Burke EK, Curtois T, Qu R, Vanden Berghe G (2013) A time predefined variable depth search for nurse rostering. INFORMS Journal on Computing 25(3):411–419

Chiaramonte MV, Chiaramonte LM (2008) An agent-based nurse rostering system under minimal staffing conditions. International Journal of Production Economics 114(2):697–713

Ernst AT, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1):3–27

Jain R, Chiu D, Hawe W (1984) A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Tech. Rep. DEC-TR-301, Digital Equipment Corporation

Komarudin, Guerry M, De Feyter T, Vanden Berghe G (2013) The roster quality staffing problem - a methodology for improving the roster quality by modifying the personnel structure. European Journal Of Operational Research 230:551–562

Larrabee JH, Janney MA, Ostrow CL, Withrow ML, Hobbs Jr GR, Burant C (2003) Predicting registered nurse job satisfaction and intent to leave. Journal of Nursing Administration 33(5):271–283

Martin S, Ouelhadj D, Smet P, Vanden Berghe G, Özcan E (2013) Cooperative search for fair nurse rosters. Expert Systems with Applications 40:6674–6683

Mühlenthaler M, Wanka R (2013) A decomposition of the max-min fair curriculum-based course timetabling problem : The impact of solving subproblems to optimality. In: Multidisciplinary International Sceduling Conference (MISTA) 2013

Smet P, Bilgin B, De Causmaecker P, Vanden Berghe G (2012a) Modelling and evaluation issues in nurse rostering. Annals of Operations Research pp 1–24

Smet P, Martin S, Ouelhadj D, Özcan E, Vanden Berghe G (2012b) Investigation of fairness measures for nurse rostering. In: the International Conference on the Practice and Theory of Timetabling (PATAT 2012)

Smet P, De Causmaecker P, Bilgin B, Vanden Berghe G (2013) Nurse Rostering: A Complex Example of Personnel Scheduling with Perspectives. Studies in Computational Intelligence 505:129–153

Stolletz R, Brunner JO (2012) Fair optimization of fortnightly physician schedules with flexible shifts. European Journal of Operational Research 219(3):622–629

Scheduling the Australian Football League Using the PEAST Algorithm

Jari Kyngäs[1], Kimmo Nurmi[1], Nico Kyngäs[1],George Lilley[2], Thea Salter[3]

[1] Satakunta University of Applied Sciences, Tiedepuisto 3, 28600 Pori, Finland

[2] Box Hill Senior Secondary College, 19 Dunloe Avenue, Mont Albert North 3129, Australia

[3] AFL HOUSE, 140 Harbour Esplanade, Docklands VIC 3008, Melbourne VIC 3001, Australia

Abstract**.** Generating a schedule for a professional sports league is an extremely demanding task. Good schedules have many benefits for the league, such as higher incomes, lower costs and more interesting and fairer seasons. This paper presents the 3-phase process needed to schedule the Australian Football League. The building of the schedule is very challenging and often requires computational intelligence to generate an acceptable schedule. There are a multitude of stakeholders with varying requests (and often requests vary significantly year on year). We used the PEAST (Population, Ejection, Annealing, Shuffling, Tabu) algorithm to schedule the 2013 season. The comparison showed that there are alternative solutions available that are comparable to the current scheduling outcome.

Keywords: Sports scheduling, Real-world scheduling, PEAST algorithm.

1 Introduction

Professional sports leagues have become big businesses. At the same time, the quality of the schedules has become increasingly important; the schedule has a direct impact on revenue for all involved parties. For instance, the number of spectators in the stadiums and the traveling costs for the teams are influenced by the schedule, and TV networks that pay for broadcasting rights want the most attractive games to be scheduled at commercially interesting times. Furthermore, a good schedule makes a season more interesting for the media and the fans, and fairer for the teams. Nurmi et al. (2010) report a growing number of cases in which academic researchers have been able to close a scheduling contract with a professional sports league owner. Excellent overviews of sports scheduling can be found in Easton et al. (2004) and Rasmussen, Trick (2008). An extensive bibliography can be found in Knust (2012) and an annotated bibliography in Kendall et al. (2010).

In a sports tournament, n teams play against each other over a period of time according to a given timetable. The teams belong to a *league*, which organizes *games* or *matches* between the teams. Each game consists of an ordered pair of teams, denoted *(i, j)* or *i-j*, where team *i* plays *at home* - that is, uses its own *venue* (stadium) for a game - and team *j* plays *away*. Games are scheduled in *rounds*, which are played on given *days*. A *schedule* consists of games assigned to rounds. A schedule is *compact* if it uses the minimum number of rounds required to schedule all the games; otherwise it is *relaxed*. If a team plays two home or two away games in two consecutive rounds, it is said to have a *break*. In general, for reasons of fairness, breaks are to be avoided. However, a team can prefer to have two or more consecutive away games if its stadium is located far from the opponent's venues, and the venues of these opponents are close to each other. A series of consecutive away games is called an *away tour*.

In a *round robin tournament* each team plays against every other team a fixed number of times. Most sports leagues play a double round robin tournament, where the teams meet once at home and once away. A *mirrored* double round robin tournament is a tournament where every team plays against every other team once in the first $n - 1$ rounds, followed by the same games with reversed venues in the last $n - 1$ rounds.

Sports scheduling involves three main problems. First, the easiest problem to solve is to find a schedule with the minimum number of breaks. De Werra (1981) has presented an efficient algorithm to compute a minimum break schedule for a single round robin tournament. If n is even, it is always possible to construct a schedule with $n - 2$ breaks.

Second, the problem of finding a schedule that *minimizes the travel distances* is called the Traveling Tournament Problem (TTP) (Easton et al., 2001). In TTP, the teams do not return home after each away game but instead travel from one away game to the next. However, excessively long away trips as well as home stands should be avoided.

Third, most professional sports leagues introduce many additional requirements in addition to minimizing breaks and travel distances. We call the problem of finding a schedule that *satisfies given constraints* (Nurmi et al., 2010) the Constrained Sports Scheduling Problem (CSSP). The goal is to find a feasible solution that is the most acceptable for the sports league owner - that is, a solution that has no hard constraint violations and that minimizes the weighted sum of the soft constraint violations. Scheduling the Australian Football League is an example of a CSSP.

Australian Rules football (officially Australian football) was invented in Melbourne, Australia. It has been played since 1858, when the first match between Melbourne Grammar School and Scotch College took place (Blainey, 2010). Originally it was invented to keep the cricketers fit during the winter time. The game has been played in some kind of league format since 1877. Some sources claim that the early history of Australian Rules football is more or less obscure, but the modern-day rules are well known (Sydney University, 2014)(OnlyMelbourne, 2014).

The game is most popular in Australia but is played practically all over the world. In all southern states of Australia it is the most popular of all sports. When measured by attendance, it is by far the most popular sport in Australia. The spectator average per match for the season 2013 was 33,500. The most popular matches in the regular season have more than 80,000 spectators.

Australian Rules football is a very physical game. What makes it different from other physical sports is the fact that the use of padding is not mandatory. Some players (ruckmen) wear shin and thigh pads, but in general, pads are rarely worn. This causes quite a lot of small injuries to thighs, hamstrings and calf muscles. The relatively high injury rates in the sport are taken into consideration by playing only once a week. There must be a minimum six-day break between the matches. This gives the players a chance to recover from minor injuries.

Section 2 presents the 3-phase process needed to schedule the Australian Football League. The section introduces the requirements, requests and other constraints the format implies. In Section 3 we describe the PEAST algorithm that has been used to schedule professional sports leagues. Section 4 reports the computational results for the 2013 season.

2 The Schedule

The Australian Football League (hereafter, AFL) has 18 teams: Adelaide Crows and Port Adelaide (Southern Australia), Brisbane Lions and Gold Coast Suns (Queensland), Fremantle and West Coast Eagles (Western Australia), Greater Western Sydney Giants and Sydney Swans (Sydney, New South Wales) and Carlton, Collingwood, Essendon, Geelong Cats, Hawthorn, Melbourne, North Melbourne, Richmond, St Kilda and Western Bulldogs (Victoria region).

AFL is trying to expand the game throughout the country and even to New Zealand. Therefore, some of the matches are played in cities and stadiums that do not have a permanent home team. Such cities are Darwin (Northern Australia), Hobart and Launceston (Tasmania), Cairns (Northern Queensland) and Wellington (New Zealand).

Australia is a big country, which causes extensive travel loads for the teams, especially for the teams from Queensland and Western Australia. The Victoria teams travel the least, about 12,000-20,000 kilometers each season, while the teams from Western Australia have to travel about 70,000 kilometers per season. Unfortunately, not much can be done to reduce the total travelling distance since no away tours (two or more consecutive away games) can be arranged due to a regular match being scheduled in each city per week.

The schedule for AFL in 2013 had a complicated structure. It consisted of each team playing against every other team once - i.e., a single round robin. In addition, each team had to play 5 additional matches. This adds up to 22 matches, 11 home and 11 away matches, for each team. The combination of a single round robin and the additional matches makes the schedule different from most of the other professional sports league schedules. The schedule consists of 23 rounds, and each team has one bye during rounds 11-13. There are 20 rounds of 9 matches and 3 rounds of 6 matches.

Building the schedule is a process comprising three phases. First, the host teams of the single round robin tournament and the five additional matches are decided. The second phase includes building the actual schedule. The schedule is built based on rounds – not the actual match days. In most cases the matches of a round are played on Friday, Saturday and Sunday. In the last phase the exact weekdays and venues of the matches are decided. These decisions are mostly constrained by various broadcasting and venue requirements.

2.1 Phase 1: Deciding the Host Teams

In the first phase the host teams of the single round robin tournament and the five additional matches are decided. The league authorities defined 12 selection rules, which we converted to constraints. Ten of the rules were strict and two had some flexibility in them. In the sports scheduling literature, a strict rule is defined as a hard constraint and a flexible rule is defined as a soft constraint.

The basic selection of the matches in the single round robin is subject to the following constraints: (H denotes a hard constraint and S denotes a soft constraint)

H1. All teams have to play a minimum of 5 matches in Victoria.
H2. Victorian teams should travel outside Victoria a minimum of 5 times.
H3. Each team must have at least one home match against Collingwood or Essendon.
H4. Each team has to travel to Western Australia at least once. If a team travels to Western Australia twice, there must be at least six rounds between the visits (this is a constraint of phase two).
H5. All clubs have to play at least one match at the MCG stadium.

The selection for the remaining 5 matches for each team is as follows:

H6. "Blockbuster" matches must be included. These are the matches between the big six teams from Victoria – Carlton, Collingwood, Essendon, Geelong Cats, Hawthorn and Richmond. These are fixed by the league authorities (there are some exceptions, for instance Collingwood and Richmond did not play against each other twice in 2013).
H7. Matches between local rivals (Adelaide Crows and Port Adelaide, Brisbane Lions and Gold Coast Suns, Fremantle and West Coast Eagles, Greater Western Sydney Giants and Sydney Swans) should be respected.
H8. The top four teams from 2012 can have only one meeting with the bottom four teams from 2012, with the exception of Sydney rivals.
S1. The top eight teams should play the top eight teams twice, a minimum of three times.
S2. The bottom ten teams should play the bottom ten teams twice, a minimum of three times.
H9. The bottom two teams from 2012 should not meet the top eight teams from 2012 twice (Sydney rivals are an exception).
H10. No team that travelled in round 23 in 2012 is to travel in round 23 this year.

According to the league authorities, the travel load is the second most important thing to consider. Therefore, we added two restrictions (as hard constraints):

H11. All teams should travel 2-3 times to either Western Australia or Queensland. This constraint somewhat equalizes the travel load between the teams (of course, only those teams not from Western Australia or Queensland). Without this restriction, some teams might visit Western Australia once and Queensland not at all, while some other teams might make a maximum number of visits to these areas (two visits to both areas).
H12. The local rivals' travel loads should be as equal in length as possible. We recognized that in previous years' schedules there were big differences in the local rivals' travel loads.

2.2 Phase 2: Building the Schedule

The second phase consists of actually building the schedule. A basic round consists of 9 matches, of which 1 is played on Friday, 5 are played on Saturday and 3 are played on Sunday. In rounds 11-13, each team has one bye (i.e. there are 6 matches in these rounds) and the distribution of matches is 1 match on Friday, 3 matches on Saturday and 2 matches on Sunday. Moreover, in rounds 1, 7 and 10, one Sunday match is actually played on Monday.

Anzac Day (25 April) is also special because no matter which weekday it happens to be, at least one match is played. The match played is between Essendon and Collingwood. As mentioned earlier, there should be a six-day break between the matches, but Anzac Day is an exception. All the teams playing on Anzac Day must be prepared to play either their preceding or following match with a shorter break. Of course, the schedule should be made in such a way that it places these teams' preceding and following matches as far away from Anzac Day as possible.

There are nine different constraints that must be used to describe the problem framework. Only one of the constraints is a soft constraint. We follow Nurmi et al. (2010) in the specification of the constraints:

C01. There are exactly 23 rounds available for the tournament.
C02. Nine matches can be assigned to rounds 1-10 and 14-23, and 6 matches to rounds 11-13.
C07. There should be at least 0 and at most 1 home matches for four different pairs of teams. The pairs are Adelaide Crows and Port Adelaide, Brisbane Lions and Gold Coast Suns, Fremantle and West Coast Eagles, Greater Western Sydney Giants and Sydney Swans. What this means is that these pairs of teams can never play at home in the same round.
C10. There are 42 pre-assigned matches (this number can vary year on year).
C13. Teams cannot have more than 3 consecutive home matches.
C14. Teams cannot have more than 3 consecutive away matches.
C19. There must be at least 6 rounds between two matches with the same opponents (soft constraint).
C24. If two teams play against each other twice, the second match cannot be played before round 11.
C25. If two teams play against each other only once, this match cannot be played after round 22.

In addition, we had to add 6 constraints that were not included in Nurmi et al. (2010). Three of these constraints are hard constraints. We prefix these constraints with the letter X to separate them from the phase one constraints:

XH1. Geelong cats must play only 4 home matches before round 10 (only valid for 2013).
XH2. There should be a minimum of 6 days break between each match.
XH3. There should be no overlapping broadcasting slots.
XS1. There must be at least 6 rounds between visits to WA / Qld.
XS2. Friday matches should mostly be played in the Etihad or MCG stadiums.
XS3. The total number of breaks should be minimized.
XS4. Venue contractual requirements. There has to be a predefined number of matches played at each stadium.

2.3 Phase 3: Deciding the Weekdays and Venues

In the last phase the exact weekdays and venues of the matches are decided. These decisions are mostly constrained by various broadcasting and venue requirements. All of the matches are broadcast on television. Foxtel is an Australian pay television company that produce and broadcast five matches a week (in a standard round) – three on Saturday and two on Sunday. The Seven Network broadcasts the remaining four matches (in a standard round) on a free-to-air network. It is important that the broadcasts do not overlap on Saturday or Sunday in Western Australia, Southern Australia, Queensland and New South Wales. This is applicable to the Adelaide and Port Adelaide teams, the Fremantle and West Coast Eagles teams, the Brisbane Lions and Gold Coast Suns teams, and also to the Sydney Swans and GWS Giants teams. This is manageable because there are at least 3 days on which to play, and subsequently there are also varying timeslots to schedule within each day so they can be scheduled on the same day in a different timeslot (eg. afternoon and night).

Two stadiums, Etihad and MCG, host almost half of all the matches (93/198). There are 10 teams that play home matches at these stadiums. Two of these teams play the majority of their home matches at Etihad Stadium and the remaining eight play a varying number of home matches at both stadiums. Of the latter, two of these teams have to play a minimum number of away matches at Etihad Stadium. We cannot be specific about the number of matches each team should play in these stadiums due to confidentiality. These stadiums should mostly be used for Friday matches.

At first sight, the two stadiums might seem to cause an optimization problem. However, this is not the case. Because of the local rivals, there are always 4 matches played in stadiums other than Etihad and MCG (local rivals can never play at home in the same round). This leaves a maximum of five matches to be played at Etihad, MCG and/or Simonds Stadium.

3 The Solution Method

This section describes the PEAST algorithm, which was used to solve all of the three phases of the schedule. The usefulness of an algorithm depends on several criteria. The two most important ones are the quality of the generated solutions and the algorithmic power. Other important criteria include flexibility, extensibility and learning capabilities. A successful heuristic most likely uses mixed local search and population-based methods. A local search method is defined by:

1) A neighborhood structure, which is a mechanism to obtain a new set of neighbor solutions by applying a small perturbation to a given solution.
2) A method of moving from one solution to another.
3) The parameters of the method.

The PEAST algorithm obtains a new neighbor solution by applying the GHCM operator. It explores promising areas in the search space by extending the basic hill-climbing step to generate a sequence of moves in one step, leading from one solution to another. The operator is based on ideas similar to the Lin-Kernighan procedures (Lin and Kernighan, 1973) and ejection chains (Glover, 1992). It moves an object, o_1, from its old position, p_1, to a new position, p_2, and then moves another object, o_2, from position p_2 to a new position, p_3, and so on, ending up with a sequence of moves. A detailed discussion of the GHCM operator can be found in Kyngäs et al. (2012).

```
Input  the population size $n$, the iteration limit $t$, the cloning interval $c$,
          the shuffling interval $s$ and the ADAGEN update interval $a$
   Generate a random initial population of schedules $S_i$ for $i = 1, ..., n$
   Set $best\_S = null$ and $iteration = 1$
WHILE $iteration \le t$
      $k = 1$
      WHILE $k \le n$
     (explore promising areas in the search space)
         Apply GHCM to schedule $S_k$ to get a new schedule
         IF $Cost(S_k) < Cost(best\_S)$ THEN Set $best\_S = S_k$
         $k = k + 1$
      END REPEAT
      (avoid staying stuck in the promising search areas too long)
      Update the simulated annealing framework
      IF $iteration \equiv 0$ (mod $c$) THEN
         (favor the best schedule, i.e. use elitism)
         Replace the worst schedule with the best one
      IF $iteration \equiv 0$ (mod $s$) THEN
         (escape from the local optimum)
         Apply shuffling operators
      IF $iteration \equiv 0$ (mod $a$) THEN
         Update the ADAGEN framework
      $iteration = iteration + 1$
   END WHILE
   Output $best\_S$
```

Figure 1: The pseudo-code of the PEAST algorithm.

The algorithm avoids staying stuck (i.e., the objective function value does not improve for some predefined number of generations) in the same areas of the search space using tabu search and the refined simulated annealing method. A tabu list (Glover et al., 1985) is used to prevent reverse order moves in a single application of the GHCM operator. The simulated annealing refinement is used to decide whether or not to commit to a sequence of moves in the GHCM operator. This refinement is different from the standard simulated annealing (Kirkpatrick et al., 1983). It is used in a three-fold manner: 1) when choosing an object to be moved, 2) when choosing the destination of the object, and 3) when the sequence of moves is cut short (a worsening move is made, and it worsens the solution more than the previous worsening move did). A detailed discussion of the tabu search and simulated annealing refinement can be found in Kyngäs et al. (2012).

The pseudo-code of the algorithm is given in Figure 1. The algorithm uses a population of solutions that enables it to explore a wide range of promising areas in the search space. In every *c* iteration, the least fit schedule is replaced with a clone of the fittest individual. This operation is completely irrespective of the globally fittest schedule (*best_S* in Figure 1) found by that time in the search process.

The PEAST algorithm applies a number of shuffling operators to perturb a solution into a potentially worse solution in order to escape from local optima. The operators are called according to a rule. The idea of shuffling is the same as in hyper-heuristics (Burke et al., 2013) but the other way around. Hyper-heuristic is a mechanism that chooses a heuristic from a set of simple heuristics, applies it to the current solution to get a better solution, then chooses another heuristic and applies it, and continues this iterative cycle until the termination criterion is satisfied. We introduce a number of simple heuristics that are used to worsen the current solution instead of improving it. Examples of shuffling operators can be found in Kyngäs et al. (2012).

The PEAST algorithm is used to solve multi-objective optimization problems - i.e., problems where multiple objective functions have to be optimized simultaneously. The objective functions usually compete in such a way that improving one objective function value most likely improves the other objective function values. The PEAST algorithm uses the adaptive genetic penalty method (ADAGEN) (Nurmi, 1998) to solve the multi-objective problems. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. The ADAGEN method assigns dynamic weights to the hard constraints based on the search trajectory and the constant weights assigned to the soft constraints. The soft constraints are assigned fixed weights according to their significance. The significance is given by the problem owner (end-user).

The PEAST algorithm uses random initial solutions. In our extensive test runs we have found no evidence that a sophisticated initial solution improves results. On the contrary, random initial solutions tend to yield superior or at least as good results (Kyngäs et al., 2012).

The PEAST algorithm has been used to solve several types of real-world scheduling problems and is in industrial use. The first version of the algorithm was used to solve school timetabling problems (Nurmi, 1998). The later versions of the algorithm have been used to solve sports scheduling problems (see, e.g., Kyngäs et al., (2014) and Kyngäs and Nurmi (2009)) and workforce scheduling problems (see, e.g., Kyngäs et al. (2012) and Kyngäs et al. (2013)). Furthermore, we have used the algorithm to solve several artificial and benchmark problems, including school timetabling (Post et al., 2012), balanced incomplete block design(Nurmi et al., 2011), single round robin tournaments with balanced home-away assignments and pre-assignments (Nurmi et al., 2011), days-off scheduling (Nurmi and Kyngäs, 2011) and constraint minimum break problems (Nurmi et al., 2010).

The industrial use of the PEAST algorithm, as well our experiences in solving different benchmark problems, can be summarized as follows:

1) The crucial components of the algorithm are random initial solutions, making moves in sequences (the GHCM operator) and using a population of solutions.
2) Either the simulated annealing refinement or shuffling operators should be used. Both should be used in real-world instances to ensure good quality solutions.
3) A tabu list improves the efficiency of the GHCM operator.
4) The initial object in a move sequence should be chosen using tournament selection.

Even though the best parameter values vary depending on the problem and the instance, our extensive test runs over several years have shown that the following values can safely be used in different real-world problems and instances:

- The population size is 10.
- The cloning interval is 500.
- The shuffling interval is 5,000.
- The maximum length of the move sequence in the GHCM operator is 10.
- The size of the tournament selection is 7.

- The length of the tabu list is 10 (equals the length of the move sequence).
- In the simulated annealing framework we stop the cooling at some predefined temperature. Therefore, after a certain number of iterations, *m*, we continue to accept an increase in the cost function with some constant probability. We choose *m* equal to *t* /2, where *t* is the maximum number of iterations and *p* is equal to 0.0001.

We are aware of the fact that we have used many different heuristic methods in the PEAST algorithm. The acronym PEAST stems from the methods used: Population, Ejection, Annealing, Shuffling and Tabu. One might think that the outcome is nothing more than a collection of old ideas. However, to the best of our knowledge, the heart of the algorithm, the GHCM operator, is one of a kind. The same applies to our implementation of the shuffling operators, simulated annealing and penalty method.

We can steadily note that the PEAST algorithm realizes the criteria given at the beginning of the section. Our industrial partners from different lines of business have stated that the algorithm constantly produces good-quality solutions in acceptable time.

4 Computational Results

We compared our schedule to the running schedule for the 2013 season. Table 1 summarizes the comparison. The optimization model was created in cooperation with one of the league authorities. However, the running schedule was not generated using exactly the same model. For example, the topics H11 and H12 may not have been in effect in the model behind the running schedule. We still believe the comparison is fair enough. The comparison shows that in our view our solution is better than the running schedule. Our solution is at least as good as the running schedule on every topic.

The biggest differences are highlighted. The most significant difference is the sum of differences of the total traveling of Non-Victorian teams (H12). We are not sure whether this was optimized in the running schedule, but we think it should have been. Balancing the traveling distance of local teams most certainly increases overall satisfaction. The second most significant difference is that the top 8 teams should play against each other a minimum of three times (S1). We chose to use a more difficult constraint in our model - that is, “exactly three times” - because we think it would be unfair if a team had to meet the top eight teams more than three times. We were able to find such a tightened solution. Note also that our solution has clearly fewer breaks.

5 Conclusions and Future Work

The format of the AFL fixture was something we had not run into before. One of the most interesting features is that it includes only a single round robin with 18 teams. In addition, each team has 5 extra matches, which increases the total number of matches played during the season to 22. The 5 extra matches are selected in a way that balances the fixtures between the teams.

The traveling is a big issue for the teams. The traveling distance is from 12,000 kilometers up to 70,000 kilometers per season per team. These are big numbers considering only 11 away matches are played during the season.

We showed that the PEAST algorithm is capable of finding good solutions to the AFL fixture. It was quite easy to apply the algorithm to handle all the three phases needed to generate the schedule. One thing to do in the future could be to merge these two phases.

Table 1: Comparison between the running schedule and our schedule. The topics are ordered as follows: phase one, phase two and miscellaneous.

Topic	**Running schedule**	**Our schedule**
All clubs to play a minimum of five matches in Victoria (H1)	5-7	6-7
Victoria teams travel maximum of six times (H2)	ok	ok
Minimum one home match against Essendon or Collingwood for each club (H3)	ok	ok
Each team should visit Perth once (H4)	ok	ok
All teams to play at least one match at MCG (H5)	ok	ok
Blockbuster matches (H6)	ok	ok
Local derbies (H7)	ok	ok
Top 4 teams not to play against bottom 4 teams twice (H8)	Western Bulldogs vs. Adelaide	ok
No top 8 team (other than Syd v GWS x 2) to play either Gold Coast Suns or GWS Giants twice (H9)	ok	ok
Round 23 – alternate travel between non-Vic teams (H10)	ok	ok
Each team should visit Queensland once (H11)	no visit for 4 teams	ok
Total visits to Perth and Queensland between 2-3 (H11)	2 teams once 2 teams four times	ok
The sum of differences of the total traveling of non-Victoria teams (H12)	9474	632
Top 8 teams to play against each other twice a minimum of three times (S1)	Once 2 times Once 4 times Once 5 times	ok
Bottom 10 teams to play bottom 10 teams twice a minimum of three times (S2)	ok	ok
Non-fixed (not pre-assigned) matches played outside regular slots (C01)	1 time	0 times
One bye per team in rounds 11-13 (C02)	ok	ok
20 rounds of nine matches and 3 rounds of six matches (C02)	ok	ok
Number of 3-breaks at home (C13)	2	2
Number of 3-breaks away (C14)	3	0
Local rivals never playing home in same round (C07)	ok	ok
Pre-assigned matches (C10)	ok	ok
Must be a minimum of six weeks between playing a team for the first and second time (C19)	ok	ok
No teams to play for the second time until after round 10 (C24)	ok	ok
All teams must play each other once by round 22 (C25)	ok	ok
No home matches for Geelong Cats at Simonds Stadium until Round 10 (XH1).	ok	ok
Back-to-back Perth minimum of 6 rounds gap (XS1)	3 times 5 rounds gap	1 time 5 rounds gap
Back-to-back Queensland minimum of 6 rounds gap (XS1)	1 time 2 rounds gap 1 time 5 rounds gap	2 times 4 rounds gap
Minimum six-day break between each match, with exceptions for Anzac Day (XH2)	ok	ok
Never play in the same or overlapping timeslot, so that all local matches can be broadcast on free-to-air in each market (XH3)	ok	ok
Number of Friday matches at MCG or Etihad (XS2)	15	17
Total number of breaks (XS3)	94	74
Venue contractual requirements (XS4)	ok	ok
Total traveling of Gold Coast and Brisbane Lions (appr. using Google Maps kilometers)	GC 18,347 BL 21,614	GC 18,347 BL 18,347
Total traveling of Adelaide and Port Adelaide	PA 11,176 A 13,126	PA 11,776 A 11,850
Total traveling of West Coast Eagles and Fremantle	WCE 33,601 F 34,801	WCE 34,101 F 34,101
Total traveling of Sydney Swans and GWS	SS 12,243 GWS 15,300	GWS 12,243 SS 12,800
Total traveling of Victoria teams	6,718-10,992	6,711-10,987
Total traveling of non-Victoria teams	160,208	153,558
Total traveling of Victoria teams	82,917	86,607
Minimum of 45 matches in MCG	ok	ok
Minimum of 46 or 48 matches in Etihad	ok	ok
Other contractual matches	ok	ok
No day or twilight matches at TIO Stadium	ok	ok
No Sunday early or Saturday afternoon matches at Patersons Stadium	ok	ok
Key Features 1-17	ok	ok
Number of breaks for one team	8 (four times) 7 (once) 6 (four times)	8 (once) 7 (once)

References

G. Blainey, “A Game of Our Own – The Origins of Australian Football”, Black Inc., 2010, pp. 7.

E,K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa E. Özcan and R. Qu, “Hyper-heuristics: a survey of the state of the art”, in *Journal of the Operational Research Society* 64, 2003, pp. 1695–1724.

K. Easton, G. Nemhauser, and M. Trick “The traveling tournament problem: description and benchmarks” in Proc of the 7th. International Conference on Principles and Practice of Constraint Programming, Paphos, pp. 580–584, 2001.

K. Easton, G. Nemhauser and M. Trick, “Sports scheduling”, in Handbook of Scheduling, edited by Leung, Florida, USA: CRC Press, pp 52.1-52.19, 2004.

F. Glover, C. McMillan and B. Novick, “Interactive Decision Software and Computer Graphics for Architectural and Space Palnning”, *Annals of Operations Research* 5, 1985, pp. 557-573.

F. Glover, “New ejection chain and alternating path methods for traveling salesman problems”, in *Computer Science and Operations Research: New Developments in Their Interfaces*, edited by Sharda, Balci and Zenios, Elsevier, 1992, pp. 449–509.

G. Kendall, S. Knust, C.C. Ribeiro and S. Urrutia, “Scheduling in Sports: An annotated bibliography”, Computers and Operations Research 37, pp. 1-19, 2010.

S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, “Optimization by Simulated Annealing”, *Science* 220, 1983, pp. 671-680.

S. Knust, “Sports Scheduling Bibliography”, [Online], Available: http://www.inf.uos.de/knust/sportssched/sportlit_class/, (Last update 12.05.2014).

J. Kyngäs and K. Nurmi, “Scheduling the Finnish 1st Division Ice Hockey League”, in *Proc of the 22nd Florida Artificial Intelligence Research Society Conference*, Florida, USA, 2009.

N. Kyngäs, D. Goossens, K. Nurmi and J. Kyngäs, “Optimizing the Unlimited Shift Generation Problem”, *Applications of Evolutionary Computation, Lecture Notes in Computer Science,* Vol. 7248, Springer, USA 2012.

N. Kyngäs, K. Nurmi and J. Kyngäs, “The Workforce Optimization Process Using the PEAST algorithm”, *Lecture Notes in Engineering and Computer Science: Proc. of The International MultiConference of Engineers and Computer Scientists*, Hong Kong, 2013.

N. Kyngäs, K. Nurmi and J. Kyngäs, “Scheduling the Highly Constrained Finnish Major Ice Hockey League”, *Lecture Notes in Engineering and Computer Science: Proc. of The International MultiConference of Engineers and Computer Scientists*, Hong Kong, 2014. (accepted)

S. Lin and B.W. Kernighan, “An effective heuristic for the traveling salesman problem”, *Operations Research* 21, 1973, pp. 498–516.

K. Nurmi, “Genetic Algorithms for Timetabling and Traveling Salesman Problems”, Ph.D. dissertation, Dept. of Applied Math., University of Turku, Finland, 1998. Available: http://www.bit.spt.fi/cimmo.nurmi/

K. Nurmi, D. Goossens, T. Bartsch, F. Bonomo, D. Briskorn, G. Duran, J. Kyngäs, J. Marenco, CC. Ribeiro, FCR. Spieksma, S. Urrutia and R. Wolf-Yadlin, “A Framework for Scheduling Professional Sports Leagues”, in Ao, Sio-Iong (ed.): IAENG Transactions on Engineering Technologies Volume 5, Springer, USA, 2010.

K. Nurmi, D. Goossens and J. Kyngäs, “Scheduling a triple round robin tournament for the Finnish national ice hockey league for players under 20”, in *Proc of the IEEE Symposium on Computational Intelligence in Scheduling*, Paris, France, 2011.

K. Nurmi and J. Kyngäs, “Days-off Scheduling for a Bus Transportation Staff”, *International Journal of Innovative Computing and Applications* Vol. 3(1), Inderscience, UK, 2011.

OnlyMelbourne, “History of Australian Football”, [Online], Available: http://www.onlymelbourne.com.au/melbourne_details.php?id=1543#.Ur6LMrRbHnY (Accessed 20.1.2014).

G. Post, J.H. Kingston, S. Ahmadi, S. Daskalaki, C. Gogos, J. Kyngas, K. Nurmi, N. Musliu, N. Pillay, H. Santos and A. Schaerf, “XHSTT: an XML archive for high school timetabling problems in different countries”, *Annals of Operations Research*, Springer, USA, 2012.

P. Rasmussen and M. Trick, “Round robin scheduling - A survey”, European Journal of Operational Research 188, pp. 617-636, 2008.

Sydney University, Australian National Football Club, “The Origin of Australian Rules”, [Online], Available: http://www.suanfc.com/suanfc/recent-history/the-origin-of-australian-rules (Accessed 20.1.2014)

D. de Werra, “Scheduling in sports”, in Studies on graphs and discrete programming, edited by Amsterdam and Hansen, pp. 381-395, 1981.

Diversity-Oriented Bi-Objective Hyper-heuristics for Patrol Scheduling

Mustafa Mısır · Hoong Chuin Lau

Abstract The patrol scheduling problem is concerned with assigning security teams to different stations for distinct time intervals while respecting a limited number of contractual constraints. The objective is to minimise the total *distance* travelled while maximising the *coverage* of the stations with respect to their security requirement levels. This paper introduces a hyper-heuristic strategy focusing on generating diverse solutions for a bi-objective patrol scheduling problem. While a variety of hyper-heuristics have been applied to a large suite of problem domains usually in the form of single-objective optimisation, we suggest an alternative approach for solving the patrol scheduling problem with two objectives. An adaptive weighted-sum method with a variety of weight schedules is used instead of a traditional static weighted-sum technique. The idea is to reach more diverse solutions for different objectives. The empirical analysis performed on the Singapore train network dataset demonstrate the effectiveness of our approach.

Keywords Hyper-heuristics · Bi-objective Optimisation · Patrol Scheduling

1 Introduction

In this paper, we consider the Patrol Scheduling Problem (PSP) on a train network as studied in [10]. Hyper-heuristics have been previously used to solve similar type of problems as the security personnel routing and rostering problem [14]. The objective of those problems is to assign a number of security personnel to the sites where security is needed while respecting the contractual constraints of the personnel. Besides this rostering aspect of the problem,

M. Mısır, H. C. Lau
Living Analytics Research Centre, School of Information Systems, Singapore Management University
Tel.: +65-6808-5227
E-mail: mustafamisir@smu.edu.sg, hclau@smu.edu.sg

routing is also critical due to travelling between different sites. Related to this problem, the home care scheduling problem [15] which was solved by hyper-heuristics, also involves routing and rostering characteristics. The goal is to assign a group of carers to assist the people who need help at their homes. The PSP presented in this paper is a bi-objective optimisation problem for assigning a number of security teams to the train stations during a day. The first objective is to minimise the total travelling time spent by the security teams. The latter objective represents the coverage of the stations with respect to the stations' security requirements related to the station size and the passenger density. A hyper-heuristic framework is proposed to solve this problem.

Hyper-heuristics [2] are high-level search and optimisation techniques for managing a given set of heuristics or automatically generating heuristics. The primary reason behind studying hyper-heuristics is their problem-independent nature which separates the problem domain and the algorithm design components. This characteristic is achieved by performing search at a higher level, i.e. heuristic level, instead of a problem level. As a consequence, hyper-heuristics promise a high level of generality for solving different kinds of problems under varying search-related challenges. Hence, in principal, a hyper-heuristic can be applied to any target problem with no additional effort. This brings a unique advantage to hyper-heuristics in comparison to most of the existing search and optimisation methods.

Hyper-heuristic designs are generally divided into two types: *selection hyper-heuristics* and *generation hyper-heuristics*. The first type operates on a suite of existing low-level heuristics that are implemented to solve a given problem. The latter type aims at automatically building problem-specific heuristics, particularly via genetic programming [3, 1] and hybridisation. This paper is concerned with selection hyper-heuristics. A traditional selection hyper-heuristic is composed of a heuristic selection method and a move acceptance criterion. The selection method tries to choose the best heuristics at each decision step. The acceptance part is required to evaluate the performance of these chosen heuristics for deciding whether to accept or reject the solutions generated by the heuristics.

Majority of existing hyper-heuristics aim at solving single-objective optimisation problems even though these problems could be multi-objective in nature. The usual methodology to solve these multi-objective problems is to consider them as single-objective optimisation problems by defining a weighted sum of the objectives. Although this idea is reasonable to quickly deliver solutions, it is likely to suffer from missing good solutions. Besides that, it is hard to assign appropriate weights to the objectives since they are usually different in metrics and their precise importance is unobvious. In this respect, a plausible solution strategy would be to deliver a number of solutions forming a set call *pareto front* [7]. A pareto front refers to a group of solutions that are non-dominated considering all the objectives. Non-dominance of solutions mean that none of these solutions are better in terms of all the objectives. Besides the quality of the solutions with respect to each objective separately, it is critical to maintain some level of diversity to have a good pareto front. Providing high diversity gives a high variety of solutions that can meet different needs. In order to furnish such diverse solutions in a weighted-sum setting, a hyper-heuristic framework with a number of weight adaptation schemes is proposed. The experimental results on the PSP show the diversity performance of the proposed approach.

The paper continues as follows. Section 2 defines the problem. Section 3 explains the hyper-heuristic approaches used to solve the problem. A detailed experimental analysis is provided in Section 4. Section 5 finalises the paper with a discussion and possible future research.

2 The Patrol Scheduling Problem

The Patrol Scheduling Problem (PSP) studied here is about addressing the security personnel requirements of a train network composed a group of stations. Each train station needs a number of security teams that should be present during different time periods. A PSP solution provides assignments of the available security teams to the stations. The goal is to generate solutions requiring short travels between a set of stations while providing better security by taking the stations' risks into account. The PSP with the first objective aiming to minimise the travelling distance was studied in [10]. A real-world dataset on the Singapore MRT network was used for the experiments. It was shown in CPLEX failed to find a solution for this particular instance within a reasonable amount of computation time. The problem was then solved by considering each line as a separate problem. The PSP with the both objectives was approached in [11], where an exact model was introduced.

Besides optimising the aforementioned objectives, a feasible PSP solution should satisfy the following constraints:

- Each team can visit only one station during each time period
- Each station should be visited at least for a number of minimum visits requested
- Each station should not be visited more than a number of maximum visits requested
- Each station cannot be visited more than a single team during each time period
- Stations visited by each team should be reachable from one station to the next
- Break periods of each team should be respected

The PSP objectives are considered in the basic weighted-sum form as generalised in Equation 1. In the equation, w_i refers to the weight for the objective o_i. However, it is well known that the weighted sum approach suffers the challenge to determine reasonable weights for the objectives. From the search landscape perspective, changing the weights can result in a fitness function converting a hard landscape to an easy one, or vice versa. It is hence a challenging task to determine what the weights should be in order to have a easy-to-search landscape for a particular algorithm. Moreover, when different parts of a landscape are separately analysed, it is even likely to see that different weights can be useful for different parts. Our proposed strategy aims at easily accessing distinct search regions which result in high solution diversity with better solution quality.

$$\sum_{i}^{n} w_i \times o_i \text{ where } \sum_{i}^{n} w_i = 1 \tag{1}$$

3 A Diversity-Oriented Hyper-heuristic Framework

We apply multi-objective selection hyper-heuristics for solving the bi-objective PSP. In the literature, a limited number of hyper-heuristics were introduced for the multi-objective optimisation, where each objective is separately considered. TSRoulWheel [4] was introduced as a selection hyper-heuristic that learns the right heuristics for optimising each objective. A genetic-programming based hyper-heuristic was proposed for automatically generating heuristics to solve the bounded-diameter minimum spanning tree problem in [9]. A population-based Markov chain hyper-heuristic incorporating reinforcement learning was proposed in [13]. Another population-based multi-objective hyper-heuristic was studied to solve the 2D guillotine strip packing and 2D cutting stock problems in [6]. A multi-objective version of a hyper-heuristic with choice function was proposed in [12].

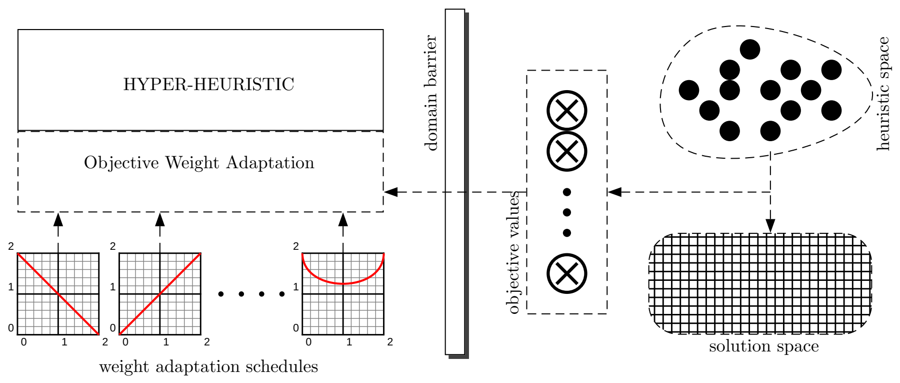


Fig. 1 A multi-objective single-point search hyper-heuristic framework

Unlike the existing hyper-heuristic methods, we introduce a hyper-heuristic framework targeting at solution diversity for multi-objective optimisation. Figure 1 illustrates our proposed framework. The framework is based on single-point search selection hyper-heuristics which manages a set of given low-level heuristics to deliver quick and high quality solutions while manipulating a

Algorithm 1: A multi-objective hyper-heuristic framework

```
1  Solution initialisation: $S \leftarrow S_{init}$
2  Check solution quality: $Q = eval(S)$ where $Q = \{v, o_1, o_2, ..., o_n\}$ and
   $f(S) = \sum_j^n w_j \times o_j + v$
   $v$: violation, $o_j$: solution quality wrt. objective $j$, $w_j$: weight for the objective $j$
3  while !stoppingCriteria() do
4      Choose a $LLH_i$
5      Generate a new solution: $S' \leftarrow LLH_i(S)$
6      Evaluate $S'$: $Q' = eval(S')$ and $f(S')$
8      if accept(f(S), f(S')) then
9          $S \leftarrow S'$
10         $updateParetoFront(S)$
       end
11     $updateWeights(W)$
   end
```

single solution. We incorporate the idea of adaptive weights when a multi-objective optimisation problem is to be solved. For the weight adaptation, it is required to have one or more functions that provide update schedules. Updating weights actually refer to changing the focus of a search. In other words, if the weight of a particular objective is higher than other objectives, the solutions found by an algorithm are likely to be better for this objective. Thus, changing weights means changing the search direction of an algorithm. For this process, three basic functions are used. In the bi-objective case, the first function is *Linear* that sets the weight of the first objective to 1 and the weight of the second objective is set to 0. The weights linearly changes in the other direction over time. In the final phase of search, the first weight becomes 0 while the second objective is set to 1. The second function, i.e. *sin180*, simply updates the weights between 1-0-1 considering the spent time for the 180 degrees of the Sine function. Hence, the objective focus starts from the first objective, gradually moves to the second objective and comes back to the first objective. The inverse case, i.e. *cos180*, applies the Cosine function for updating the weights. Of course, updating weights at each iteration may result in moving around a very small search region. That way, each update is performed at each $1/50$ time of the whole search process. Algorithm 1 explains the steps of the complete framework in details.

For applying this framework, the simple random heuristic selection mechanism [5] is combined with the great deluge move acceptance criterion [8] using exponential diversification scheme. The selection method randomly chooses a heuristic at each iteration. The acceptance criterion accepts better or equal quality solutions and accepts worsening solutions w.r.t. the initial solution and time. Algorithm 2 explains the acceptance procedure.

For initialisation, solutions are randomly constructed while taking some of the constraints into account in order to deliver (near-)feasible solutions. In particular, station consecutiveness, break times and team availability information were considered.

Algorithm 2: Great deluge move acceptance

```
1 if $f(S') \leq f(S)$ then
2     $S \leftarrow S'$
3 else if $f(S') \leq f(S_{initial}) \times (t_{remaining}/t_{total})^2$ then
4     $S \leftarrow S'$
  end
```

7 low-level heuristics are implemented to solve the PSP. These heuristics are detailed as follows:

- *LLH*_1: Change a randomly selected visit with another station
- *LLH*_2: Shift left visits from a randomly selected team and add a randomly selected visit instead of last shifted visit while removing the first visit
- *LLH*_3: Shift right visits from a randomly selected team and add a randomly selected visit instead of first shifted visit while removing the last visit
- *LLH*_4: Change a randomly selected visit causing per station minimum visit violation
- *LLH*_5: Change a randomly selected visit causing per station maximum visit violation
- *LLH*_6: Swap two visits between two randomly selected team
- *LLH*_7: Swap two visits for a randomly selected team

4 Computational Results

The experiments are performed on an Intel i5 1.7 GHz PC with 4 GB of memory. Each test is repeated for 10 times due to the stochastic nature of the hyper-heuristics. Different execution time limits are used for the PSP instances retrieved from the Singapore MRT network as shown in Figure 2. Table 1 presents these instances. Each of these instances is spread across 20 time periods. The first 4 instances represent separate lines. The EW+NS instance is a combination of two lines and EW+NS+NE is composed of three lines as stated in their instance names. The last instance, i.e. ALL, refers to the complete train network involving all the aforementioned lines. 1 minute is set as the running time for the first two instances. The execution time increases to 10 minutes for the next two instances and 30 minutes for the two subsequent instances. The proposed approach is run for 1 hour on the ALL instance.

Table 2 provides the best objective values found on each objective. The results indicate that there is no single weight adaptation scheme that will deliver the best performance. This is consistent with the underlying idea behind hyper-heuristics where there is no single heuristic that always work well. In this respect, a selection method for the update scheme or a learning method to actually adapt the update scheme might be an effective way to resolve this issue. The other way is to run all the update functions to deliver a pareto front together, which refers to the method called *Combined*.

Table 1 The patrol scheduling problem instances where the total number of periods is 20 (Per station: minimum number of visits = 1, maximum number of visits=2)

Instance	#Teams	#Stations
NE	3	16
CC	3	16
NS	5	25
EW	6	31
EW+NS	9	53
EW+NS+NE	12	67
ALL	14	79

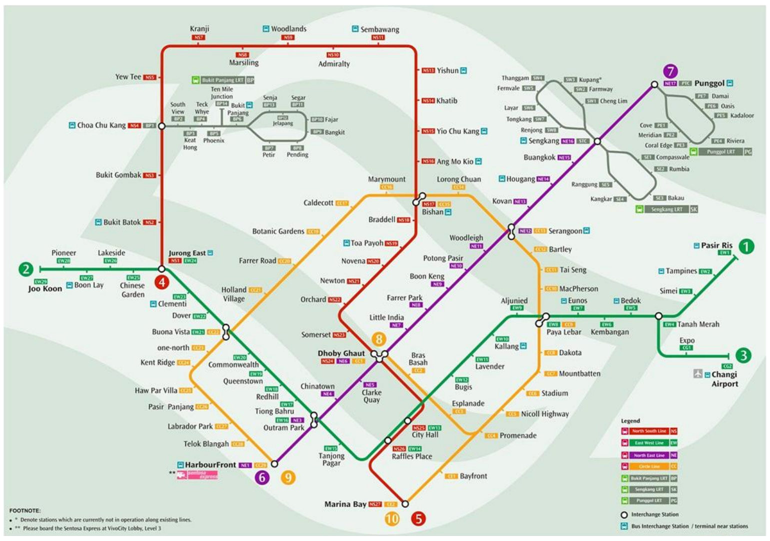


Fig. 2 The Singapore MRT network

Table 2 Best objective values achieved for the total distance travelled and the coverage of the stations w.r.t. their security team requirements (Distance|Coverage)

Instance	Linear	Cos180	Sin180	Combined
NE	15|367	15|325	15|361	15|367
CC	15|281	15|257	15|277	15|281
NS	25|947	25|955	25|954	25|955
EW	30|1009	30|1027	30|942	30|1027
EW+NS	47|1309	47|1379	47|1369	47|1379
EW+NS+NE	70|1893	71|1943	70|1958	70|1958
ALL	85|1693	80|1724	74|1685	74|1724

Figure 3 indicates the number of times when each station is visited in the pareto solutions. The solutions reveal that a few stations are frequently visited on certain time periods. For instance, the Raffles Place station that is used by both the NS and EW lines, is visited 9 times during the time period 13 as the highest frequently visited station on a single time period. It is additionally

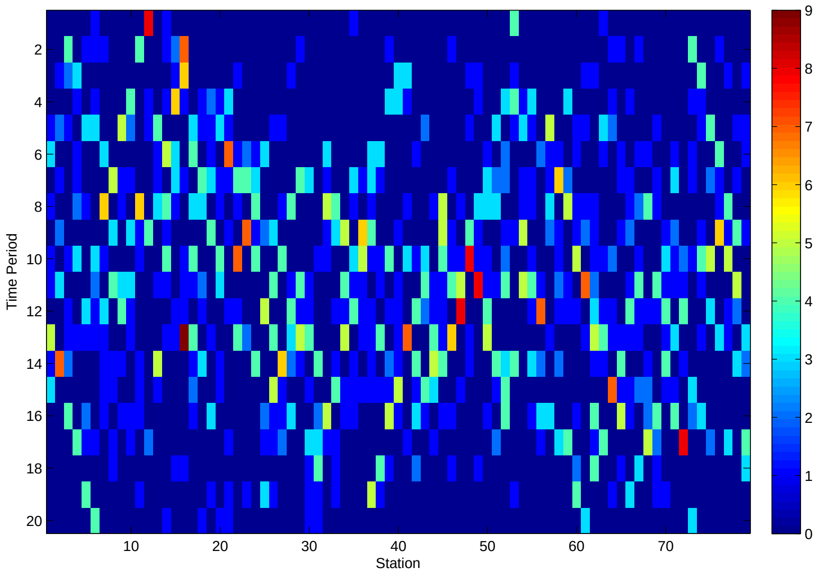


Fig. 3 Visit frequency of the stations on different time periods based on 16 pareto solutions found

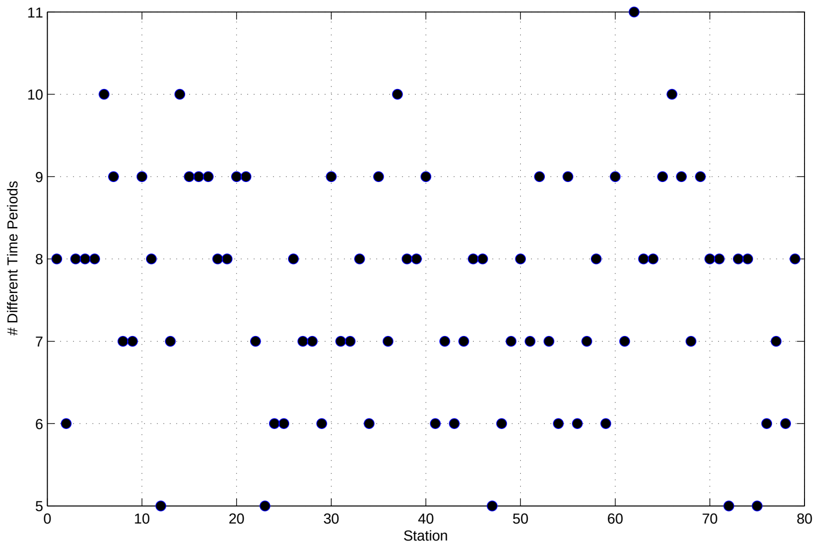


Fig. 4 Number of different time periods for each station visit based on 16 pareto solutions found

visited 7 times on the time period 2 and 6 times on the time period 3. However, it is still possible to visit this particular station during 9 different time periods due to the solution diversity provided by the proposed strategy. Among all the stations, each station is visited at least 4 times considering the highest visit frequency time periods like the City Hall station. The Serangoon station which is a common station between the CC and NE lines, is visited during 11 different time periods, thus it can be considered the most flexible station in terms of visiting time. Figure 4 shows clearer details about the number of different time periods where each station visited. The results indicate that diversity is achieved at the time period level by generating different patrol schedules.

Figure 5 presents the pareto fronts found after using each objective weighing schedules and the one using all methods as black-box, i.e. Combined. The results show that the linear schedule provides high diversity on both objectives while the solutions are relatively low quality compared to both the cos180 and sin180 schedules. Since the cos180 schedule aims to minimise the distance objective more, the solution quality in terms of this objective is better than the rest. However, this approach provides diversity on the other objective. Inversely, sin180 is able to deliver better solutions in terms of the coverage objective and higher diversity for the distance objective. In the Combined version, the pareto is composed of the solutions found both by using cos180 and sin180.

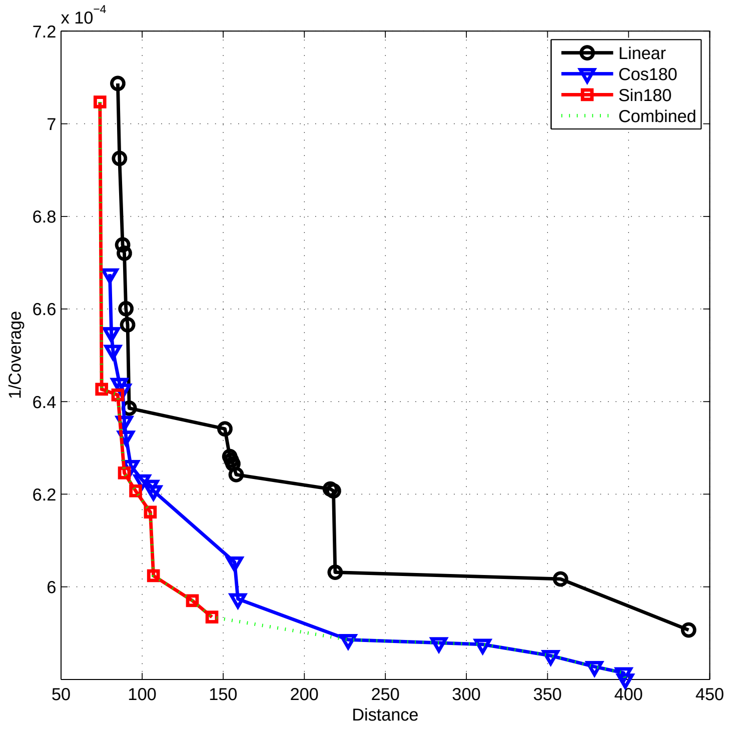


Fig. 5 Pareto fronts determined by different objective weighting schedules on the ALL instance

Figure 6 shows the pareto solutions returned for the remaining instances. The linear strategy delivers the best performance both on the NE and CC instances. Although sin180 has a higher effect on the combined pareto front of the NS instance, the combined pareto front consists of the solutions from all these weight schedule schemes. For the EW instance, the final pareto front blends the linear and cos180 solutions. All these three methods contribute to the pareto front of the EW+NS instance while the linear and sin180 schedules provide pareto solutions on the EW+NS+NE instance. As discussed on the numerical results, these pareto fronts indicate that there is no single weight adaptation strategy for high level solution diversity as well as a better pareto front. Thus, combining different weight adaptation schemes in a black-box form, i.e. Combined, is an effective way to overcome this issue. However, this doesn't necessarily mean that there is no a single mathematical function that can deliver similar or better pareto fronts.

5 Conclusion

This paper studies the problem of generating diverse schedules for the bi-objective patrol scheduling problem. We propose an approach for incorporating objective weight schedules or adaptation schemes that change over time as a single-point search selection hyper-heuristic framework. The idea is to change the objective focus by updating the objectives' weights while solving a given problem instance. The weight updating process is handled by incorporating basic mathematical functions. Besides independently using these functions, a combined approach is additionally proposed to deliver a better overall performance. Experimentally, we evaluated the performance in terms of solution diversity among the resulting pareto solutions. We performed empirical analysis on a real-world dataset for the Singapore rail network, and our results indicated that the weights are extremely critical for diversity. Among the tested weight update schemes, there is no a single scheme which always works well. Thus, choosing the right scheme can be considered another interesting selection problem for hyper-heuristics research. However, we showed that the combined strategy using the strengths of multiple update schemes addressed this issue reasonably well.

Our future research will be about incorporating better weight update schemes using different functions, not just for patrol scheduling, or potentially for any multi-objective optimization problem where solution diversity is the key concern. The test domains will also be extended to those with more than two objectives to evaluate the generality of our approach. Finally, a distributed version of this approach will be devised to take advantage of using multiple machines.

Acknowledgements This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA).

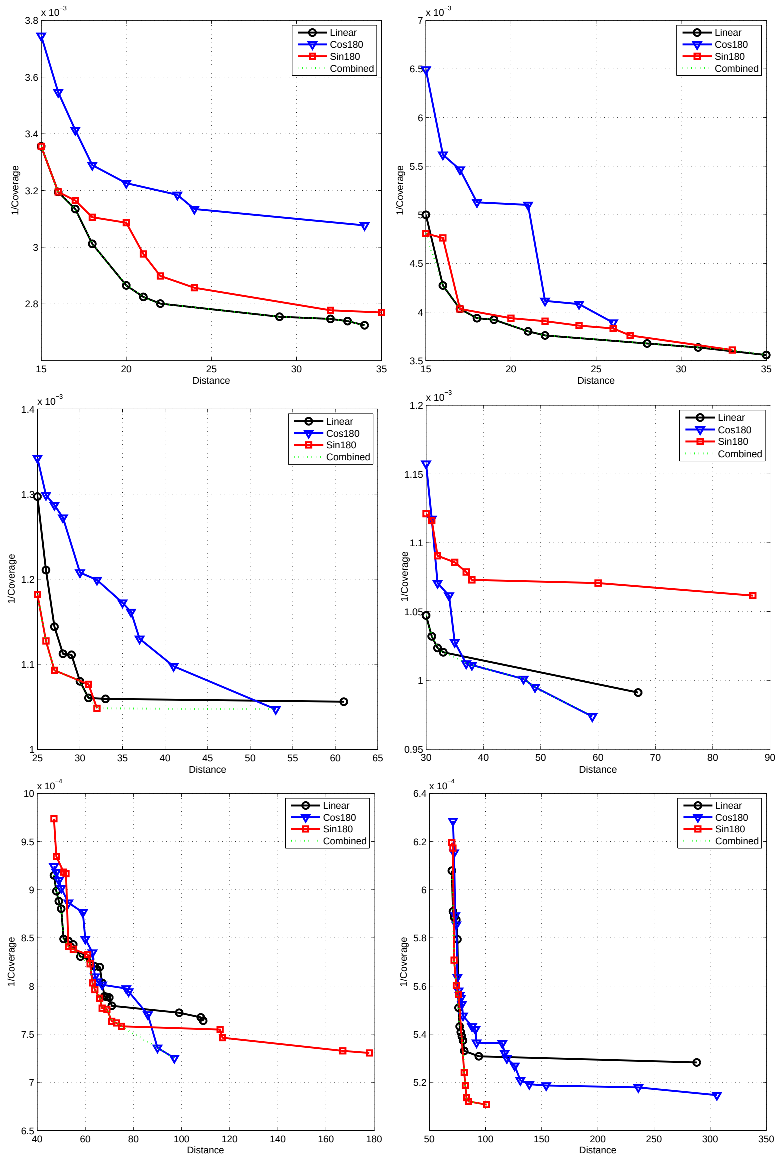


Fig. 6 Pareto fronts determined by different objective weighting schedules on the NE, CC, NS, EW, EW+NS and EW+NS+NE instances (from left to right, top to bottom)

References

1. Bader-El-Den, M., Poli, R., Fatima, S.: Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. Memetic Computing **1**(3), 205–219 (2009)
2. Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society

695–1724 (2013)

3. Burke, E., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.: Exploring hyper-heuristic methodologies with genetic programming. Collaborative Computational Intelligence. Springer (2009)
4. Burke, E., Silva, J.L., Soubeiga, E.: chap. Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, pp. 129–158. Springer (2005)
5. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Selected papers from the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT'00), pp. 176–190. Springer-Verlag, London, UK (2001)
6. de Armas, J., Miranda, G., León, C.: Hyperheuristic encoding scheme for multi-objective guillotine cutting problems. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 1683–1690. ACM (2011)
7. Deb, K., et al.: Multi-objective optimization using evolutionary algorithms, vol. 2012. John Wiley & Sons Chichester (2001)
8. Dueck, G.: New optimization heuristics: The great deluge algorithm and the record-to-record travel. Journal of Computational Physics **104**(1), 86–92 (1993)
9. Kumar, R., Bal, B.K., Rockett, P.I.: Multiobjective genetic programming approach to evolving heuristics for the bounded diameter minimum spanning tree problem. In: Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO'09), pp. 309–316. ACM (2009)
10. Lau, H.C., Gunawan, A.: The patrol scheduling problem. In: Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT'12). Son, Norway (2012)
11. Lau, H.C., Yuan, Z., Gunawan, A.: Patrol scheduling in urban rail network. Annals of Operations Research (In press)
12. Maashi, M., Özcan, E., Kendall, G.: A multi-objective hyper-heuristic based on choice function. Expert Systems with Applications (to appear)
13. McClymont, K., Keedwell, E.: Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11), pp. 2003–2010 (2011)
14. Mısır, M., Smet, P., Verbeeck, K., Vanden Berghe, G.: Security personnel routing and rostering: a hyper-heuristic approach. In: Proceedings of the 3rd International Conference on Applied Operational Research (ICAOR'11), *LNMS*, vol. 3, pp. 193–205. Istanbul, Turkey (2011)
15. Mısır, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10), pp. 2875–2882. Barcelona, Spain (2010)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

The Connectedness of Clash-free Timetables

Moritz Mühlenthaler · Rolf Wanka

Abstract We investigate the connectedness of clash-free timetables with respect to the Kempe-exchange operation. This investigation is related to the connectedness of the search space of timetabling problem instances, which is a desirable property, for example for two-step algorithms using the Kempe-exchange during the optimization step. The theoretical framework for our investigations is based on the study of reconfiguration graphs, which model the search space of timetabling problems. We contribute to this framework by including period availability requirements in the analysis and we derive improved conditions for the connectedness of clash-free timetables in this setting. We further show that the diameter of the reconfiguration graphs increases only linearly due to the period availability requirements. We apply the theoretical insights to establish the connectedness of clash-free timetables for a number of benchmark instances.

1 Introduction

According to the classification of heuristic optimization algorithms for timetabling problems in [17], many approaches in the literature fall in the category of *two-step optimization algorithms*. The general procedure is the following: In the first step, the underlying search problem is solved and the resulting feasible solution is used as a starting point for the second step, during which the optimization is performed. In the second step only feasible solutions are considered. A recent example of a state-of-the-art two-step approach is [19], numerous other examples can be found in [17]. During the optimization step, feasible timetables are modified using Kempe-exchanges or similar operations that preserve their feasibility. It is natural to ask whether any feasible timetable, in particular an optimal one, can be reached from an initial feasible timetable. We give a partial answer to this question by investigating conditions that establish the connectedness of the search space of *clash-free* timetables.

A timetable is clash-free, if no two conflicting events are scheduled simultaneously. In our analysis, we model the structure of the search space of clash-free timetables in terms

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,rolf.wanka}@cs.fau.de

of *reconfiguration graphs*. Such graphs have been studied in the context of *reconfiguration problems*. Given an instance $\mathscr{I}$ of a combinatorial search problem, the corresponding reconfiguration problem asks whether one feasible solution to $\mathscr{I}$ can be transformed into another feasible solution in a step-by-step manner by making local changes, such that each intermediate solution is also feasible. Reconfiguration variants of classical combinatorial problems have been studied for example in [4, 13, 14, 15]. The heart of the matter of timetabling problems in the academic context is the vertex coloring problem: A clash-free timetable corresponds to a proper coloring of the event conflict graph, see e.g. [9]. The connectedness of the (proper) vertex colorings of a graph has been investigated for example in [5, 8, 2]. The local change applied to a coloring in these works is an *elementary recoloring*, which changes the color of an individual node of the graph. In [16], Las Vergas and Meyniel establish conditions for the connectedness of vertex colorings using on a more general local change, the Kempe-exchange. The Kempe-exchange is a popular operation used by algorithms for timetabling problems for exploring the search space, including many of the two-step algorithms cited above. Therefore, their results can be applied in the timetabling context. Clash-freeness is typically necessary but not sufficient for a timetable to be feasible.

In many timetabling problem formulations (see e.g. [7, 27, 6]) a set of available time periods is given for each event, and all events are required to be placed strictly in their available time periods. We extend the techniques from [16] to derive conditions for the connectedness of clash-free timetables that satisfy period availability requirements. We further show that the diameter of the corresponding reconfiguration graphs increases only linearly (in the number of events) due to the period availability requirements. Our evaluation indicates the connectedness of clash-free timetables for a number of benchmark instance sets, with and without period availability requirements.

The remainder of this work is organized as follows: In Section 2 we provide the basic formalisms required for our analysis of the connectedness of clash-free timetables presented in Section 3. In Section 4 we investigate the connectedness of the clash-free timetables for number of standard benchmarking instance sets.

2 Background

2.1 The University Timetabling Problem

The University Timetabling Problem (UTP) formalizes in terms of a search problem the task of creating a course or examination schedule at a university.

Definition 1 (University Timetabling Problem (UTP))
INSTANCE:

- a set of events $E = \{e_1, \dots, e_n\}$
- a set of rooms $R = \{r_1, \dots, r_\ell\}$
- a set of time periods $P = \{p_1, \dots, p_k\}$
- a graph $G = (E, L)$ with nodes E and edges $L \subseteq \{\{u,v\} \mid u,v \in E\}$

The graph G is referred to as the *conflict graph*. Two events are called *conflicting* if they are adjacent in G. The set $P \times R$ contains the *resources*. A *timetable* τ is an assignment $\tau : E \to P \times R$. Two events e, e' are *overlapping*, if $e \neq e'$ and $\tau(e) = \tau(e')$. A timetable is called *overlap-free* if no two events overlap. Two events e, e' are *clashing* in τ, if they are conflicting and they are assigned to the same period. A timetable is *feasible*, if it is clash-free

and overlap-free.
TASK: Find a feasible timetable.

The UTP as defined above is equivalent to the problem given in [9, Section 3.4]. The clash-freeness requirement and its relation to the vertex coloring problem is the heart of the matter of timetabling problems in the academic context, see generally [9, 26]. Other kinds of requirements such as *availability requirements* and *precedence requirements* often occur in practice, see e.g. [7, 27], and in the benchmarking problem models, see e.g. [6, 12, 22]. Later, we will consider the UTP above with additional period availability requirements. These requirements mandate that only specific periods can be assigned to an event. We formalize period availability requirements in terms of an availability function α, which determines for each event the set of available periods:

$$\alpha : E \to \mathscr{P}(P) \enspace .$$

An important subproblem of the UTP is the *room assignment problem*. Given a period $p \in P$, then events $E' \subseteq E$ admit a room assignment, if there is an assignment $\rho : E' \to R$ such that $(p, \rho(e))$ is available for each $e \in E'$.

2.2 Vertex Coloring

A graph $G = (V(G), E(G))$, for short $G = (V, E)$, consists of a set of *vertices* V and a set of *edges* $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Unless stated otherwise, we assume that graphs are loopless and finite. We denote by $u — v$ that the vertices u and v are adjacent, i.e., $\{u, v\} \in E$. The graph $G[U]$ denotes the subgraph of G induced by the vertices $U \subseteq V(G)$. G is a mapping $c : V \to \{1, \dots, k\}$ that assigns one of the colors $\{1, \dots, k\}$ to each vertex of G. A coloring is called *proper*, if no two adjacent nodes have the same color. Unless stated otherwise, we will use the term *coloring* as a shorthand for *proper coloring*. The *vertex coloring problem* asks, whether a graph admits a k-coloring. A k-coloring c of G decomposes the vertices of G into k independent sets called *color classes*. A color class $a \in \{1, \dots, k\}$ contains all vertices of color a. We denote by $G(a, b)$ the bipartite subgraph induced by the color classes a and b. A connected component in $G(a, b)$ is referred to as *Kempe-component*.

Given a set $L(v)$ (called *list*) of available colors for each $v \in V$, a *list coloring* $c : V \to \bigcup_{v \in V} L(v)$ of G is a coloring of G such that $c(v) \in L(v)$ for each $v \in V$. Graph coloring is a special case of list coloring, where all colors are available for each node. By using a standard technique, see e.g. [9, Proposition 3.2], list coloring can be reduced to vertex coloring: Let the colors be labeled $1, \dots, k$, where $k = |\bigcup_{v \in V} L(v)|$. Now, let the graph G' be a copy of G to which we add a clique C on k (new) nodes $v_1, \dots, v_k$. For each $v \in V(G)$, we add an edge $v — v_i$ to G', whenever $i \notin L(v)$. Clearly, G' admits a k-coloring if and only if G admits a list coloring. The problem of deciding if a given UTP instance admits a clash-free timetable that satisfies period availability requirements is equivalent to deciding if the conflict graph admits a list coloring, where the $L(e) = \alpha(e)$ for each event e.

2.3 The Vertex Coloring Reconfiguration Problem

Reconfiguration problems formalize the question, if a solution to a problem instance can be transformed into another solution in a step-by-step manner by some reconfiguration operation, such that each intermediate solution is feasible [14]. Reconfiguration variants of

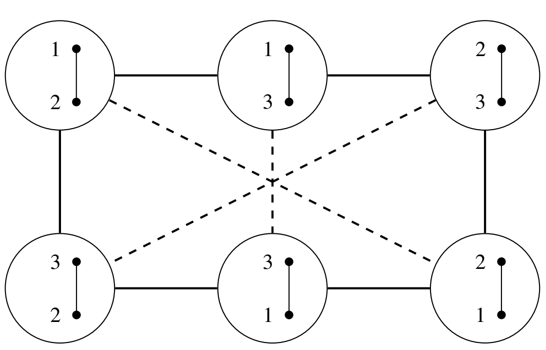


Fig. 1: The (Kempe-)3-coloring graph of the graph K_2. Solid edges correspond to elementary recolorings. Dashed edges correspond to Kempe-exchanges that are not equivalent to an elementary recoloring.

the vertex coloring problem have been studied for example in [5, 2, 23, 3]. In this context, *elementary recolorings* and *Kempe-exchanges* have been considered as reconfiguration operations. Given a coloring c of a graph G, an elementary recoloring changes the color of a single vertex of G. Two k-colorings c_1 and c_2 of G are adjacent, $c_1 \sim_E c_2$, if there is an elementary recoloring that transforms c_1 into c_2. The Kempe-exchange is a generalization of the elementary recoloring operation. Given two colors a and b, a Kempe-exchange switches the colors of a Kempe-component, i.e., a connected component in $G(a,b)$. The result of this operation is a new coloring, such that, within the Kempe-component, all vertices of the of color a are assigned to color b and vice versa. Two colorings c_1 and c_2 of G are adjacent with respect to the Kempe-exchange, $c_1 \sim_K c_2$, if there is a Kempe-exchange that transforms c_1 into c_2. Each of the two adjacency relations $\sim_E$ and $\sim_K$ gives rise to a graph structure on the set of k-colorings of G.

Definition 2 ((Kempe-)k-coloring graph) For a graph $G = (V,E)$ and $k \in \mathbb{N}$ let

$$\begin{aligned}
\mathscr{V} &:= \{c : V \to \{1,\dots,k\} \mid c \text{ is a } k\text{-coloring of } G\} \\
\mathscr{E}_E &:= \{\{c_1,c_2\} \mid c_1,c_2 \in \mathscr{V} \text{ and } c_1 \sim_E c_2\} \\
\mathscr{E}_K &:= \{\{c_1,c_2\} \mid c_1,c_2 \in \mathscr{V} \text{ and } c_1 \sim_K c_2\} \quad .
\end{aligned}$$

Then the k-coloring graph is the graph $\mathscr{C}_k(G) = (\mathscr{V},\mathscr{E}_E)$. The Kempe-$k$-coloring graph is the graph $\mathscr{K}_k(G) = (\mathscr{V},\mathscr{E}_K)$.

Figure 1 shows $\mathscr{C}_3(K_2)$ and $\mathscr{K}_3(K_2)$, where K_2 is the graph consisting of two vertices connected by an edge. The diameter and the connectedness of (Kempe-)k-coloring graphs have been investigated in [23, 2, 3]. The analysis of the UTP search space will follow this line of research. A graph G is called k-degenerate, if its vertices can be ordered such that each vertex has at most k neighbors preceding it. The smallest k for which G admits such an ordering is the *degeneracy* $\deg(G)$. A witness vertex ordering of the degeneracy $\deg(G)$ can be found by repeatedly removing vertices of minimal degree [21, 28]. Equivalently, the degeneracy is the largest minimum degree of any subgraph. Let $S(G)$ be the set of permutations of the vertices of G and let $\text{pred}(v,\sigma)$ denote the number of neighbors of the vertex $v \in V(G)$ that precede v in the ordering $\sigma \in S(G)$. In formal terms, the two characterizations

Algorithm 1: KEMPERECONFIGURATION

```
input  : graph $G$, labeling $v_1,\dots,v_n$ of the vertices, $k$-colorings $c_1$, $c_2$ of $G$
output: list of Kempe-exchanges transforming $c_1$ into $c_2$
data   : array $c$ of length $n$ storing the current color of each vertex, list $K$ of Kempe-exchanges

$K \longleftarrow$ empty list;
for $i \longleftarrow 1$ to $n$:
    $c[i] \longleftarrow c_1(v_i)$

for $i \longleftarrow 1$ to $n$:
    $H \longleftarrow G[v_1,\dots,v_i]$;
    /* Kempe-exchange $\kappa = (a,b,u)$, where $a,b$ are colors and $u \in V(H)$   */
    for $\kappa = (a,b,u) \in K$:
        without loss of generality $c[i] \neq a$;
1       if $c[i] = b$ and $v_i$ has exactly one neighbor of color $a$ in $H$:
            $c[i] \longleftarrow a$;
2       else if $c[i] = b$ and $v_i$ has at least two neighbors of color $a$ in $H$:
            choose color $b' \neq b$, which is not used by any neighbor of $v_i$ in $H$;
            insert Kempe-exchange $(b,b',v_i)$ right before $\kappa$ in $K$;
            $c[i] \longleftarrow b'$;
3   append Kempe-exchange $(c_2(v_i),c[i],v_i)$ to $K$;
return $K$;
```

of $\deg(G)$ can be stated as follows:

$$\deg(G) := \max_{H\subseteq G} \min_{v\in V(H)} \{d_H(v)\} = \min_{\sigma\in S(G)} \max_{v\in V(G)} \operatorname{pred}(v,\sigma) \ , \tag{1}$$

where $d_H(v)$ denotes the degree of v in H. The degeneracy of a graph is an upper bound on its chromatic number. Furthermore, the degeneracy has been used to establish the connectedness of Kempe-k-coloring graphs:

Theorem 1 ([16, Proposition 2.1]) *For any graph G, the Kempe-k-coloring graph $\mathscr{K}_k(G)$ is connected if $k > \deg(G)$.* □

The proofs given in [16, 23] are essentially an analysis of the algorithm KEMPERECONFIGURATION shown in Algorithm 1. This algorithm transforms a source coloring c_1 into a destination coloring c_2 by a sequence of Kempe-exchanges, provided that a sufficient number of colors is available. The vertices are processed one-by-one according to the given labelling. The general idea is to prevent the current vertex from interfering with the Kempe-exchanges dealing with the previously processed vertices.

3 The Connectedness of Clash-free Timetables

In the following, let G be the conflict graph G of a UTP instance $\mathscr{I}$ with periods $\{1,\dots,p\}$ and let α be the period availability function. Further, let G' be the graph derived from G by the reduction from list coloring to vertex coloring from Section 2.2. In our analysis, we consider timetables that differ only with respect to how rooms are assigned as equivalent. Each p-coloring of G corresponds to an equivalence class of clash-free timetables. Thus, the adjacency relation $\sim_K$ on the p-colorings of G induces an adjacency relation on the clash-free timetables and therefore, $\mathscr{K}_p(G)$ models the search space of clash-free timetables connected by Kempe-exchanges. If $\mathscr{K}_p(G)$ is connected, then a two-step algorithm that uses Kempe-exchanges in order to explore the search space can reach an optimal solution from any starting point. If not, then the algorithm may fail to find an optimal solution due to the structure of the search space.

In most applications, clash-freeness is not the only requirement a timetable needs to satisfy. Additional types of requirements such as period availability requirements, room availability requirements, and overlap-freeness requirements restrict the set of feasible timetables, and, as a consequence, an equivalence class corresponding to a coloring may be empty. Let C be the set of colorings of G that correspond to non-empty equivalence classes of timetables. Then the search space of $\mathscr{I}$ is connected if $\mathscr{K}_p(G)[C]$ is connected. In particular, for the additional requirements above, the corresponding reconfiguration graphs are the subgraphs of $\mathscr{K}_p(G)$ induced by the following sets of nodes:

1. period availability requirements:
$$C_\pi = \{c \in V(\mathscr{K}_p(G)) \mid \forall v \in V(G) : c(v) \text{ is available for event } v\}$$
2. overlap freeness and room availability requirements:
$$C_\rho = \{c \in V(\mathscr{K}_p(G)) \mid \forall i \in P : \text{ color class } i \text{ admits a room assignment}\}$$

Conditions establishing the connectedness of $\mathscr{K}_p(G)$ result directly from Theorem 1.

Corollary 1 *The search space of clash-free timetables is connected if $p > \deg(G)$.* □

Regarding overlap freeness and room availability requirements, to the best of our knowledge, the properties of the corresponding reconfiguration graphs have not been studied so far. The *bounded vertex k-coloring problem* with bound $b \in \mathbb{N}$ is the problem of coloring a graph with k colors such that each color is used at most b times. The bounded vertex coloring problem has been studied for example by Lucarelli [20], and Baker and Coffmann [1] in the setting of unit-time task scheduling on multiple processors and by de Werra in the timetabling context [10]. If overlap freeness is required and no particular room availability requirements are present, then the graph $\mathscr{K}_P(G)[C_\rho]$ is the reconfiguration graph of a bounded vertex coloring instance. The reconfiguration variant of the bounded vertex coloring problem seems to be an interesting problem which deserves further investigation. The situation gets more involved if room availability requirements are present. Checking if the k events in a color class admit a room assignment is equivalent to checking if a bipartite graph admits a matching of cardinality k.

We will now focus on structural properties of $\mathscr{K}_p(G)[C_\pi]$. First, we show that $\mathscr{K}_p(G)[C_\pi]$ is connected if and only if $\mathscr{K}_p(G')$ is connected. The main obstacle is that there is no Kempe-exchange on $\mathscr{K}_p(G)[C_\pi]$ corresponding to a Kempe-exchange on G' involving any of the nodes $v_1, \dots, v_p$. We construct a graph K, which is a copy of $\mathscr{K}_p(G)[C_\pi]$ with a self loop added to each node. Additionally, we add to K an edge between two colorings $u, v \in V(K)$, if there are two colors i and j such that u can be transformed into v by swapping the colors in all except a single connected component of $G(i, j)$. These additional edges are merely shortcuts for several individual Kempe-exchanges. Therefore, $\mathscr{K}_p(G)[C_\pi]$ is connected if and only if K is connected. Figure 2 shows the various graphs under consideration for a list-coloring instance consisting of a graph $G = (\{u, v\}, \{u — v\})$ and color lists $L(u) = \{1\}$ and $L(v) = \{2\}$. The nodes 1 and 2 of G' were added by the reduction from list to graph coloring.

Lemma 1 *There is a graph homomorphism $f : \mathscr{K}_p(G') \to K$.*

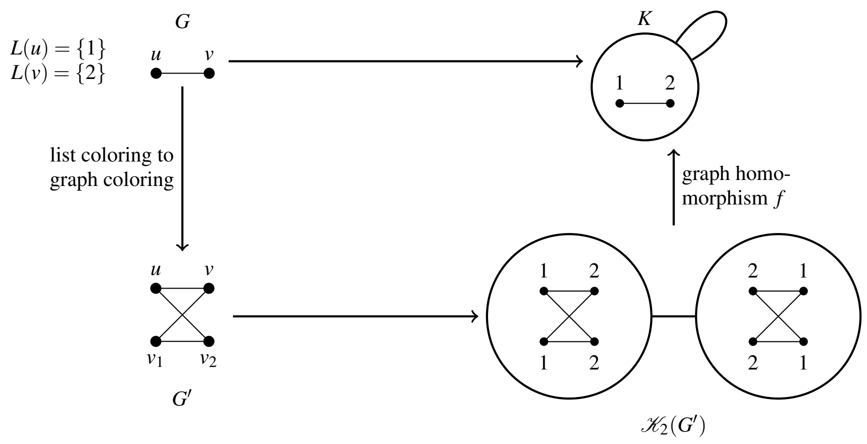


Fig. 2: Relations between the graphs G, G', K and $\mathscr{K}_p(G')$. The choice of G and the available colors determines the other graphs as described in the text. The existence of the graph homomorphism f is established by Lemma 1.

Proof We construct the mapping $f : V(\mathscr{K}_p(G')) \to V(K)$. Let $c \in V(\mathscr{K}_p(G'))$. First, we swap the colors of the p color classes such that v_i has color i for each $i \in \{1,\dots,p\}$. This can be achieved by applying a sequence of Kempe-exchanges to the coloring c: For each color $j \in \{1,\dots,p\}$, if the current color of v_j is $i \neq j$ we swap the colors in $G'(i,j)$. One Kempe-exchange is required for each Kempe-component of $G'(i,j)$. Let c' be the resulting coloring. Except for the vertices $v_1,\dots,v_p$ and their incident edges, G' is just a copy of G. Now, pick $f(c) = \tilde{c}$, where $\tilde{c}$ is equivalent to c' restricted to the vertices $V(G) \subset V(G')$. Clearly, $\tilde{c}$ is a coloring of G. Due to the construction of G', $\tilde{c}$ satisfies the list coloring requirements for G, i.e., for each $v \in V(G)$ we have $c(v) \in \alpha(v)$. Therefore, $\tilde{c} \in V(K)$.

We show that the mapping f is a graph homomorphism as required. Let c, d be colorings of G' such that $c — d$ in $\mathscr{K}_p(G')$. Further, let κ be a witness of $c \sim_K d$. There are two cases to consider:

1. The Kempe-exchange κ does not involve any of the nodes $v_1,\dots,v_p$. Then f renames the color classes of the colorings c and d if required and there is a Kempe-exchange corresponding to κ that establishes $f(c) — f(d)$ in K.
2. The Kempe-exchange κ involves two nodes $u,v \in \{v_1,\dots,v_p\}$. We need to consider following two subcases. If $G'(c(u),c(v))$ is connected then $f(c) = f(d)$ and therefore, $f(c) — f(d)$, since each node of K has a self-loop. Otherwise, $f(c)$ and $f(d)$ differ with respect to the color classes $a(u)$ and $c(v)$. We show that $f(c)$ and $f(d)$ are connected by a sequence of Kempe-exchanges that swaps the colors in all except a single Kempe-component of $G'(c(u),c(v))$ and thus $f(c) — f(d)$ by the construction of K above. To obtain $f(b)$, we first apply κ to c on G' and then apply f to the resulting coloring. The Kempe-exchange κ swaps the colors of the connected component of $G'(c(u),c(v))$ containing u and v, and then f swaps the colors in $G'(c(u),c(v))$. As a result, $f(d)$ can be obtained from $f(c)$ by swapping the colors in $G'(c(u),d(v))$ except the one containing u and v in the preimage $f^{-1}(V(G(c(u),d(v))))$.

In summary, for all $c,d \in V(\mathscr{K}_p(G')) : c — c$ implies $f(c) — f(d)$. □

The graph homomorphism f induces the equivalence relation $\sim_f$ on $V(\mathscr{K}_p(G'))$: for $a,b \in V(\mathscr{K}_p(G')) : a \sim_f b$ if $f(a) = f(b)$.

Theorem 2 *$\mathscr{K}_p(G)[C_\pi]$ is connected if and only if $\mathscr{K}_p(G')$ is connected.*

Proof We noted above that $\mathscr{K}_p(G)[C_\pi]$ is connected if and only if K is connected. Let $f : \mathscr{K}_p(G') \to K$ be the graph homomorphism from Lemma 1.
"Only if" part: Let $\mathscr{K}_p(G')$ be connected. Then K is connected since there is a graph homomorphism $\mathscr{K}_p(G') \to K$, and graph homomorphisms preserve connectedness. Therefore, $\mathscr{K}_p(G)[C_\pi]$ is connected.
"If" part: Let $\mathscr{K}_p(G)[C_\pi]$ be connected. Then K is connected. Due to the first isomorphism theorem, $K \cong \mathscr{K}_p(G')_{/\sim_f}$ and thus, $\mathscr{K}_p(G')_{/\sim_f}$ is also connected. Any two colorings u, v of G' such that $u \sim_f v$ are connected by Kempe-exchanges since one can be obtained from the other by permuting the colors of the color classes. □

For general graphs, not much is known about the diameter of their corresponding Kempe-k-coloring graphs. Using the graph homomorphism from Lemma 1, we show that the reduction from list to graph coloring increases the (possibly unknown) diameter only moderately:

Theorem 3 $\operatorname{diam}(\mathscr{K}_p(G)[C_\pi]) \le \lfloor \frac{|V(G)|-1}{2} \rfloor \cdot \operatorname{diam}(\mathscr{K}_p(G'))$.

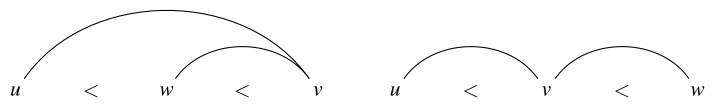
Proof For any adjacent nodes $c,d \in \mathscr{K}_p(G')$, we count how many Kempe-exchanges are required to get from $f(c)$ to $f(d)$ in $\mathscr{K}_p(G)[C_\pi]$. Let κ be the Kempe-exchange that is a witness of $c — d$, and let i and j be the involved color classes. If $c \sim_f d$ then, in the worst case, all except one connected component of $G(i,j)$ need to be switched to get from c to d for the reasons stated in cases 1 and 2 in the proof of Lemma 1. There are at most $\lfloor (|V(G)|-1)/2 \rfloor$ components and at most one Kempe-exchange is required for each of them. If $c \not\sim_f d$ then there is a single Kempe-exchange on G that establishes $f(c) — f(d)$. Thus, a shortest path of maximum length t in $\mathscr{K}_p(G')$ corresponds to a path of length at most $t \cdot \lfloor (|V(G)|-1)/2 \rfloor$ in $\mathscr{K}_p(G)[C_\pi]$. □

Given two colorings c and c' of G', the algorithm KEMPERECONFIGURATION transforms c into c' as long as there is a sufficient number of colors available. However, G' contains K_p as a subgraph therefore $\deg(G') \ge p$. According to Theorem 1, at least $p+1$ colors are needed by KEMPERECONFIGURATION and therefore Theorem 1 is not useful for proving the connectedness of clash-free timetables in the presence of period availability requirements. To overcome the limitations of Theorem 1, we fix the colors of the clique vertices $v_1,\dots,v_p$ of G'. As a consequence, if we exclude the clique from the recoloring process, the number of colors required by KEMPERECONFIGURATION is no longer dominated by the clique.

We will first consider the general case, where the colors of some vertices $F \subseteq V(G)$ are assumed to be fixed. We denote by $\overline{F} = V(G) \setminus F$ be the remaining vertices. Further, let $S' \subset S(G)$ be the vertex orderings satisfying

$$\forall u,v \in \overline{F}, w \in F : u < v \wedge u — v \wedge v — w \Rightarrow w < v \ . \tag{2}$$

That is, if v is a successor of u and they are adjacent, then all neighbors of v in F must precede v. Figure 3 shows two examples of vertex orderings of the graph $u — v — w$. For $F = \{w\}$, ordering 3a satisfies the condition in Eq. 2 and 3b does not. We will prove next that KEMPERECONFIGURATION does not change the color of any vertex in F if the vertices



(a) Ordering compatible with Eq. (2) (b) Ordering incompatible with Eq. (2)

Fig. 3: Two vertex orderings of the graph $u — v — w$, $F = \{w\}$.

$V(G)$ are processed according to an ordering in S'. In order to bound the number of colors required for our analysis, we introduce the following generalization of the degeneracy of a graph:

Definition 3 (Subdegeneracy) Let G be a graph and let $F \subseteq V(G)$. The *subdegeneracy* $\operatorname{subdeg}(F,G)$ of G relative to F is defined as:

$$\operatorname{subdeg}(F,G) = \min_{\sigma \in S'} \max_{v \in V(G) \setminus F} \operatorname{pred}(v,\sigma)$$

Note that $\operatorname{subdeg}(F,G) = \deg(G)$ if F is empty. If F is not empty then $\operatorname{subdeg}(F,G) \leq \deg(G)$. Intuitively, we are looking for a vertex ordering in S' that minimmizes the maximum number of adjacent predecessors of any vertex, however, the number of predecessors of any vertex in F is irrelevant.

Theorem 4 *Let c, c' be k-colorings of G that agree on F. Then* KEMPERECONFIGURATION *returns a sequence of Kempe-exchanges such that*

1. *all intermediate colorings also agree on F, and*
2. *no more than* $\operatorname{subdeg}(F,G) + 1$ *colors are required.*

Proof We first show that the colors of the vertices F are not changed by KEMPERECONFIGURATION. Assume for a contradiction that in some intermediate coloring a vertex $w \in F$ has a color different from $c(w)$. Then w has been recolored because a neighbor u of w preceding it in σ received color $c(w)$. There are two possible reasons: Either u was recolored to $c(w)$ because $c'(u) = c(w)$, but then $c'(w) \neq c(w)$, a contradiction. If this is not the case, then u was recolored in case 1 or 2 of KEMPERECONFIGURATION, because of a neighbor v preceding it. But this is a contradiction to $\sigma \in S'$.

We now show that $\operatorname{subdeg}(F,G) + 1$ colors are sufficient. Since the vertices in F are never recolored, we consider only the vertices $\overline{F}$. An unused color may be picked for a vertex $v \in \overline{F}$ in case 2 of Algorithm 1. For each $v \in \overline{F}$, there are at most $\operatorname{subdeg}(F,G)$ neighbors of v preceding it, and there are at most $\operatorname{subdeg}(F,G) - 1$ colors different from the color of v present among these vertices. Thus, there is at least one other color available for v. □

We propose a heuristic approach to finding a witness vertex ordering of $\operatorname{subdeg}(F,G)$. Let $\tilde{S} \subseteq S'$ be the vertex orderings such that the vertices F precede all other vertices. Recall that for any graph G a witness vertex ordering of the degeneracy $\deg(G)$ can be found by repeatedly removing vertices of minimal degree. In a similar fashion, we can determine an optimal solution to:

$$\lambda(F,G) := \min_{\sigma \in \tilde{S}} \max_{v \in \overline{F}} \operatorname{pred}(v,\sigma)$$

Moreover, $\lambda(F,G)$ is equivalently characterized by a max-min expression and the min-max expression above, analogous to the characterizations of the degeneracy shown in Eq. (1):

Algorithm 2: VERTEXELIMINATION

```
input  : graph $G$, vertices $F \subseteq V(G)$
output: ordering $v_1,\ldots,v_{|\overline{F}|}$ of the vertices $\overline{F} = V(G) \setminus F$

$G_{|D|} \longleftarrow G$ ;
for $i \longleftarrow |\overline{F}|$ downto 1 do
    choose $v_i$ from $\operatorname{argmin}_{v \in \overline{F}} \{\delta(v, G_i)\}$ ;
    $G_{i-1} \longleftarrow G_i - v_i$.
return $v_1,\ldots,v_{|\overline{F}|}$;
```

Theorem 5 *For any graph G and $F \subseteq V(G)$,*

$$\lambda(F,G) = \min_{\sigma \in \tilde{S}} \max_{v \in V(G)\setminus F} \operatorname{pred}(v,\sigma) = \max_{G[F] \subseteq H \subseteq G} \min_{v \in V(H)\setminus F} \{d_H(v)\} \ .$$

Furthermore, VERTEXELIMINATION *produces a witness vertex ordering for the min-max expression.*

Proof The proof is based on the remark on the optimality of VERTEXELIMINATION in [21]. Let $\ell = |\overline{F}|$ and for an ordering $v_1,\ldots,v_\ell$ of $\overline{F}$ let $G_i = G[F \cup \{v_1,\ldots,v_\ell\}]$. Further, let

$$\hat{\delta} := \max_{G[F] \subseteq H \subseteq G} \min_{v \in V(H)\setminus F} \{d(v,H)\} \ .$$

Intuitively, $\hat{\delta}$ is analogous to the degeneracy of G, but the vertices F are irrelevant. If an ordering $\sigma = v_1,\ldots,v_\ell$ of $\overline{F}$ is an output of VERTEXELIMINATION then

$$\begin{aligned}\max_{1 \le i \le \ell} \operatorname{pred}(v_i,\sigma) &= \max_{1 \le i \le \ell} \{d(v_i,G_i)\} \\ &= \max_{1 \le i \le \ell} \min_{v \in V(G_i)\setminus F} \{d(v,G_i)\} \le \hat{\delta} \ .\end{aligned}$$

The graphs G_i coincide with those in Algorithm 2.

Now let H^* be a graph such that $G[F] \subseteq H^* \subseteq G$ and

$$\min_{v \in V(H^*)\setminus F} \{d(v,H)\} = \hat{\delta} \ .$$

Let $v_1,\ldots,v_\ell$ be any ordering of $\overline{F}$ and let i be the smallest index such that $H^* \subseteq G_i$. Then v_i must be a vertex of H^* and $d(v_i,G_i) \ge \hat{\delta}$. Therefore, for any ordering $v_1,\ldots,v_\ell$ of $\overline{F}$, $\max_{1 \le j \le \ell}\{d(v_j,G_j)\} \ge \hat{\delta}$, with equality if the vertex ordering is an output of VERTEX-ELIMINATION. □

Certainly, the optimality of VERTEXELIMINATION is only established with respect to the subset $\tilde{S} \subseteq S'$. The vertex ordering obtained from the algorithm can potentially be improved by the following post-processing step: Let $v_1,\ldots,v_{|\overline{F}|}$ be an output of VERTEX-ELIMINATION and let k be the largest number such that $v_1,\ldots,v_k$ are independent. Then the vertices $v_1,\ldots,v_k$ can be moved before the vertices F in the ordering without violating condition (2). The resulting ordering $\sigma' \in S'$ is not in $\tilde{S}$ and can thus not be generated by VERTEXELIMINATION. There is a potential advantage because the construction guarantees that $\max_{v \in \overline{F}} \operatorname{pred}(v,\sigma') \le \max_{v \in \overline{F}} \operatorname{pred}(v,\sigma)$.

In summary, the heuristic for computing a vertex ordering $\sigma \in S'(G)$ such that the value $\max_{v \in \overline{F}} \operatorname{pred}(v,\sigma)$ is close to $\operatorname{subdeg}(F,G)$ performs the following two steps:

1. Run VERTEXELIMINATION to generate an ordering $v_1, \dots, v_{|\overline{F}|}$ of the vertices $\overline{F}$.
2. Let $k \in \mathbb{N}$ be the largest number such that $v_1, \dots, v_k$ are independent in G. Move the vertices $v_1, \dots, v_k$ before the vertices F in the ordering.

We apply this heuristic to prove the connectedness of the clash-free timetables that satisfy the availability constraint for a number of benchmark instances. First, we use the reduction from list to graph coloring described above to construct from a conflict graph G the graph G', which contains a clique $v_1, \dots, v_p$. Then we choose $F \subseteq V(G')$ to include every vertex with $p-1$ neighbors in $\{v_1, \dots, v_p\}$, that is

$$F = \{v \in V(G') \mid |\Gamma(v) \cap \{v_1, \dots, v_p\}| = p-1\} \ . \tag{3}$$

Now we can apply the heuristic to obtain an orderign $\sigma \in S'$ and thus an upper bound $\text{subdeg}'(F, G') = \max_{v \in V(G') \setminus F} \text{pred}(v, \sigma) \geq \text{subdeg}(F, G')$. If $p \geq \text{subdeg}'(F, G') + 1$ then Theorem 4 implies that the clash-free timetables are connected.

4 Results

We use the theory developed in the previous section to establish the connectedness of clash-free timetables for a range of UTP benchmark instances. By Theorem 1, reconfiguration graphs of clash-free timetables are connected if $p > \deg(G)$ and by Theorem 4, the reconfiguration graphs of the clash-free timetables that satisfy availability requirements are connected if $p > \text{subdeg}(F, G')$ for a suitably chosen $C \subseteq V(G')$. We use the heuristic from the previous section to determine a bound $\text{subdeg}'(F, G') \geq \text{subdeg}(F, G')$. The set F of "fixed" vertices is chosen as shown in Eq. (3).

Table 1 indicates the connectedness of the clash-free timetables according to theorems 1 and 4 for instances from the CB-CTT, PE-CTT benchmark sets, as well as instances from the University of Erlangen-Nürnberg. All instances can be obtained from the SaTT group website at the University of Udine [11]. The instances `comp01`,...,`comp21` are from the CB-CTT track of the International Timetabling Competition 2007 (ITC2007) competition. The instances `ITC2_i01`,...,`ITC2_i24` are from the PE-CTT track of the same competition. The `erlangen` instances are large real-world instances from the engineering department of the University of Erlangen-Nürnberg. The `toy` instance is a small example instance from the website [11]. For each instance we give the number of periods p, the degeneracy of the conflict graph $\deg(G)$, and the bound $\text{subdeg}'(F, G') \geq \text{subdeg}(F, G')$. Table entries in bold face indicate that the corresponding value $\deg(G)$ or $\text{subdeg}'(F, G')$ certifies the connectedness of the clash-free timetables.

According to the data in Table 1 the clash-free timetables for all CB-CTT and `erlangen` instances are connected, while the conditions imposed by Theorem 1 are not satisfied for any of the PE-CTT instances. For eight CB-CTT instances, the upper bound on $\text{subdeg}(F, G')$ is sufficient to show that the reconfiguration graphs are connected in the presence of availability constraints. The situation is quite different for the PE-CTT instances, since neither $\deg(G)$ nor $\text{subdeg}'(C, G')$ is sufficient to show the connectedness of the reconfiguration graphs, better bounds on $\text{subdeg}(F, G')$ are of no use here since $\text{subdeg}(F, G') \geq \deg(G)$. Therefore, new techniques are needed for proving the connectedness (or disconnectedness) of the reconfiguration graphs for these instances.

In Tables 2 and 3, the degeneracy values of the corresponding conflict graphs are given for the Lewis/Paechter [18] and the Metaheuristic Network [24] instance sets. On these

Table 1: For each instance from the CB-CTT, PE-CTT, and Erlangen instance sets, we give the number p of periods, $\deg(G')$ and an upper bound $\text{subdeg}'(C,G') \geq \text{subdeg}(F,G')$ produced by the heuristic. All instances are available from the website [11].

instance	p	$\deg(G)$	$\text{subdeg}'(C,G')$	instance	p	$\deg(G)$	$\text{subdeg}'(C,G')$
`comp01`	30	**23**	**24**	`ITC2_i01`	45	91	109
`comp02`	25	**23**	30	`ITC2_i02`	45	99	119
`comp03`	25	**22**	27	`ITC2_i03`	45	73	92
`comp04`	25	**17**	25	`ITC2_i04`	45	78	100
`comp05`	36	**26**	43	`ITC2_i05`	45	81	99
`comp06`	25	**17**	28	`ITC2_i06`	45	80	100
`comp07`	25	**20**	**24**	`ITC2_i07`	45	80	106
`comp08`	25	**20**	**24**	`ITC2_i08`	45	69	97
`comp09`	25	**22**	25	`ITC2_i09`	45	89	108
`comp10`	25	**18**	27	`ITC2_i10`	45	97	116
`comp11`	45	**27**	**27**	`ITC2_i11`	45	75	93
`comp12`	36	**22**	40	`ITC2_i12`	45	91	109
`comp13`	25	**17**	**22**	`ITC2_i13`	45	87	106
`comp14`	25	**17**	**23**	`ITC2_i14`	45	87	107
`comp15`	25	**22**	27	`ITC2_i15`	45	79	106
`comp16`	25	**18**	25	`ITC2_i16`	45	55	83
`comp17`	25	**17**	25	`ITC2_i17`	45	50	71
`comp18`	36	**14**	**32**	`ITC2_i18`	45	91	112
`comp19`	25	**23**	27	`ITC2_i19`	45	101	120
`comp20`	25	**19**	**23**	`ITC2_i20`	45	73	92
`comp21`	25	**23**	28	`ITC2_i21`	45	72	90
`erl.2011-2`	30	**22**	32	`ITC2_i22`	45	98	118
`erl.2012-1`	30	**14**	31	`ITC2_i23`	45	117	128
`erl.2012-2`	30	**20**	32	`ITC2_i24`	45	77	97
`erl.2013-1`	30	**16**	30	`toy`	20	**10**	**11**

instances, each period is available for each event. Values in bold face indicate the connectedness of clash-free timetables is established by Theorem 1.

Finally, we will show that for the instance `toy`, the proposed heuristic yields a vertex ordering that is a witness for $\text{subdeg}(F,G')$. Let G be the conflict graph for this instance and let G' be the graph that results from the reduction from list to graph coloring. In the CB-CTT formulation, the events are grouped into courses and for each course, events of the course are a clique in G and G'. Similarly, if two courses are in conflict, then the events of both courses are a clique in G and G'. If certain periods are unavailable for a course, then the events of the course and the periods are a clique in G'. In the `toy` instance, there are four courses which consist of 16 events in total. Figure 4 shows a succinct representation of an optimal vertex ordering of the graph G'. Each node of the shown graph is a clique, as noted below the nodes, and the cliques are ordered from left to right. Two nodes of the shown graph are connected if all nodes of the corresponding cliques are connected. The nodes T, A, S and G correspond to the courses labeled SceCosC, ArcTec, TecCos and Geotec, respectively. The node P_1 represents to the periods marked unavailable for course ArcTec and the node P_2 represents the periods unavailable for SceCosC. Any two conflicting courses are connected. Let $C = V(P_1) \cup V(P_2)$.

Let $\sigma \in S(G')$ such that the cliques are arranged in the order P_1, P_2, T, A, S, G with some arbitrary choice of the relative ordering of the vertices within each clique. This ordering is a

Table 2: The connectedness of the clash-free timetables for the Lewis/Paechter instances [18]. For each instance we give the degeneracy $\deg(G)$ of the conflict graph G. Values in bold face indicate the connectedness of clash-free timetables is established by Theorem 1.

instance	deg(G)	instance	deg(G)	instance	deg(G)
`small_1`	54	`med_1`	59	`big_1`	60
`small_2`	**41**	`med_2`	67	`big_2`	68
`small_3`	98	`med_3`	67	`big_3`	64
`small_4`	69	`med_4`	69	`big_4`	80
`small_5`	84	`med_5`	87	`big_5`	75
`small_6`	**24**	`med_6`	101	`big_6`	93
`small_7`	68	`med_7`	120	`big_7`	111
`small_8`	84	`med_8`	98	`big_8`	82
`small_9`	124	`med_9`	121	`big_9`	77
`small_10`	136	`med_10`	64	`big_10`	77
`small_11`	**34**	`med_11`	97	`big_11`	76
`small_12`	**22**	`med_12`	78	`big_12`	76
`small_13`	146	`med_13`	105	`big_13`	84
`small_14`	100	`med_14`	92	`big_14`	74
`small_15`	79	`med_15`	101	`big_15`	127
`small_16`	118	`med_16`	145	`big_16`	115
`small_17`	120	`med_17`	126	`big_17`	184
`small_18`	60	`med_18`	188	`big_18`	131
`small_19`	141	`med_19`	173	`big_19`	159
`small_20`	**28**	`med_20`	153	`big_20`	144

Table 3: The connectedness of the clash-free timetables for the Metaheuristic Network instances [24]. For each instance we give the degeneracy $\deg(G)$ of the conflict graph G. Values in bold face indicate the connectedness of clash-free timetables is established by Theorem 1.

instance	deg(G)	instance	deg(G)	instance	deg(G)
`easy01`	**15**	`medium01`	49	`hard01`	68
`easy02`	**19**	`medium02`	53	`hard02`	67
`easy03`	**13**	`medium03`	52		
`easy04`	**12**	`medium04`	51		
`easy05`	**20**	`medium05`	47		

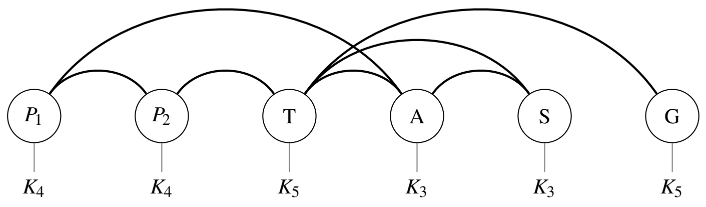


Fig. 4: Succinct representation of an optimal vertex ordering of the graph G' obtained from the conflict graph of the instance `toy` by the reduction to graph coloring. All nodes represent cliques as denoted below the nodes.

possible output of the algorithm VERTEXELIMINATION. From

$$\max_{v \in V(G')\setminus C} \operatorname{pred}(v,\sigma) = 11 \ ,$$

we can conclude that $\operatorname{subdeg}(F,G') \le 11$.

Proposition 1 *For the instance* `toy`*,* $\operatorname{subdeg}(F,G') = 11$.

Proof Let σ be an ordering of $V(G')$ and $V' \subseteq V(G') \setminus C$. The maximum number of predecessors adjacent to any vertex of V' in G' is denoted by

$$p(V',\sigma) = \max_{v \in V'} \operatorname{pred}(v,\sigma) \ .$$

Note that for a clique $K \in \{T,A,S,G\}$, the value $p(K,\sigma)$ is determined by the last vertex of K in σ. Thus, the value of $p(K,\sigma)$ depends only on the relative order of the last vertices of the cliques $\{T,A,S,G\}$ in σ. Let $\tilde{S}$ be the vertex orderings of G' such the vertices C precede all other vertices of G' and let $\hat{S}$ be the total orderings of $\{T,A,S,G\}$. For each ordering $\sigma' \in \hat{S}$ we can pick an ordering $\ell(\sigma')$ of G' that is compatible with σ' in the sense that the relative ordering of the last vertices of the cliques is in accordance with σ'. We have,

$$\operatorname{subdeg}(F,G') = \min_{\sigma \in \tilde{S}} \max_{K \in \{T,A,S,G\}} p(K,\sigma) = \min_{\sigma' \in \hat{S}} \max_{K \in \{T,A,S,G\}} p(K,\ell(\sigma')) \ .$$

For any ordering $\sigma' \in \hat{S}$ such that $G < T$, we have $p(T,\ell(\sigma')) \ge 13$, because the last vertex of T has at least 13 adjacent predecessors in G'. Thus, we only need to consider orderings such that $G > T$. Furthermore, since no vertex of G is adjacent to any vertex of A or S, changing the relative order of A and G or S and G does not change the number of adjacent predecessors. Hence, we can assume G is a maximum in any ordering of interest. We enumerate the values of $p(K,\ell(\sigma'))$ all for $K \in \{T,A,S,G\}$ for the 6 permutations of $\{T,A,S\}$:

clique ordering $\sigma' \in \hat{S}$	$p(T,\ell(\sigma'))$	$p(A,\ell(\sigma'))$	$p(S,\ell(\sigma'))$	$p(G,\ell(\sigma'))$
T,A,S,G	8	11	10	9
T,S,A,G	8	14	7	9
A,T,S,G	11	6	10	9
S,T,A,G	11	11	2	9
A,S,T,G	14	6	5	9
S,A,T,G	14	9	2	9

Thus,

$$\operatorname{subdeg}(F,G') = \min_{\sigma' \in \hat{S}} \max_{K \in \{T,A,S,G\}} p(K,\ell(\sigma')) = 11$$

We can conclude that the proposed heuristic produces a witness of $\operatorname{subdeg}(F,G') = 11$ on the instance `toy`.

5 Conclusions

We investigated the connectedness of clash-free timetables with respect to the Kempe-exchange operation. This investigation is related to the connectedness of the search space of timetabling problem instances, which is a desirable property, for example for two-step algorithms using the Kempe-exchange during the optimization step. We include period availability requirements in our analysis and derive improved conditions for the connectedness of clash-free timetables in this setting. We further show that the diameter of the reconfiguration graphs increases only linearly due to the period availability requirements. Our results indicate the connectedness of the clash-free timetables for a number of benchmark instances.

For future research, other properties of feasible timetables such as overlap-freeness may be considered as well. Furthermore, two kinds of possible improvements may be considered with respect to establishing the connectedness of clash-free timetables in the presence of period availability requirements: Both, a better analysis of Algorithm 1 and a better heuristic approach (or exact algorithm) for determining the subdegeneracy may lead to a lower number of periods required to certify the connectedness of clash-free timetables.

References

1. Brenda S. Baker and Edward G. Coffman, Jr. Mutual exclusion scheduling. *Theoretical Computer Science*, 162(2):225–243, 1996. doi:10.1016/0304-3975(96)00031-X.
2. Marthe Bonamy, Matthew Johnson, Ioannis Lignos, Viresh Patel, and Daniël Paulusma. On the diameter of reconfiguration graphs for vertex colourings. *Electronic Notes in Discrete Mathematics*, 38:161–166, 2011. doi:10.1016/j.endm.2011.09.028.
3. Marthe Bonamy, Matthew Johnson, Ioannis Lignos, Viresh Patel, and Daniël Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 247(1):1–12, 2014. doi:10.1007/s10878-012-9490-y.
4. Paul Bonsma. The complexity of rerouting shortest paths. In *Proc. 37th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 222–233, 2012. doi:10.1007/978-3-642-32589-2_22.
5. Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410: 5215–5226, 2009. doi:10.1016/j.tcs.2009.08.023.
6. Alex Bonutti, Fabio De Cesco, Luca Di Gaspero, and Andrea Schaerf. Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, and results. *Annals of Operations Research*, 194(1):59–70, 2012. doi:10.1007/s10479-010-0707-0.
7. Michael W. Carter. A comprehensive course timetabling and student scheduling system at the University of Waterloo. In *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling III (PATAT)*, pages 64–82, 2001. doi:10.1007/3-540-44629-X_5.
8. Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(5–6):913 – 919, 2008. doi:10.1016/j.disc.2007.07.028.
9. Dominique de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985. doi:10.1016/0377-2217(85)90167-5.

10. Dominique de Werra. Restricted coloring models for timetabling. *Discrete Mathematics*, 165–166:161–170, 1997. doi:10.1016/S0012-365X(96)00208-7.
11. Luca Di Gaspero and Andrea Schaerf. Curriculum-based course timetabling web-site. `http://satt.diegm.uniud.it/ctt/`. Accessed September, 2013.
12. Luca Di Gaspero, Barry McCollum, and Andrea Schaerf. The second international timetabling competition (ITC-2007): Curriculum-based Course Timetabling (Track 3). In *Proceedings of the 1st International Workshop on Scheduling, a Scheduling Competition (SSC)*, 2007. URL `http://pst.istc.cnr.it/RCRA07/articoli/P08-digaspero-etal-RCRA07.pdf`.
13. Parikshit Gopalan, Phokion G. Kolaitis, Elitza Maneva, and Christos H. Papadimitriou. The connectivity of Boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009. doi:10.1137/07070440X.
14. Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.
15. Marcin Kamiński, Paul Medvedev, and Martin Milanič. Shortest paths between shortest paths and independent sets. In *Proc. 21st Int. W'shop. on Combinatorial Algorithms (IWOCA)*, pages 56–67, 2011. doi:10.1007/978-3-642-19222-7_7.
16. Michel Las Vergnas and Henri Meyniel. Kempe classes and the Hadwiger conjecture. *Journal of Combinatorial Theory, Series B*, 31(1):95–104, 1981. doi:10.1016/S0095-8956(81)80014-7.
17. Rhydian Lewis. *Metaheuristics for University Course Timetabling*. PhD thesis, Napier University, Edinburgh, Scotland, 2006.
18. Rhydian Lewis and Ben Paechter. New "harder" instances for the university course timetabling problem. `http://www.soc.napier.ac.uk/~benp/centre/timetabling/harderinstances.htm`. Accessed September, 2013.
19. Zhipeng Lü and Jin-Kao Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010. doi:10.1016/j.ejor.2008.12.007.
20. Giorgio Lucarelli. *Scheduling in Computer and Communication Systems and Generalized Graph Coloring Problems*. PhD thesis, Athens University of Economics and Business, 2009.
21. David W. Matula. A min-max theorem for graphs with application to graph coloring. *SIAM Review*, 10(4):467–490, 1968. doi:10.1137/1010115.
22. Barry McCollum, Paul McMullan, Edmund K. Burke, and Rong Parkes, Andrew J.and Qu. The second International Timetabling Competition: Examination timetabling track. Technical Report QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, Queen's University, Belfast, Sep 2007. Available from `http://www.cs.qub.ac.uk/itc2007/`.
23. Bojan Mohar. Kempe equivalence of colorings. In Adrian Bondy, Jean Fonlupt, Jean-Luc Fouquet, Jean-Claude Fournier, and Jorge L. Ramírez Alfonsín, editors, *Graph Theory in Paris*, Trends in Mathematics, pages 287–297. Birkhäuser, 2007. doi:10.1007/978-3-7643-7400-6_22.
24. Olivia Rossi-Doria, Michael Sampels, Mauro Birattari, Marco Chiarandini, Marco Dorigo, Luca M. Gambardella, Joshua Knowles, Max Manfrin, Monaldo Mastrolilli, Ben Paechter, Luis Paquete, and Thomas Stützle. Supporting material for the paper [25]. `http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.htm`. Accessed February, 2014.

25. Olivia Rossi-Doria, Michael Sampels, Mauro Birattari, Marco Chiarandini, Marco Dorigo, Luca M. Gambardella, Joshua Knowles, Max Manfrin, Monaldo Mastrolilli, Ben Paechter, Luis Paquete, and Thomas Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. In Edmund Burke and Patrick Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 329–351. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-540-45157-0_22.
26. Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13 (2):87–127, 1999. doi:10.1023/A:1006576209967.
27. Katja Schimmelpfeng and Stefan Helber. Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum*, 29(4):783–803, 2007. doi:10.1007/s00291-006-0074-z.
28. George Szekeres and Herbert S. Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968. doi:10.1016/S0021-9800(68)80081-X.

FlexMatch - A Matching Algorithm with linear Time and Space Complexity

Nina Nöth · Peter Wilke

Abstract FlexMatch, a new matching algorithm with linear time and space complexity is introduced.

FlexMatch is based on a self organizing cell structure yielding next neighbour candidates for optimized matching.Results obtained when applying the FlexMatch algorithm on a real world problem are presented.

Keywords Matching Algorithm · Linear Time and Space Complexity · Self Organizing Cell Structure

1 Introduction

Our research group also acts as competence centre at our university regarding multi-criteria optimization problems. Recently we have been approached to implement a matching portal for students looking for hands-on training outside our faculty labs. The duration and extent depend on the degree and subject. As the metropolitan region of Nuremberg and especially the City of Erlangen is a national centre for medical engineering and technology numerous companies, ranging from very small business to major players, are offering this type of off-campus training.

The matching problem consists of a facts based part and a political component. Of course all companies would prefer to hire the best students, and of course most students would prefer a major company. But from a regional

Nina Noeth
E-mail: Nina.Noeth@Studium.Informatik.Uni-Erlangen.DE

Peter Wilke
University of Erlangen-Nuernberg
Computer Science Department
Multi Criteria Optimisation Group
Snail-mail: Martensstrasse 3, 91058 Erlangen, Germany
E-mail: Peter.Wilke@FAU.DE

developer's view students should get familiar with small but highly specialized companies even though they might not be widely known. And from an university's point of view all these training opportunities should be offered to all students.

2 The Problem

The problem to be solved consists of matching students with companies' training offers, where their constraints reflect the student's resp. companies' requirements.

It should be obvious that this optimization problems has contradictory constraints.

The initial situation consists of supply (companies' offers) and demand (students' requirements) which are both entered in a form with several categories. The values entered are either single numerical values, ranges or binary/boolean values.

In the context of this paper the concrete nature of the constraints or data is of no interest. It is sufficient to regard the data as a multidimensional vector and to presume the existence of a cost function to evaluate the current solution.

The solution should reflect the policy that students should be introduced to all kinds of companies and that companies should be offered students of all performance levels. But of course the individual requirements on both sides are the most significant matching criteria.

The solution consist of one company for each student and one student for each training opportunity offered. If this matching doesn't lead to a acceptance each party can request additional suggestions.

3 The FlexMap- and FlexMatch-Algorithms

On the top level abstraction layer solving the problem is divided into the following steps:

1. Building two separate ring structures: one connecting requirements and the other connecting requirements and offers having a preferable small distance to each other,
2. Matching the requirements and offer by using the resulting neighbourhood relation of the ring structures.
3. Improving the single matches by looking for better partners in a deeper neighbourhood.

3.1 FlexMap

FlexMap [Fritzke and Wilke(1991)] is a self-organizing neural network, which is linear in its time and space complexity. Problems similar to the Travelling

Salesman Problem can be solved using the growing ring structure yielding the round-trip we're looking for. I.e. the FlexMap algorithm connects each city with it's next – with respect to the length of the Hamiltonian cycle – neighbours, inducing some kind of a topology order on the nodes.

The basic idea is a growing cell structure. The initial structure consists of three cells. Repeated insertion and distribution steps extends the structure until all cells can be matched with its corresponding node, e.g. the city (Fig. 2). A node (e.g. a city) is chosen randomly and it next neighbour edge is calculated, a new cell is inserted in the middle of that edge and the cells are moved towards the chosen node (Fig. 1).

Table 1 shows the $O(n)$-version of the FlexMap algorithms in detail, followed by it's structogram.

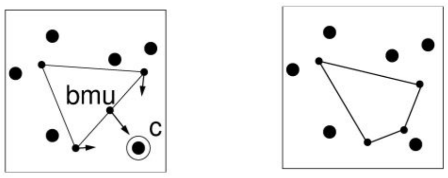


Fig. 1: Example of a distribution step: inserting a *bmu* in the neighbouring edge and moving three cells towards the node C. [Fritzke and Wilke(1991)]

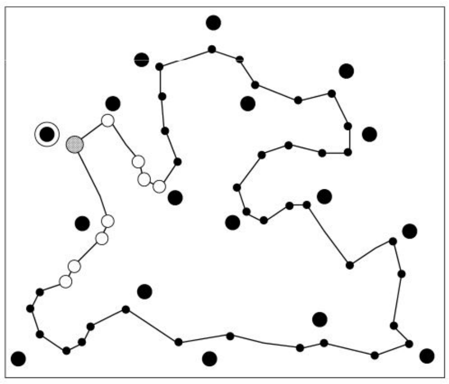


Fig. 2: Local search for the best matching unit: the previous *bmu* is shown as shaded circle and the local neighbours (here up to degree 4) as white circles. [Fritzke and Wilke(1991)]

Some remarks regarding it's complexity:

Step 1 $k_{neighbour}$ is a constant, so the neighbourhood search can be done in constant time $O(1)$.

Step 2 A cell becomes a member of the set of high error cells when is often becomes the bmu and therefore its error variable is increased quite often. As there are only $n_{distribution}$ steps, so the search can be performed in constant time $O(n)$. Obviously we can't guarantee to find the global maximum but most likely a cell with a high error value.

Step 3 All bookkeeping and pinning operations can be performed in constant time $O(1)$.

Steps 1 -3 The loop is performed n-times, all steps require only constant time, therefore the time complexity is linear to n, i.e. $O(n)$. The required space consists of memory space for the nodes and the cells. The maximum number of cells is equal to the number of nodes, i.e. $O(2*n) = O(n)$, which is linear in n too.

Step	Action
Init	Start with a ring structure consisting of three cells at randomly chosen positions. Initialize all error variables with 0.0.
Step 1	Perform a constant number $n_{distribution}$ of distribution steps. Update the error variable of the *bmu* (best matching unit) after every step. Keep a reference from each cell to the cell, which was most recently its bmu. When the cell is chosen again the search of its bmu is restricted to its last bmu and its neighbourhood up to $k_{neighbour}$ in each direction. $k_{neighbour}$ is a constant
Step 2	Find the edge e_{worst} in the set of high error cells connecting two cells v_1 and v_2, such that the *edgeerror* $$err(e_{worst}) := err(v_1) + err(v_2)$$ is a maximum for all edges in the ring structure. Insert a new cell c_{new} in the centre of e_{worst}. Initialize the error variable of the new cell with $$err(c_{new}) := \frac{1}{3}err(v_1)\frac{1}{3}err(v_2)$$ Adjust the error variables of v_1 and v_2 such that the total error of v_1, v_2 and c_{new} stays constant: $$err(v_1) := \frac{2}{3}err(v_1) + \frac{2}{3}err(v_2)$$
Step 3	If a cell has been bmu to a node *threshold* times the cell is pinned to that node and removed from the set of free cells. When every node has an associated cell, a solution is found. Otherwise continue with Step 1

Table 1: The $O(n)$ version of Flexmap [Fritzke and Wilke(1991)]

3.2 FlexMap Description

```
global variables:
    unripe      { MatchingObject -List, set of all elements, which
                should be part of the ring structure to be gener-
                ated}
    firstCell   { Cell -variable}
    tmp         { MatchingObject -variable}
constants:
    D_CONS=50
                { int -variable, specifying the number of distribu-
                tion steps}
    N_CONS=20
                { int -variable, specifying the number of neighbours,
                that should be examined while looking for a better
                bmu}
    E_BMU=0.05
                { double -variable}
initialize three random Cells
firstCell ← one of the Cells
unripe is not empty
    i ← 0
    i < D_CONS
        tmp ← random object of unripe
        tmp has already a bmu?
            true:
                search bmu for tmp by re-
                searching the N_CONS next
                and previous neighbours of
                the current bmu of tmp
            false:
                search the bmu in the whole
                structure of Cells
                ∅
        update the error of the new bmu by adding the distance
        between bmu and tmp
        update the position of the bmu and its two direct neigh-
        bours by moving them to the direction of tmp by the
        fraction E_BMU of the distance from each to tmp
        put the new bmu of tmp a list of the last used bmus
        has the new bmu of tmp a final object
            true:
                bmu_counter of
                tmp ← 0
                ∅
            false:
                was the current bmu of tmp
                the bmu for D_CONS
                times
                    true:
                        set tmp to the final ob-
                        ject of its bmu
                        remove tmp from unripe
                    false:
                        ∅
    find the edge with the largest error in the set of the D_CONS
    th last visited bmus
    insert a new Cell in the middle of the two vertices $v_1$ and $v_2$
    of the worst edge
    the error of the new Cell ← (error(($v_1$) + error($v_2$))*(1/3)
    the error of $v_1$ ← error($v_1$)*(2/3)
    the error of $v_2$ ← error($v_1$)*(2/3)
step over the Cell structure in one direction and take out all final
MatchingObject . Put them into a double linked list in the order of
removing them from the Cell structure
```

After the FlexMap algorithm is applied to the original matching problem we have two circular structures. One *req* linking the requirements and the other *all* linking all requirements and offers in a ring.

3.3 FlexMatch

The next step is to calculate the matching. Again we apply a growing cell structure algorithm, this time it's called FlexMatch[Nth(2014)].

Table 2 shows the $O(n)$-version of the FlexMatch algorithms in detail, followed by it's structogram. The structogram does not include step 5.

Step	Action
Step 1	Step over the circular structure *all* linking all objects. Take out all requirements having an offer as direct neighbour in the structure. and insert the pair of requirement and offer into a linked list *requested*. If one requirement has two direct neighboured offers take the one with the smaller distance.
Step 2	Step through the *requested* list and research if there are better offers for requirements. Take a pair of the list and check if the current offer of the requirement has an offer with smaller distance to the requirement in its direct neighbourhood. If this is the case swap the current offer with the better one.
Step 3	Extract the requirement of the first pair of the *requested* list as left restricting object and the requirement of the next pair in the list as right restricting object. Use only linking requirements to navigate through the list. Take the next requirement of the requirement list of the left restricting object.
Step 4	Test whether the euclidean distance to the offer of the left or of the right restricting object is smaller to the current requirement. Put the pair of the researched requirement and the smaller offer in a linked list of pairs *matchingList*. Take the next element of the current requirement of the requirement list. If this element is the right restricting object set the left restricting object to the right one and for the left one take the next element of the *requested* list. Continue with step 3. Otherwise continue with step 4. Go to step 5 when all elements of the requirement list have be processed.
Step 5	Take the structure *req* and choose one requirement *tmp* (is linked in both ring structures) . Research if the n_{better}[1] neighbours of *tmp* in both direction include a offer having a smaller distance to *tmp* then the current offer of *tmp*. If there is a better one exchange it with the current one of *tmp*. Take the next requirement of the structure *req* and repeat step 5 until you get to the first choosen requirement of step 5.

Table 2: The FlexMatch

[1] In the current implementation n_{better} hast the value 20.

3.4 FlexMatch Description

```
local variables:
    requirement
                 { MatchingObject -List}
    all          { MatchingObject -List}
    matchingList
                 { MatchingPair -List}
    requested
                 { MatchingObject -List}
    tmp          { MatchingObject }
$i \leftarrow 0$
$i <$ sizeof( all )
    $i \leftarrow i + 1$
    tmp is a requirement element?
    true
        tmp has a offer element as direct neighbour?
        true
            set the offer neighbour with the smaller distance to tmp to the offer of tmp
            put tmp into requested
        false
            ∅
    false
        ∅
    tmp $\leftarrow$ i -th element of all
$i \leftarrow 0$
$i <$ sizeof( requested )
    tmp $\leftarrow$ i -th element of requested
    has previous or next element of tmp a offer with a smaller distance to tmp
    true
        put the better offer to the offer of tmp and put them into the matchingList
    false
        put tmp and its offer into the matchingList
    $i \leftarrow i + 1$
$i \leftarrow 0$
tmp $\leftarrow$ the first element of requirement
requirement is not empty
    a $\leftarrow$ i -th element of requested
    b $\leftarrow$ i+1 -th element of requested
    the offer of a has a smaller distance to tmp
    true
        put the offer of a to the offer of tmp and put them into the matchingList
        ∅
    false
        put the offer of b to the offer of tmp and put them into the matchingList
    remove tmp from requirement
    tmp $\leftarrow$ the next element of tmp in requirement
    $i \leftarrow i + 1$
```

4 Difference to other matching algorithms

The most significant differences between the FlexMatch-Algorithm and non approximative matching algorithms is the cardinality of mapping and the runtime behaviour. The Gale-Shapley-Algorithm(GSA) [Gusfield and Irving(1989)] allows only a 1:1-mapping. This means every element of the entry set gets matched to one other element. The FlexMatch allows a 1:n-mapping. One element of the offering set can be proposed to one or more elements of the requesting set. The aim of algorithms like the GSA or the Hungarian Algorithm (HA) is the common weal. The FlexMatch however tries to find a solution having the priority of the individual satisfation for every element of the requesting set. The runtime complexity of the GSA is in $O(n^2)$ [Gusfield and Irving(1989), page 8] and of the HA it is $O(n^3)$ [James Munkres(1957)]. Some approximative matching algorithms[Vinkemeier and Hougardy(2005)] [Duan and Pettie(2010)] have a linear or nearly linear runtime, but they also do not permit 1:n mappings in their solutions.

5 Results

System

- CPU: Intel(R) Core(TM) i3-2350M CPU @ 2.30GHz
- Memory: 4GiB
- Compiler: Java version 1.7.0_25
- Runtime environment: OpenJDK Runtime Environment (IcedTea 2.3.10) (7u25-2.3.10-1ubuntu0.12.10.2)
- VM: OpenJDK 64-Bit Server VM (build 23.7-b01, mixed mode)

The test data sets were generated randomly. The single elements of a test set belong to two subsets having the same size. One is the set of requirements and one is the set of offers. The test sets have three different characteristics:

Randomness All elements of both sets are randomly distributed in a defined area

Independency The sets of the requirements and the offers have no overlapping area.

Overlapping The elements of the requirements lie inside the area of elements of the offers.

Fig. 3 shows how the characteristics of the sets would look like in the two dimensional space. The varying problem sizes were 100, 300, 500, 700, 1000, 3000, 4000, 5000, 6000, 8000 and 10000. For each size and characteristic 25 sets were generated. So there are 10*25*3 = 750 test sets. The FlexMatch was run five times for every set.

Fig. 4 shows the different runtime behaviour of the FlexMatch and of the brute force algorithm testing a element of a subset with all elements of the other subset. The computation of the runtime was done by taking the average

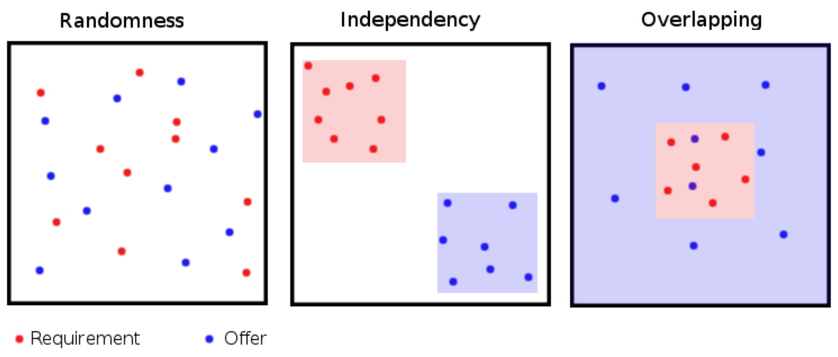


Fig. 3: Characteristics of the sets

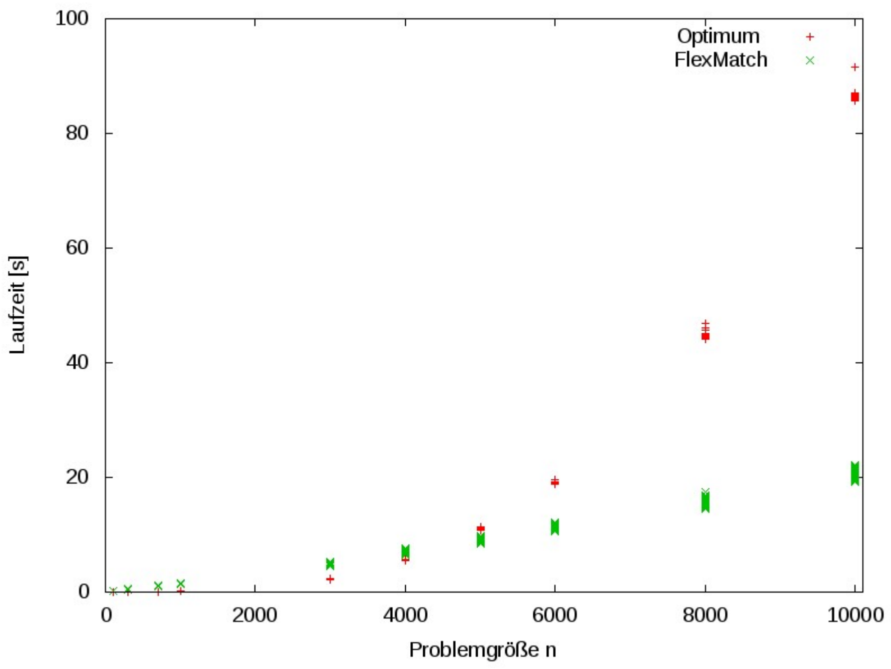


Fig. 4: Runtime behaviour

execution time per test set. The bottom line is that the FlexMatch algorithm shows a linear complexity and outperforms the brute force algorithm when the problem size exceeds the break even point at approx. 5000 elements.

Fig. 5 shows that most tests have a relative error smaller than 20%. The relative error $= (\frac{sumFM}{sumOpt} - 1) * 100\%$ was computed by summing up the distance between all matched pairs being found by the brute force algorithm (*sumOpt*) and the FlexMatch (*sumFM*).

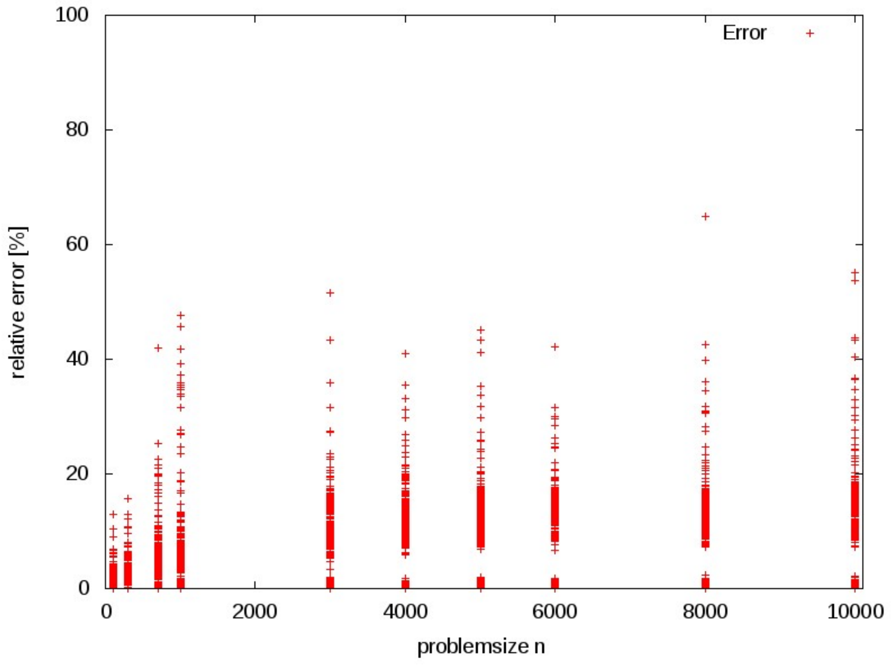


Fig. 5: Error statistics

Fig. 6 shows quintessential how the cell ring structure over all elements of the two sets generated by the FlexMap based part of the FlexMatch could behave in a two dimensional space with a problem size of 20. Fig. 7 and 8 represent the ring structures *req* linking all requirements and *all* linking all requirements and offers. Fig. 9 Shows the suggested matching pairs as a result of using the neighbourhood relation of the two ring structures.

6 Summary and outlook

In this paper we have shown how a matching algorithm with linear time and space complexity using a self organising map is designed and implemented. The matching algorithm yields results of sufficient quality but some artefacts have to be investigated as their costs are beyond the statistical expectations.

Currently we are running experiments using different data sets, initial situations and problem size and would like to improve the results and runtime behaviour further more.

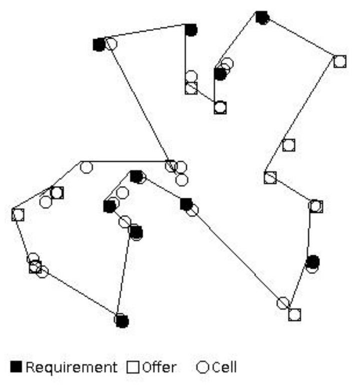


Fig. 6: Cell-Structure

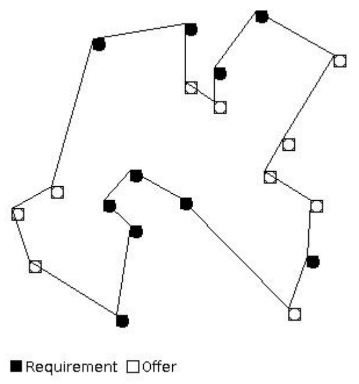


Fig. 7: Requirements and offers linked in a structure

The next steps would be the deployment of the software in the web-based matching portal and analysis of the real world problem behaviour.

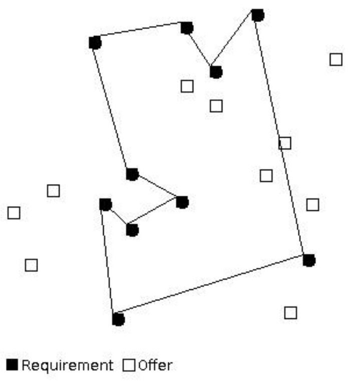


Fig. 8: Requirements linked in a structure

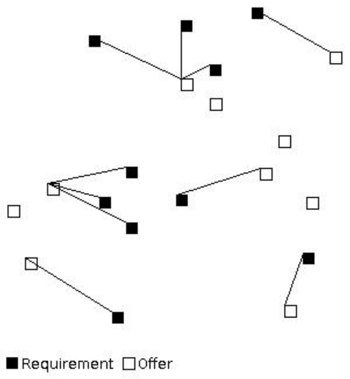


Fig. 9: Linked matchingpairs

References

[Duan and Pettie(2010)] Duan R, Pettie S (2010) Approximating maximum weight matching in near-linear time. In: Proceedings 51st IEEE Symposium on Foundations of Computer Science (FOCS), pp 673–682

[Fritzke and Wilke(1991)] Fritzke B, Wilke P (1991) FLEXMAP - A neural network with linear time and space complexity forthe travelling salesman problem. In: Proceedings IJCNN Int. Joint Conf. Neural Networks, Singapore

[Gusfield and Irving(1989)] Gusfield D, Irving RW (1989) The stable marriage problem: Structure and algorithms. Foundations of computing, MIT Press, Cambridge and Mass

[James Munkres(1957)] James Munkres (1957) Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics 5(1):32–38

[Noeth(2014)] Noeth N (2014) Optimierte Vergabe von Praktikumsplaetzen

[Vinkemeier and Hougardy(2005)] Vinkemeier DED, Hougardy S (2005) A linear-time approximation algorithm for weighted matchings in graphs. ACM TRANSACTIONS ON ALGORITHMS 1(1):107–122

Improvement by Combination

How to increase the Performance of Optimisation Algorithms by combining them

Johannes Ostler · **Peter Wilke**

Abstract Working with different optimisation algorithms leads to the observation that different types of solutions are generated, disclosing their different nature, their pros and cons. We investigated the question whether or not the combination of optimisation algorithms yield even better results than each of its constituents or will their drawbacks make things even worse.

Keywords Optimisation Algorithms · Combination of Algorithms · Performance Improvement

1 Introduction

Two runs of random based optimisation algorithms lead to different solutions, especially if different optimisation algorithms are used [Wilke and Ostler(2008)].There are different approaches to escape local minima like slow cooling in Simulated Annealing or jump to a worse solution like in Walk Dow Jump Up.We want to combine the good performance of neighbourhood based algorithms and the advantage of genetic operators.

Johannes Ostler
E-mail: Johannes.Ostler@informatik.uni-erlangen.de

Peter Wilke
University of Erlangen-Nuernberg
Computer Science Department
Multi Criteria Optimisation Group
Martensstrasse 3, 91058 Erlangen, Germany
E-mail: Peter.Wilke@fau.de

2 Basic Algorithms

2.1 Algorithms

The basic algorithms are plain vanilla implementations of Genetic Algorithm[Goldberg(2013)], Simulated Annealing[Kirkpatrick et al(1983)Kirkpatrick, Gelatt, and Vecchi,Preiss(1998)], Late Acceptance [Burke and Bykov(2008)] and Walk Down Jump Up [Wilke and Killer(2010)]. The used plain vanilla genetic algorithm implementation is corresponding to [Ho Pham L. Huy Anh and Nam(2011)].

All the basic algorithms are implemented for solving different timetabling problems defined in the REC model[Ostler and Wilke(2010)] of the EATTS [Gröbner et al(2003)Gröbner, Wilke, and Büttcher].

So the algorithms in the stand alone version are used for solving the problems. Several runs of the algorithms on the same problem data base provide the reference values for the combined algorithms.

For the reference runs we used a population with 10 or 20 individuals. The mutation rate was between 20 and 40 percent and between 0.5 and 5 percent of the not fixed resource lists of the mutated individuals were changed. The champion (i.e. the best individual) was excluded from mutation. The next generation consisted of the best and 5 or 10 other individuals. 40 % good individuals, which means cost better than average costs, and 60% random based individuals. To fill the population two point cross over was used. So 4 or rather 9 individuals were be created in every iteration

2.2 Data Base

2.2.1 University Exercise Group Planning

The Exercise Group Planning Problem assigns students to a list of exercise groups. The constraints are the limitation of the group size, the first and second choice selections made by the students and their wish to share the group with the best friends.

2.2.2 GAT 2011 courses at a university

Our university organises a girl-and-technology 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 1. Each student has a list of 4 preferred courses and can declare friends with whom she wants to share the same courses.

Table 1 Statistics of example problems

University Exercise Group Planning

Events:	25
Courses	25
Students	798
possible solutions	$\binom{798}{25}$

GAT 2011 example

Events:	229
TimeSlots:	57
Subjects:	52
Girls:	170
possible solutions	10^{1313}

3 Combining Algorithms

Preceding research [Wilke and Ostler(2008)] shows, that different random based runs mostly lead to different results, especially if different algorithms were used. Combining these different solutions by a Crossover operator could be a useful way for escaping local minima.

The essential idea is to combine the run of the Genetic Algorithm with other optimisation algorithms. For example Simulated Annealing can be used in the optimisation step or Walk Down Jump Up to generate decent individuals for the start population. The algorithms used in the optimisation step will optimise the solution by using random based decisions. And the Crossover operation over the different solutions will provide new solutions that are not in the neighbourhood of a local minimum.

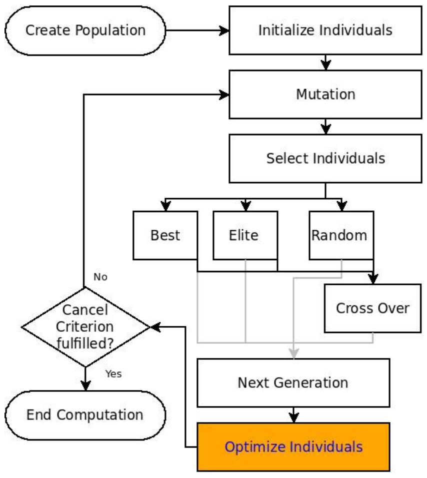


Fig. 1 Genetic Algorithm - modified implementation

For the test runs we used the same parameters as for the plain vanilla version. The interesting modification was an additional modification step. We optimised the new generation by short runs of different optimisation algorithms. We used Late Acceptance, Walk Down Jump Up, Simulated Annealing and Hill Climbing. In the optimisation step for every individual will be selected one of these algorithms by random. The runs ends after 2 seconds or 100000 iterations. The computation of a modified Genetic Algorithm runs 200 iterations.

The algorithm offers a good speed up when running the optimisation steps parallel. Because there are only synchronisation points needed after running the optimisation steps.

4 Results

4.1 Test Environment

The tests were running for every configuration 5 times. The best and worst case were dropped and the average value of the remaining results was computed. The combined algorithm was running in sequential mode. The speed up of running the optimisation steps parallel would distort the results.

4.2 Chart Types

4.2.1 Population Comparison Chart - PCC

The Population Comparison Chart PCC shows the quality change of some individuals and the best solution during the optimisation process. The vertical axis shows the costs, the horizontal axis the iteration. The costs are computed before the cancel criterion were checked meaning after the cross over or the optimisation step. For better scaling the first 10 iterations are dropped. If an individual dies it will be replaced by a child born in the cross over process. This chart provides a good look on changes of the different individuals of a genetic algorithm.

4.2.2 Genetic Step Chart - GSC

The Genetic Step Chart plots the population average costs after one specific step of the genetic algorithm in relation to the iteration. The vertical axis shows the costs, the horizontal axis the iteration. The specific steps are shown in table 2.

Table 2 Genetic Step Chart - GSC

mutate	the average costs after mutation
select	the average costs after selecting individuals (reduced population size)
cross over:	the average costs after cross over individuals (recovered population size)
optimise	the average costs after optimisation step
best	the costs of the best individual at the end of the iteration

The GSC shows the costs changes of the population after the single steps of the iteration. Some steps increase the costs like mutation, other will bring them down like the optimisation or selection.

4.2.3 Algorithm Comparison Chart - ACC

The Algorithm Comparison Chart ACC shows the comparison between different optimisation algorithms. For comparison issues a standardised base is need. The iteration counter is not suitable, because the computation time is quite different between one iteration of different algorithms. The only useful base for comparison seems to be execution time. But execution time depends also on external influences. So the tests must run several times and the average must be computed while dropping best and worst solution.

The ACC shows the best cost value on the vertical axis corresponding to the time value on the horizontal axis. For better scaling the costs axis the first seconds are dropped. The ACC contains one line per executed algorithm. With ACC different runs of the same algorithm with different parameter setting can also be compared.

4.3 Test and Results

4.3.1 Results EGP Problem

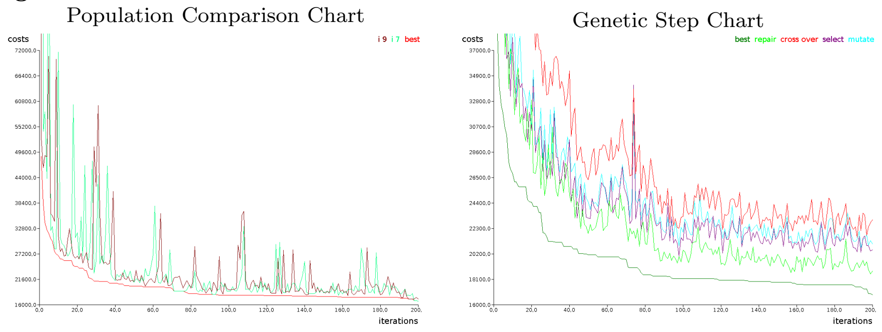
First we run the combined algorithm with a high mutation rate of 5 or 3 percent and the best individual was not mutated. The effect was that the best individual was improved by the different runs of optimisation algorithms, but there was no "genetic effect", which means that the other individuals which were mutated after every step have no chance to get best individual. So a sequential run of different optimisation algorithms on the best individual would have the same effects.

After setting the mutation rate to a half percent the results were different. The Population Comparison Chart 2 shows that after some iterations the best individual is reset by another individual. For better clearness we select two individuals and the best cost line. If the best cost line is shown in red colour a not selected individual is the best.

The Genetic Step Chart shows that the average cost increases after mutation step between 10 and 15 percent. Then the selection of some individual decrease the average costs a little bit. But the crossover operator increases the costs between 20 and 50 percent. The optimisation step decreases the costs so that the costs after optimisation are lower than after the mutation step. The most important insight by looking on the GSC is that the average costs decreases.

The Algorithm Comparing Chart shows the comparison of Walk Down Jump Up, Simulated Annealing and Late Acceptance and the combined genetic algorithm in comparison. After 320 seconds Simulated Annealing offers

Fig. 2 EGP Problem



the best solution, but after 1500 seconds the best individual of the combined algorithm is better. Simulated Annealing stagnates at this time, the combined is still improving. Late Acceptance offers a good solution after 2240 seconds but is also stagnating after this.

So after 3200 seconds the combined algorithm leads to the best solution with 16639 penalty points the results are following (table 3):

4.3.2 GAT Problem

The Algorithm Comparison Chart for the Girls and Techniques Problem shows the problem of neighbourhood bases algorithms: stagnation in local minima. This time ranking of the reference algorithm is inverted. Walk Down Jump Up leads to the best result. But this solution has with 7237 over the double costs of the combined algorithm, which solution has 3519 penalty points.

Table 3 Comparing Results EGP and GAT Problem

	EGP problem		GAT Problem	
Algorithms	Costs	Diff to Combined	Costs	Diff to Combined
Late Acceptance	17334	+4.2 %	7237	+106 %
Simulated Annealing	18808	+13.0 %	7507	+113 %
Walk Down Jump Up	20676	+24.2 %	10795	+207 %

5 Summary

So over all the combined algorithm could be a good alternative to neighbourhood search based algorithms. Like other genetic algorithms the usage of main storage for n individuals is n times higher than the usage of neighbourhood search based algorithms.

But if enough time and storage could be spend the combined algorithm should lead to good solutions. If a multi core or distributed hardware can be used there is good speed up while running the optimisation step parallel.

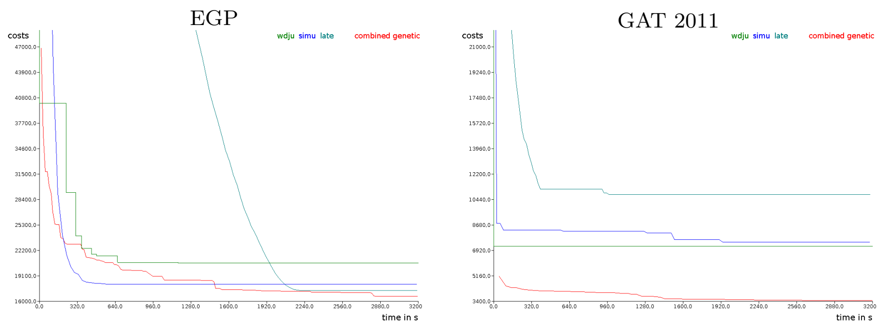


Fig. 3 Algorithm Comparison Chart of the Problems

6 Outlook

The setting of the parameters, especially of the mutation rate has big influence to the performance of the algorithm. So an automated parameter setting could be helpful. Especially the adaptation of run time and selection probability of the neighbourhood based algorithms corresponding to their average cost improvement could have positive effects to runtime and solution quality.

References

[Burke and Bykov(2008)] Burke EK, Bykov Y (2008) A late acceptance strategy in hill-climbing for exam timetabling problems. In: Proceeding of the 7th international conference on the Practice and Theory of Automated Timetabling, Patat2008, URL http://w1.cirrelt.ca/∼ patat2008/PATAT_7_PROCEEDINGS/Papers/Post-WD2a.pdf

[Goldberg(2013)] Goldberg DE (2013) Genetic Algorithms. Pearson Education, URL http://books.google.de/books?id=6gzS07Sv9hoC

[Gröbner et al(2003)Gröbner, Wilke, and Büttcher] Gröbner M, Wilke P, Büttcher S (2003) A standard framework for timetabling problems. In: Practice and Theory of AutomatedTimetabling IV, Springer Berlin / Heidelberg, Lecture Notes in Computer Science, vol 2740, pp 24–38, DOI 10.1007/b11828, URL http://www.springerlink.com/content/2mj0rwlpbp5uvh1v/

[Ho Pham L. Huy Anh and Nam(2011)] Ho Pham L Huy Anh KKA, Nam NT (2011) Modeling identification of the nonlinear robot arm system using miso narx fuzzy model and genetic algorithm. In: Robot Arms, InTech, pp 1–10, URL http://http://openaccessbooks.org/book/robot-arms

[Kirkpatrick et al(1983)Kirkpatrick, Gelatt, and Vecchi] Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671 – 680

[Ostler and Wilke(2010)] Ostler J, Wilke P (2010) The erlangen advanced timetabling system (eatts) unified xml file format for the specification of timetabling systems. In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, Patat2010 - Queens University Belfast, pp 447–464, URL http://www.cs.qub.ac.uk/∼ b.mccollum/patat10/Proceedings_patat10.pdf

[Preiss(1998)] Preiss BR (1998) Data Structures and Algorithms with Object-Oriented Design Patterns in Java, http://www.brpreiss.com/books/opus5/html/page474.html, chap Simulated Annealing, p 474

[Wilke and Killer(2010)] Wilke P, Killer H (2010) Walk down jump up - a new hybrid algorithm for time tabling problems. In: Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, Patat2010 - Queens University Belfast, pp 440–446, URL http://www.cs.qub.ac.uk/∼ b.mccollum/patat10/Proceedings_patat10.pdf

[Wilke and Ostler(2008)] Wilke P, Ostler J (2008) Solving the School Time Tabling Problem Using Tabu Search, Simulated Annealing, Genetic and Branch & Bound Algorithms. In: Burke E, Gendreau M (eds) PATAT '08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, URL http://www5.informatik.uni-erlangen.de/Forschung/Publikationen/2008/Wilke08-STS.pdf

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Integer Programming for Minimal Perturbation Problems in University Course Timetabling

Antony E. Phillips · Cameron G. Walker · Matthias Ehrgott · David M. Ryan

Abstract In this paper we present a general integer programming-based approach for the minimal perturbation problem in university course timetabling. This problem arises when an existing timetable contains hard constraint violations, or infeasibilities, which need to be resolved. The objective is to resolve these infeasibilities while minimising the disruption or perturbation to the remainder of the timetable. This situation commonly occurs in practical timetabling, for example when there are unexpected changes to course enrolments or available rooms.

Our method attempts to resolve each infeasibility in the smallest neighbourhood possible, and utilises the exactness of integer programming. Operating within a neighbourhood of minimal size keeps the computations fast, and does not permit large movements of course events, which cause widespread disruption to timetable structure. We demonstrate the application of this method using an example based on real data from the University of Auckland.

Keywords University Course Timetabling · Integer Programming · Decision Support Systems

1 Introduction

University course timetabling is a well-known problem in which a time period and a room are determined for each course event (e.g. a lecture). This may be conducted prior to the start of enrolment, or after enrolment data is known. The former case is referred to as *curriculum-based timetabling*, because time clashes between courses are determined by sets of courses known as *curricula*. The latter case is referred to

This research has been partially supported by the European Union Seventh Framework Programme (FP7-PEOPLE-2009-IRSES) under grant agreement #246647 and by the New Zealand Government as part of the OptALI project.

A. E. Phillips, C. G. Walker, D. M. Ryan
Department of Engineering Science, The University of Auckland
E-mail: antony.phillips@auckland.ac.nz

M. Ehrgott
Department of Management Science, Lancaster University

as *enrolment-based timetabling* because clashes can be determined and weighted by known enrolments for each course.

In a practical setting, both of these problems are applicable to some extent (Kingston, 2013a). The timetable is typically constructed significantly prior to the start of enrolments, and it will commonly need to be modified as enrolments take place. During each of these phases, the situation can arise where an existing timetable becomes infeasible due to changes in the underlying data. The minimal perturbation problem addresses how to modify an existing timetable so that feasibility is found with a minimal amount of perturbation (or disruption) to the structure of the timetable.

The minimal perturbation problem is first comprehensively addressed in the context of general dynamic scheduling (El Sakkout, Richards, and Wallace, 1998; El Sakkout and Wallace, 2000). El Sakkout and Wallace (2000) propose an algorithm based on constraint programming techniques, which leverages the efficiency of linear programming to solve part of the problem.

Barták, Müller, and Rudová (2004) are the first to study minimal perturbation problems in the context of university course timetabling, using a constraint satisfaction heuristic combined with a branch-and-bound process. The authors continue this work with a local search-based metaheuristic, known as "iterative forward search", which improves performance significantly (Müller, Rudová, and Barták, 2005). Finally, Rudová, Müller, and Murray (2011) present a summary of this approach as part of a broader course timetabling process, which is implemented at Purdue University, USA. This includes detailed results on the iterative forward search algorithm as applied to minimal perturbation problems, and is described in a practical setting.

Kingston (2013b) addresses a similar problem in the context of high school timetabling, proposing an ejection chain heuristic method.

In this paper we present a new general method for solving minimal perturbation problems which arise in practical timetabling. Around each infeasibility, we define a small *neighbourhood* of events, time periods, and rooms, which we are willing to perturb. Within this neighbourhood we solve an integer programme which attempts to resolve the infeasibility with minimal disruption to existing timetable structure. Utilising the exactness of integer programming, we will only expand the size or scope of the neighbourhood when we have certainty that the current neighbourhood is insufficient to resolve the infeasibility. This process aims to ensure the computational tractability of each integer programme. Limiting the neighbourhood size is also desirable because it prevents large movements of course events, which are seen as disruptive to the timetable structure.

We demonstrate the application of this method using an example based on real data from the University of Auckland. The expanding neighbourhood methodology has been successfully demonstrated in other real-world applications (Rezanova and Ryan, 2010).

2 Minimal Perturbation Problems in University Course Timetabling

A complete solution to the university course timetabling problem is a timetable which specifies a time period and room for every course event. The solution can be considered feasible if it does not include any violated hard constraints, or *infeasibilities*. Quality measures, or soft constraints, are desirable features of a feasible solution which may also be considered. For a coverage of commonly used hard and soft constraints we refer to the benchmarking paper by Bonutti, De Cesco, Di Gaspero, and Schaerf (2012).

University course timetabling is widely accepted to be a dynamic problem in practice, where data may continually change throughout construction and implementation of a timetable (McCollum, 2007; Kingston, 2013a). For complex timetabling at large universities, we broadly discuss how minimal perturbation problems can arise in each stage of the timetabling process. We draw on our own experiences at the University of Auckland, which bears many similarities to other large universities considered in the timetabling literature.

The early construction phase of timetabling occurs when most of the data has been gathered, and construction of a timetable is starting. Most time or room assignments are considered tentative, and may be changed relatively freely. At this stage, almost any changes to the data are possible e.g. new or removed courses, changes to staff employment status, room availabilities etc. Some infeasibilities may not need to be resolved until the data is more complete.

The late construction phase occurs when the timetable is close to being finalised for publication. This stage is the most similar to the curriculum-based timetabling problem addressed in the literature. At the University of Auckland, time assignments are determined in close collaboration with faculties, to satisfy their specific and complex requirements. In this case, changing the time period for an event would be disruptive, whereas the room assignment may be more freely perturbed. Major changes to the data are less likely at this stage, and all infeasibilities should be resolved. We note that infeasibilities may also arise due to the method of constructing a timetable, rather than solely due to changes in the data. For example, if faculties choose their own time-assignments independently (often "rolled-forward" from the previous year with changes), this can produce a timetable for which there is no feasible room assignment.

The enrolment phase of timetabling begins once the timetable is published, and students have started to select courses. At this stage, the most common changes to the data involve adjusting the expected course enrolments to actual course enrolments, which may cause some room assignments to be no longer suitable. In this phase, it can be disruptive to change either the time period or the room assignment for an event. However, the former is likely to be particularly disruptive, as students and staff may have external obligations affecting their personal timetable. We note that in the case of the University of Auckland, because of legal requirements it is not possible to have any excess or overflow of students in a room. Although not all events will be attended by every enrolled student (e.g. sickness, retention, recorded lectures), the first events of a semester are typically well attended.

In practice, the specific timing and cause of an infeasibility can significantly affect the choice of which perturbations we are willing to perform. The algorithm we present in this paper was motivated by two particular practical situations. The first arises in the late construction phase of timetabling, where faculties have approved a high quality time assignment for all their events, but we require perturbation in order for a feasible room assignment to exist. Secondly, we address the ubiquitous problem in the enrolment phase of timetabling, where unpredictable course enrolments have caused existing room assignments to become unsuitable.

3 Room Assignment

This section gives a brief overview of our room assignment algorithm. As mentioned in Section 2, many minimal perturbation problems in the construction phase and particularly the enrolment phase of timetabling, involve an infeasible room assignment. Although the room assignment algorithm is not the focus of this paper, attempting to solve a room assignment for an infeasible timetable will reveal important information about the nature of the infeasibilities. For a detailed coverage of the room assignment process, we refer to Phillips, Waterer, Ehrgott, and Ryan (2014).

The relationship between the room assignment and timetable perturbation process is shown in Figure 1. Each rectangular box corresponds to solving an integer programme to maximise the room assignment quality with respect to a particular objective. We maximise each objective sequentially, and fix its value in subsequent iterations, known as *lexicographic optimisation* (Ehrgott, 2005). A lexicographic approach implies a strict precedence of the importance of quality measures, which is appropriate in this case.

Firstly we would like to maximise the number of events which are assigned to a feasible room, or *event hours*. Due to the exactness of integer programming, if we do not find a room for all events, we can be certain that no such complete room assignment exists. This means we will need to perform a timetable perturbation in order to find a feasible room assignment.

However, we would first like to improve the value of other objectives. We next maximise the *seated student hours* which aims to assign larger events to a room, in preference to smaller events. This means the unassigned events will be of the smallest size possible. Thirdly, we maximise the *seat utilisation* where events are placed in rooms which “fit” well, i.e. most of the seats are occupied. This means that any rooms which are left unused (in each time period) will be the largest possible. Finally we maximise the *building preference* which favours holding events in rooms which are geographically close to the associated teaching department.

The resulting partial room assignment provides us with essential information. Firstly, it shows which time periods contain unassigned events (infeasibilities), and therefore require solving the minimal perturbation problem. Secondly, the partial room assignment contains information which helps to focus our neighbourhood search to restore feasibility (see Section 4.4).

4 Minimal Perturbation

4.1 Expanding Neighbourhood Algorithm

Solving the minimal perturbation problem requires assigning both a time period and a room for each event, rather than handling these problems separately. However, building a model with variables indexed over all events, time periods and rooms could easily result in millions of variables (Burke, Mareček, Parkes, and Rudová, 2008), which would be intractable. As a result, we would like to build a model which resolves each infeasibility in as small a *neighbourhood* as possible. The neighbourhood around an infeasibility is defined by a restricted set of events which can be moved, and subsets of time periods and rooms to which events can

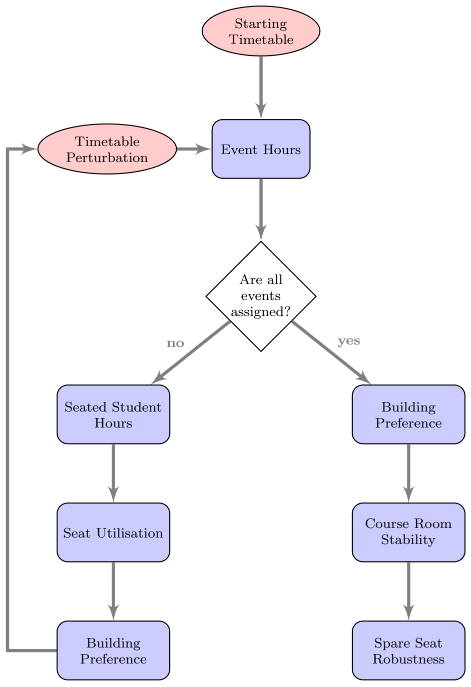


Fig. 1 University of Auckland Timetabling Process 2010

be moved. All events outside this neighbourhood are *fixed*. Based on information from the partial room assignment, the specific composition of a neighbourhood is heavily customised to the nature of the infeasibility.

In many universities, it is common for rooms to be utilised in approximately 50% of available time periods (Beyrouthy, Burke, Landa-Silva, McCollum, McMullan, and Parkes, 2007). Therefore, it is very likely that a feasible timetable will exist where all events are held in a suitable time period and room. Furthermore, whether the infeasibility arises from rolling forward an old timetable with changes, or if there are unexpected post-enrolment changes, it is likely that the infeasible timetable will be "close" to feasibility, i.e. only a small number of events will need to change time period or room.

We will initially generate a very small neighbourhood, where we are only willing to move events of a similar size to our unassigned event(s), and only to/from the time periods one hour before and after their current time (for example). Within this neighbourhood we solve an integer programme (IP) to reassign all neighbourhood events in the least disruptive way that removes the infeasibility. If this is not possible, we will expand the scope of the neighbourhood, and solve again until the infeasibility can be resolved. Although several iterations of this process may be required, each IP model will be relatively small due to the optimistic methodology of starting with a small neighbourhood. Because we use an exact algorithm to attempt resolving the infeasibility, we know with certainty whether a feasible reassignment exists in this neighbourhood or not. This is an advantage over using a heuristic method, where it can be difficult to determine whether a given problem is infeasible or whether this heuristic is unable to find a solution.

When we need to expand the neighbourhood, this is done in a similar way to how the starting neighbourhood is constructed, in the sense that the expansion rules will prioritise adding the most promising variables or options first. Once we have found a feasible timetable and room assignment within this neighbourhood, we can apply this solution to the timetable, and proceed to resolve any other infeasibilities. This algorithm is presented as Algorithm 1.

Algorithm 1 Expanding Neighbourhood Algorithm

```
for all Infeasible Time Periods do
    $N \leftarrow$ GenerateInitialNeighbourhood()
    $searching \leftarrow$ True
    while $searching$ do
        $IP \leftarrow$ BuildNeighbourhoodIP(N)
        $IP$.Solve()
        if IP is Feasible then
            $Timetable$.Update()
            $searching \leftarrow$ False
        else
            $N$.Expand()
        end if
    end while
end for
```

Finally, it is worth recalling that we are not only looking for a feasible solution; we are also aiming to minimise the timetable disruption. Although the latter is our objective function within the neighbourhood, in some cases a feasible timetable and room assignment can be found within a given neighbourhood, but only through significant disruption to the underlying timetable. In this case, it may be possible to expand the neighbourhood further, and find a solution with a lower disruption. This is desirable, if it is computationally feasible to explore the larger neighbourhood size.

4.2 Event-based Neighbourhood IP

Within a given neighbourhood, we solve an integer programme to attempt to find a feasible and low disruption assignment of events to time periods and rooms. As mentioned in Section 4.1, significant attention is paid to the scope of the neighbourhood, in terms of which (re)assignments we will consider. All sets defining a neighbourhood are listed in Table 1, where variables are generated over the sets E, T_e and R_{et}. Many of the complex constraints relating to either the time or room assignments (e.g. clashes between courses, staff requirements etc.), are implicitly represented by the variable generation.

Control over the neighbourhood sets offers significant modelling power. In particular, for many courses it is required that a maximum of one event is held per day. If a *fixed* event (i.e. a 'non-neighbourhood' event) from a course c is held on day d, then time periods on this day will be excluded from T_e for any other events of this course $e \in E_c$. Similarly, a *curriculum* is a group of courses which cannot be held in the same time period (as they are taken by a common group of students). If an event from curriculum *curr* is fixed in a time period within the neighbourhood, then no events from courses in this curriculum can be moved to this time period. If we are solving a problem in the enrolment phase of timetabling, the curricula can be determined by actual student enrolments. It is important that no enrolled student has a timetable which becomes infeasible after the minimal perturbation problem is solved.

Timetable elements which span multiple consecutive time periods (e.g. two-hour lectures) are generalised as sets of events known as *long events*. If a long event lies only partially in the neighbourhood, it is not permitted to be moved to ensure the time stability. In many situations, it is also required to ensure room stability across all time periods of a long event. In this case, if the long event is partially in the neighbourhood and already has a room, it is excluded. If it is partially in the neighbourhood but does not have a room assigned, the neighbourhood must expand to include the full long event. In our model we have assumed that both time and room stability are required.

E	events in neighbourhood	T_e	time periods suitable for event e
C	courses	T	time periods in neighbourhood
CU	curricula	D	days of neighbourhood time periods
E_c	events of course c	T_d	time periods on day d
E_{curr}	events of courses in curriculum $curr$	R_{et}	rooms suitable for event e and available in time period t
E_F	events which are single-period or the first of a long event		
E_{tr}	events suitable for assignment to time period t and room r		

Table 1 Notation for Neighbourhood Sets

Using the notation defined in Table 1, we present an integer programming formulation of an *event-based* neighbourhood perturbation model. In this formulation, the binary variables x_{etr} are indexed by suitable event-time-room assignments. Specifically, let the variable x_{etr} take the value 1 if event $e \in E$ is to be held at time $t \in T_e$ in room $r \in R_{et}$. For a given weighting of penalties, v_{etr}, an optimal assignment of events to time periods and rooms can be determined by solving the following integer programme (1)–(7). The formulation is considered event-based because the penalty for reassigning an event is independent of the other neighbourhood events from the same course. The disruption penalties for an event can vary depending on the number of time periods it moves, whether the room changes, and how this relates to any fixed events from this course. With sufficient available data, precise penalties can be specified for each disruption.

$$\text{minimise} \quad \sum_{e \in E} \sum_{t \in T_e} \sum_{r \in R_{et}} v_{etr} * x_{etr} \tag{1}$$

$$\text{subject to:} \quad \sum_{e \in E_{tr}} x_{etr} \leq 1 \qquad t \in T, r \in R_t \tag{2}$$

$$\sum_{t \in T_e} \sum_{r \in R_{et}} x_{etr} = 1 \qquad e \in E \tag{3}$$

$$\sum_{\substack{e \in \\ (E_c \cap E_F)}} \sum_{\substack{t \in \\ (T_e \cap T_d)}} \sum_{r \in R_{et}} x_{etr} \leq 1 \qquad c \in C, d \in D \tag{4}$$

$$\sum_{e \in E_{curr}} \sum_{r \in R_{et}} x_{etr} \leq 1 \qquad curr \in CU, t \in T \tag{5}$$

$$x_{etr} - x_{(e-1)(t-1)r} = 0 \qquad e \in (E \setminus E_F), t \in T_e, r \in R_{et} \tag{6}$$

$$x_{etr} \in \{0, 1\} \qquad e \in E, t \in T_e, r \in R_{et} \tag{7}$$

The objective function (1) minimises the total timetable *disruption* between the proposed timetable solution, and the initial (infeasible) timetable.

Constraints (2) ensure that each room in each time period is occupied by a maximum of one event, while constraints (3) ensure that all events are assigned to exactly one room in one time period. Constraints (4) ensure that two events from the same course cannot move to any time period on the same day. Because long events are represented as more than one individual event $e \in E$, only the first event e in any long event is included in each constraint. Constraints (5) ensure that two events from the same curriculum cannot move to the same time period. Lastly, constraints (6) enforce the strict time stability and room stability on the events of a long event.

If room stability is not required for a long event (i.e. it is acceptable to change rooms between the individual event-hours), constraints (6) can be altered such that each constraint is summed over all suitable rooms, rather than applied as one constraint per room.

4.3 Course-based Neighbourhood IP

Although the event-based formulation is versatile in terms of representing penalties, it is not able to model *time stability* for a course. This is a common quality measure where it is considered desirable to schedule all weekly events from a course at the same time of day. To build upon the simpler event-based formulation (1)–(7), we define the following: C_{stab} is the set of courses which desire time stability, H is the set of hours from all neighbourhood time periods, c_e is the course with which event e is associated, and h_t is the hour of the day for time period t. Let the variable y_{ch} take the value 1 if *any* event of course c is held in hour h in the timetable.

For a given weighting of time stability penalties, w_{ch}, we can solve the following course-based integer programme.

$$\begin{aligned}
\text{minimise} \quad & (1) + \sum_{c \in C} \sum_{h \in H} w_{ch} * y_{ch} && && (8)\\
\text{subject to:} \quad & (2)\text{–}(7) && && \\
& x_{etr} - y_{c_e h_t} \leq 0 && e \in E, t \in T_e, r \in R_{et} && (9)\\
& y_{ch} \in \{0,1\} && c \in C_{stab}, h \in H && (10)
\end{aligned}$$

The additional term in the objective function (8) penalises each course for each unique hour of the day it uses for any of its events. Constraints (9) appropriately tie the values of the y_{ch} variables to the x_{etr} variables.

This formulation requires a different starting neighbourhood and set of expansion rules to the event-based model. Because entire courses are required to move together, the initial neighbourhood should not focus around a single infeasible time period, but rather all weekly infeasible time periods for an hour of the day.

4.4 Limiting the Neighbourhood

As mentioned in Section 4.1, it is desirable to keep the neighbourhood as small as possible, as determined by the sizes of the sets in Table 1. This necessitates a selective definition of the neighbourhood at each stage of expansion to include the most promising variables (i.e. event-time-room allocations) in the model.

The initial definition and expansion rules for a neighbourhood are guided by the existing partial room assignment, which can offer substantial insight into the specific cause of the infeasibility. By maximising the *seated student hours*, we ensure that any unassigned events will be as small as possible. Therefore, if we observe that large events remain unassigned, we can infer that the associated time periods are in shortage of large rooms. As a result, when expanding the neighbourhood we know to focus on events which currently occupy large rooms, and ideally we can expand into time periods with vacant large rooms. Without maximising the seated student hours, unassigned large events could be due to a general lack of rooms of any size.

The room assignment process also maximises the *seat utilisation*, where it is favourable to assign events to rooms which are closely matched in size. This optimisation is important, particularly for time periods which do not contain an unassigned event themselves, but are adjacent or near to time periods with unassigned events. In the case of a full room assignment for a particular time period, the previous maximisations (of event hours and seated student hours) will permit assigning small events in larger rooms than necessary, as long as it is still possible to assign all events. Maximising the seat utilisation has two useful outcomes for the neighbourhood search. Firstly, where possible, it will leave larger rooms vacant as they are more flexible. Secondly, by assigning events into closely fitting rooms, the neighbourhood expansion can delay incorporating an event and room assignment which "fit" well, since the room is already well utilised. The last quality measure in the room assignment is the *building preference*, which can also be useful when prioritising which event-to-room assignments should enter the neighbourhood first.

Finally, we note that for some practical situations, infeasibilities may be caused by a shortage of a particular type of room (e.g. computer laboratories) or room attribute (e.g. piano, fume cupboard), rather than a specific size of room. In this situation, the neighbourhood definition will include rooms from other time periods with this specific feature, even if they are presently occupied by a closely fitting event. The room assignment process may even be altered to include a quality measure which favours assigning events with specific room requirements. This helps identify which room features are in shortage, and is analogous to maximising seated student hours to identify the critical room sizes.

5 Practical Example

To demonstrate our algorithm, we consider a scenario based on the University of Auckland's timetabling data from Semester 2 in 2010. The University of Auckland timetable features 2131 events, 72 rooms, and 50 weekly time periods (8am to 6pm, from Monday to Friday). Like many universities, the average room is utilised in approximately 60% of time periods. However, the utilisation is above 80% in "peak" time periods, which are typically between 10am and 3pm. The utilisation of the largest rooms is also above average, due to a long-term increase in student enrolments.

Computational experiments are run using Gurobi 5.0 running on 32-bit Ubuntu 12.04, with a quad-core 3.33GHz processor (Intel i5-660).

5.1 Over-enrolment Example

In this example, we analyse the problem of unexpectedly high enrolment numbers in the enrolment phase of timetabling. We assume that the revised enrolment numbers are received prior to the start of semester, but clearly after the timetable is originally constructed. As a result, it is considered disruptive to make changes to the time at which events are held, although changing room assignments is more acceptable. Specific penalties are given for each of the approaches in Sections 5.2 and 5.3.

The example scenario is given in Table 2, where the enrolment numbers for an introductory sociology and law course are substantially larger than expected. The courses have two and three events respectively, in the time periods listed. These are the type of courses which are likely to have an unpredictable enrolment, as they are available to new-entrant students and may be taken as electives by students from many academic programmes. In cases when enrolments are only marginally larger than expected, ideally the existing rooms will still be suitable. The room assignment process (Figure 1) maximises the *spare seat robustness* objective, for this purpose.

Course Name	Time Periods	Planned Enrolment	Revised Enrolment
SOCIO 100	Mon 12pm, Thu 2pm	320	500
LAW 121G	Mon 12pm, Wed 12pm, Fri 12pm	269	500

Table 2 Scenario changes to course enrolments

Initially, the state of having five events (from two courses) with no assigned room is identified as the infeasibility in the timetable. In order to resolve this infeasibility, we will first re-solve the room assignment for this timetable, as explained in Section 3. This will only change the room assignment for a very small number of events, which are held in the same time periods as the infeasibilities. If there are suitable vacant rooms in these time periods, or if it is possible to re-assign other events to "free up" such rooms, we will have a feasible room assignment without needing to perturb the time assignments at all. For this problem, maximising the event hours proves it is not possible to leave fewer than five events unassigned, without perturbing the timetable. This is because the events are held in highly congested time periods where all large rooms are presently occupied. However, by maximising the seated student hours, we are able to change our partial room assignment so that the five unassigned events are those which have the smallest number of students. The five events which remain unassigned are given in Table 3, where it can be seen that three of the events from the "SOCIO 100" and "LAW 121G" courses have been replaced by smaller events from other courses.

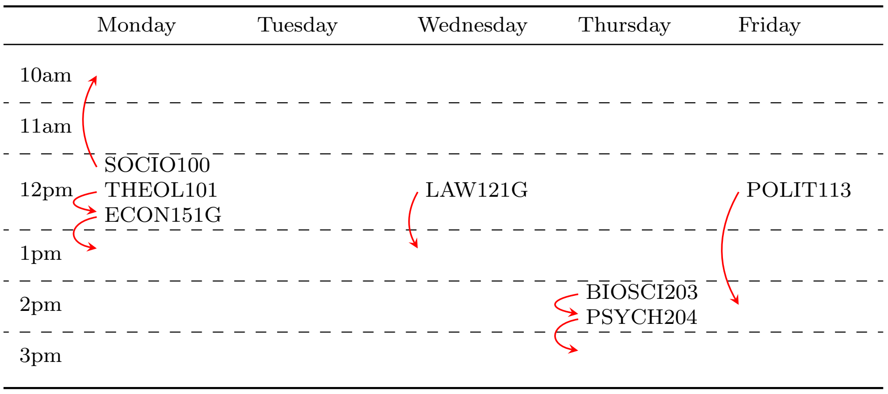
Time Period	Course Name	Enrolment
Mon 12pm	SOCIO 100	500
Mon 12pm	THEOL 101	490
Wed 12pm	LAW 121G	500
Thu 2pm	POLIT 113	347
Fri 12pm	BIOSCI 203	360

Table 3 Events without a room after room assignment

5.2 Event-based Perturbation

We first attempt to solve the problem of unassigned events (Table 3) using an event-based IP formulation (1)–(7) within the expanding neighbourhood algorithm (Algorithm 1). Around each of the four time periods, we initialise a neighbourhood to include similar sized events in one time period before and after. If this is insufficient to resolve the infeasibility, we expand the neighbourhood to include events from two time periods before and after. We set a simple penalty (for each event) of one unit for each hour moved and two units for each day moved from the original time period. There is a very small penalty for moving rooms within the same time period, so that this can occur, but only when necessary.

The solution to this problem is given in Table 4, where events are relocated to nearby time periods as shown, for a total penalty of 7. The events for "SOCIO100", "LAW121G" and "POLIT113" are able to move to other time periods with suitable vacant rooms in their respective neighbourhoods. The events for "THEOL101" and "BIOSCI203" are able to stay in their time periods and move into the rooms vacated by "ECON151G" and "PSYCH204" respectively. This manoeuvre commonly occurs in solutions to these problems, where one event changes time periods to free its room for another event. This is because each event has a unique set of time periods to which it can move, as determined by curricula and other requirements. Some events inevitably have a greater flexibility or a different set of suitable time periods.



	Monday	Tuesday	Wednesday	Thursday	Friday
10am					
11am					
12pm	SOCIO100 THEOL101 ECON151G		LAW121G		POLIT113
1pm					
2pm				BIOSCI203 PSYCH204	
3pm					

Table 4 Event-based solution

Solving the 6 IPs required for this problem was very rapid, as they all featured less than 500 binary variables, and terminated optimally in less than one second.

5.3 Course-based Perturbation

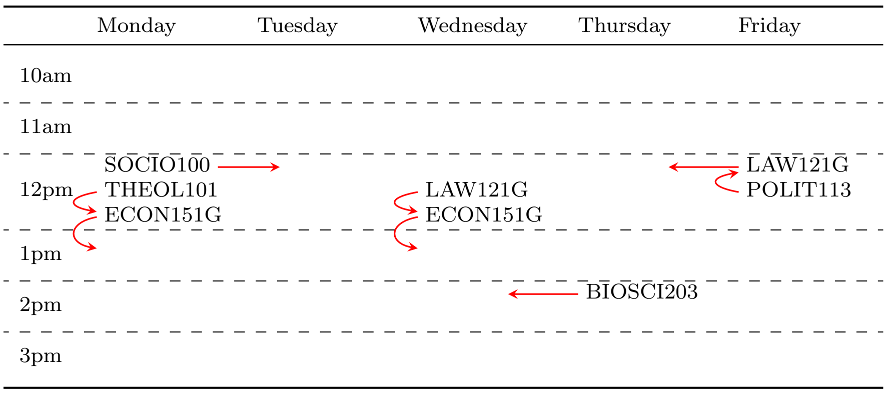
Because many faculties appreciate time stability for their courses, we also demonstrate the course-based IP formulation (1)–(10). In the previous solution (Table 4), perturbations to the events from several courses (e.g. "ECON151G", "LAW121G", and "POLIT113") incurred an unmeasured disruption to time stability. For the course-based model we use a different starting neighbourhood which includes all time periods at this same time of day across the week. This neighbourhood then expands to include all time periods one hour before and after. Because the first expansion causes the neighbourhood to include a total of 15 time periods, it is important to be selective about which events are included in each time period. To keep the model size manageable, we only include large events and rooms.

The solution to this problem is given in Table 5, where events are relocated to nearby time periods as shown for a total *event-based penalty* of 8. This is a greater penalty than in the event-based solution, however we are now able to resolve the infeasibility with no penalty incurred from disruption to the time stability.

Solving the 3 IPs required for this problem was again rapid, although there were up to 5000 binary variables in the largest case. However, due to the near-naturally integer property of these models, they still terminated optimally in less than one second.

6 Conclusion and Future Work

We have proposed a general integer programming-based approach for minimal perturbation problems which arise in practical university course timetabling. This



	Monday	Tuesday	Wednesday	Thursday	Friday
10am					
11am					
12pm	SOCIO100 → THEOL101 ECON151G		LAW121G ECON151G	←	LAW121G POLIT113
1pm					
2pm			←	BIOSCI203	
3pm					

Table 5 Course-based solution

approach is versatile, as there are substantial possibilities for customisation in the way the neighbourhoods are constructed and expanded. We have shown an example application of this process on real data from the University of Auckland.

An interesting extension to the proposed formulations would be the incorporation of multi-objective optimisation techniques, where additional quality measures are considered explicitly rather than implicitly in terms of disruption. This also leads to the question of whether it is possible (and desirable) to disrupt a given feasible timetable in order to improve the room assignment. Finally, we would like to consider equity and fairness across faculties and courses when making timetabling perturbations.

References

Barták R, Müller T, Rudová H (2004) A new approach to modeling and solving minimal perturbation problems. In: Apt KR, Fages F, Rossi F, Szeredi P, Vncza J (eds) Recent Advances in Constraints, Lecture Notes in Computer Science, vol 3010, Springer Berlin, pp 233–249

Beyrouthy C, Burke EK, Landa-Silva D, McCollum B, McMullan P, Parkes AJ (2007) Towards improving the utilization of university teaching space. Journal of the Operational Research Society 60(1):130–143

Bonutti A, De Cesco F, Di Gaspero L, Schaerf A (2012) Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Annals of Operations Research 194(1):59–70

Burke EK, Mareček J, Parkes AJ, Rudová H (2008) Uses and abuses of MIP in course timetabling. Poster at the workshop on mixed integer programming, MIP2007, Montréal, 2008, available online at `http://cs.nott.ac.uk/jxm/timetabling/mip2007-poster.pdf`

Ehrgott M (2005) Multicriteria optimization, 2nd edn. Springer Berlin

El Sakkout H, Wallace M (2000) Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 5(4):359–388

El Sakkout H, Richards T, Wallace M (1998) Minimal perturbation in dynamic scheduling. In: Prade H (ed) Proceedings of the 13th European Conference on Artifical Intelligence, ECAI-98

Kingston JH (2013a) Educational timetabling. In: Uyar AS, Ozcan E, Urquhart N (eds) Automated Scheduling and Planning, Studies in Computational Intelligence, vol 505, Springer Berlin, pp 91–108

Kingston JH (2013b) Repairing high school timetables with polymorphic ejection chains. Annals of Operations Research pp 1–16

McCollum B (2007) A perspective on bridging the gap between theory and practice in university timetabling. In: Burke EK, Rudová H (eds) Practice and Theory of Automated Timetabling VI, Lecture Notes in Computer Science, vol 3867, Springer Berlin, pp 3–23

Müller T, Rudová H, Barták R (2005) Minimal perturbation problem in course timetabling. In: Burke EK, Trick M (eds) Practice and Theory of Automated Timetabling V, Lecture Notes in Computer Science, vol 3616, Springer Berlin, pp 126–146

Phillips A, Waterer H, Ehrgott M, Ryan DM (2014) Integer programming methods for large scale practical classroom assignment problems. Submitted to Computers & Operations Research

Rezanova NJ, Ryan DM (2010) The train driver recovery problem - a set partitioning based model and solution method. Computers & Operations Research 37(5):845–856

Rudová H, Müller T, Murray K (2011) Complex university course timetabling. Journal of Scheduling 14(2):187–207

A Study of the Practical and Tutorial Scheduling Problem

Nelishia Pillay

School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal

+27 33 2605644

pillayn32@ukzn.ac.za

Abstract: The practical and tutorial allocation problem is a problem encountered at tertiary institutions and essentially involves the allocation of students to practical or tutorial groups for the different courses the student is enrolled in. Practical and tutorial scheduling for first year courses is becoming more and more challenging as the number of permissible course combinations and student numbers increase at tertiary institutions, and while this has previously been done manually and independently for each course, this is no longer feasible. The paper firstly presents a formal definition of the practical and tutorial scheduling problem. Low-level construction heuristics for this domain are defined and a heuristic approach for solving this problem is proposed. A tool namely, PRATS, incorporating this approach is described. The performance of PRATS on six sets of real-world data is discussed. The paper also reports on a hyper-heuristic implemented to automatically generate low-level construction heuristics and compares the performance of the generated heuristics to the human intuitive heuristics used.

Keywords: educational timetabling, construction heuristics, practical and tutorial scheduling, hyper-heuristic

1. Introduction

Educational timetabling essentially encompasses university course timetabling (McCollum et al. 2008), university examination timetabling (Qu et al. 2009) and school timetabling (Pillay 2013). Initially, the practical and tutorial scheduling problem formed part of the university course timetabling problem, with a period or afternoon session set aside for the practical or tutorial for a course. However, with time one session was found not to be sufficient for certain courses and practical and tutorial allocations were done separately from course timetabling for these courses, with more than one session scheduled for certain courses and students choosing the session that would suit them best. These selections were done manually and usually independently for the different courses. As the number of permissible course combinations, students and hence practical/tutorial sessions per course has increased at first year level, this is no longer feasible. As with the other types of educational timetabling which were initial done manually, as the complexity of these problems grew manual timetable construction was no longer an option and methods for automating the process were sought, this is now true of practical and tutorial allocations as well. The paper presents a description of the problem and a heuristic approach to solve this problem. Low-level construction heuristics are derived based on human intuition and evaluated for this purpose. A hyper-heuristic was implemented to automatically derive low-level construction heuristics for the domain and the evolved heuristics are compared to the human intuitive heuristics used to solve the problem. The proposed approach was used to solve this problem for the College of Agriculture, Engineering and Science at the University of KwaZulu-Natal. The performance of the heuristics on six sets of data corresponding to three semesters is presented and discussed.

The following section defines the practical and tutorial scheduling problem. Section 2 describes the heuristic approach proposed to solve the problem and the tool PRATS developed for use by administrators for practical and tutorial allocation. The genetic programming system implemented to evolve heuristics is explained in section 3. Section 4 discusses the performance of the heuristic approach and the evolved heuristics on the six real-world data sets. A summary of the findings of the study and future work is outlined in section 5.

2. The Practical and Tutorial Scheduling Problem (PTSP)

Science courses offered by the College of Agriculture, Science and Engineering have a three hour weekly practical or tutorial. Subjects such as Chemistry, Physics and Computer Science have a practical and courses like Mathematics and Statistics a tutorial. While the other educational timetabling problems, namely, school, university course and examination timetabling aims at scheduling events in timetable periods and venues, the practical and tutorial scheduling problem involves the allocation of practical/tutorial periods to each of the courses a student is registered for.

The practical/tutorial scheduling problem can be considered to be a version of the student sectioning problem (Muller et al., 2007; Muller et al., 2010; Murray et al., 2007) with the following differences:

- All students are pre-registered.
- The problem does not include the scheduling of lectures for students. This is done by the university administration using the block system and cannot be changed. The lectures are generally scheduled during the morning until lunch time and practicals and tutorials in the afternoon commencing at 14:10 (with one practical/tutorial timetable session in the morning at 09:35).
- Student preferences are not taken into consideration but rather department preferences. The department for each course specifies a set of practical/tutorial sessions for the course and students are allocated into the sessions so as to prevent clashes.
- Once the allocations to practicals/tutorials are made this can only be changed if the curriculum of the student changes resulting in a clash.

The student sectioning problem has essentially been solved by allocating students according to priorities, based on their preferences, and performing intelligent backtracking to resolve clashes (Muller et al. 2010). The research presented in this paper focuses on the derivation of heuristics for construction of solutions to this problem, both human intuitive and automatically generated.

The allocations for PTSP need to be made so as to ensure that:

- All courses for each student must be allocated a practical or tutorial period.
- The capacities for each practical/tutorial period must not be exceeded.
- Each student must not be scheduled to attend more than one practical in the same period, i.e. there must be no clashes.
- Grouping requirements - Students registered for certain degrees must attend practicals/tutorials in the same session for one or more courses. For example, all Engineering students must attend Physics on a Monday afternoon and Chemistry on a Tuesday afternoon.

Each year is comprised of two semesters and a practicial/tutorial schedule has to be created for each semester. The College has two campuses, with each campus requiring a different schedule. Each problem instance is defined in terms of:

- The courses offered by the College, practicals/tutorial sessions for each course and the capacity for each session. An example is illustrated in Table 1.

- The grouping requirements specifying the degrees and the session allocations for each course. Table 2 provides an example of this data, e.g. all Geological Sciences students (BSGLS) must have a Chemistry practical on a Tuesday, Geography on a Wednesday, a Mathematics tutorial on a Thursday and Geology on a Friday.
- Student registrations - Student number, degree, College (some schools in the College offer services courses for degrees offered by other Colleges) and courses for which each student is registered.

  The following section proposes a method for solving the PTSP.

Table 1. Example of Course Details

Course	Mon	Tues	Wed	Wed(am)	Thurs	Fri
BIOL101	186	186	186	186		
BIOL103						187
BIOL195				192		
CHEM110	96	192	192	192	192	120
CHEM195						72
COMP100					250	
GEOG110			184			
GEOL101	90				51	90
MATH130		408	378			
MATH140						72
MATH150		350		750		350
MATH195				60		
PHYS110			159			
PHYS131	240	320			320	
PHYS139	42					
PHYS195			25			
STAT130	350			350	350	

3. Solving the PTSP

This section firstly presents the low-level heuristics and method used to solve the practical and tutorial scheduling problem. This is followed by a description of the tool developed, employing the method outlined in section 3.1.

Table 2. Example of Group Requirements

Degree	Mon	Tues	Wed	Wed(am)	Thurs	Fri
B-MDSC		PHYS131	BIOL101		CHEM110	MATH150
BMDSCP		PHYS131	BIOL101		CHEM110	MATH150
BOPT	PHYS139	MATH150	CHEM110			BIOL103
B-PHAR	PHYS131	MATH150	CHEM110			BIOL103
B-PHYS		PHYS131				BIOL103
BS-ENS	PHYS131	CHEM110			GEOL101	MATH150
BSGLS		CHEM110	GEOG110	MATH150	PHYS131	GEOL101
BSCF	GEOL101					
BSCA	GEOL101					
BSLES	GEOL101					
BSAPC	GEOL101					
BSBLS	GEOL101					
BSCMB	GEOL101					

3.1 Heuristic Approach for Solving the PTSP

Low-level construction heuristics have proven to be effective in creating solutions to educational timetabling problems. These heuristics have essentially been used to order events, e.g. examinations, lessons, in order of difficulty to schedule. For example in the domain of examination timetabling graph colouring heuristics, e.g. largest degree, largest weighted degree, largest enrollment , largest colour degree and saturation degree (Carter et al., 1996) have been used. Usually, one heuristic is used to order examinations for scheduling. For example, the largest degree heuristic is the number of clashes that an examination can be involved in and a higher value indicates a more difficult exam to schedule. In some instances the low-level heuristic is used to create an initial solution which is further improved using optimization methods such as variable neighbourhood search, tabu search and evolutionary algorithms (Qu et al. 2009). In Carter et al. (1996) a low-level heuristic is used to create an initial timetable which is improved using backtracking. These graph colouring heuristics are also used for university course timetabling. Similar heuristics are used for timetable construction for the domain of school timetabling (Pillay, 2013). Given the crucial role that low-level construction heuristics have played in educational timetabling it was felt that the first step to solving the PTSP would be to define low-level construction heuristics for this domain.

The problem essentially involves assigning practical and tutorial sessions for each student. For this domain it was decided to view this as the student being the entity to assign and thus heuristics for assessing the difficulty of scheduling each student was investigated. This was defined in terms of the practicals/tutorials that had to be scheduled for the student. Two heuristics were defined for ordering students for practical/tutorial allocation:

- Allocation degree - the number of allocations for the student, e.g. if the student is registered for three courses that require practical/tutorial allocations the allocation degree will be 3. Priority is given to students with a higher allocation degree. It is anticipated that students requiring more allocations will be more difficult to schedule as there is a greater chance of clashes.
- Total opts - This heuristic is the sum of the options for each practical/tutorial that needs to be scheduled for the student. The reasoning behind this is that the less options there are available the more difficult it will be to schedule the practicals/tutorials for the student.

These heuristics are static and remain constant throughout the allocation process. These are calculated for each student at the beginning of the allocation process. In addition to heuristics for choosing which student to perform allocations for first, a low-level heuristic, namely option degree, for selecting which practical/tutorial to schedule first was also defined. This heuristic is the number of options, i.e. sessions, for each practical/tutorial. Practicals/tutorials with fewer options are given priority to be scheduled. The final low-level heuristic defined is to choose the session to schedule the practical in. The session with the higher capacity amongst the feasible sessions, i.e. sessions with sufficient capacity that do not result in clashes, is chosen. Both these heuristics are dynamic and need to be updated during the allocation process as allocations will change the number of options and capacities. Table 3 provides a summary of the low-level heuristics.

Table 3. Low-level Heuristic Summary

Category	**Heuristics**	**Type**
Student allocation heuristics	• Allocation degree • Total opts	Static
Course heuristics	• Option degree	Dynamic
Session heuristics	• Capacity degree	Dynamic

The allocation process begins by sorting the students according to one of the student allocation heuristics. The allocation for each student starts with sorting the practicals/tutorials to be scheduled according to the option degree. A period is assigned to the practical/tutorial with the lowest option degree using the capacity degree heuristic. If the degree the student is registered for and practical/tutorial is one of the group requirements, the specified session is allocated. The allocated practical/tutorial is removed and the list to be scheduled resorted according to the option degree as the value of the option degree may change in some cases as a result of the allocation. If a period cannot be found as a result of insufficient capacity or if the allocation will result in a clash, the allocation is not made and the number of unallocated practicals/tutorials is incremented. A list of clashes and insufficient capacities for the different courses is maintained and reported together with the number of unallocated practicals/tutorials at the end of the allocation process. The following section describes the tool developed to be used by administrators for practical/tutorial allocation. The algorithm is illustrated in Figure 1.

3.2 PRATS (Practical and Tutorial Scheduler)

The importance of working with administrators that will be using the system and bridging the gap between research and practice is emphasized in McCollum (2007). Thus, the tool implementing the method describe in section 3.1 for practical/tutorial allocation was developed in consultation with the College Director: College Professional Services and the College Academic Services. Prior to the implementation of PRATS the practical/tutorial allocation for first year was done manually. The schedule was stored in an Excel spreadsheet which was given to administrators to update in cases where students wished to change allocations as a result of any changes in curriculum for the particular student.

```
begin
 for 1 to no_of_students
  Sort practicals/tutorials to be allocated
   while(there are practicals/tutorials to allocate)
    if the practical/tutorial is part of  a group requirement
     allocate specified session
    else
     sort the feasible sessions for allocation
     schedule the practical/tutorial in the session with the maximum capacity
     if  a session cannot be found for the practical/tutorial
      increment the number of unallocated practicals/tutorials
      update the insufficient capacities/clashes list
     endif
   endif
  endwhile
endfor
 Report the number of unallocated practicals/tutorials
 Report insufficient capacities
 Report clashes
end
```

Figure 1. Practical/tutorial allocation algorithm

The spreadsheet contained:

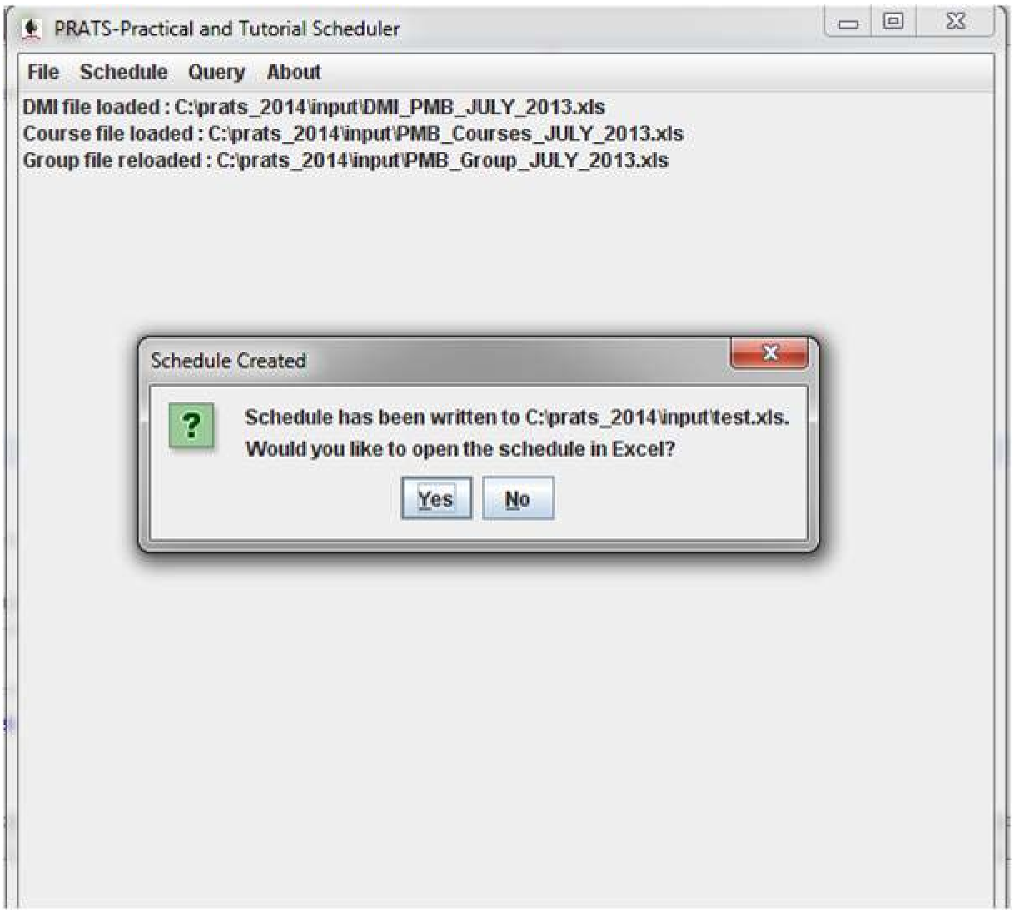
- The courses and capacities for each practical/session given the current allocations.
- Practical and tutorial allocations for the students registered for first year Science modules in the College.
- Formulae linking the student allocations and capacities for each practical/tutorial session so that any de-allocation/reallocation of practicals/tutorials for a particular student results in the capacity for the session/s involved being automatically updated.

It was requested that the software developed should solve the practical and tutorial allocation problem and output a spreadsheet with the same format as the that of the spreadsheet schedule that was previously created manually. The tool

PRATS was developed for this purpose. Input to for each problem instance includes:

- A spreadsheet of all student registrations for first year Science courses in the College (DMI file).
- A spreadsheet listing all first year Science courses offered by the College and the practical/tutorial sessions for each course and corresponding capacities.
- The spreadsheet listing the group requirements for specific degrees (Groups file).

Figure 2 illustrates a PRATS session. The user is required to load the input files and choose the option to create a schedule, specifying the file the schedule must be stored in.



DMI file loaded : C:\prats_2014\input\DMI_PMB_JULY_2013.xls
Course file loaded : C:\prats_2014\input\PMB_Courses_JULY_2013.xls
Group file reloaded : C:\prats_2014\input\PMB_Group_JULY_2013.xls

Schedule has been written to C:\prats_2014\input\test.xls.
Would you like to open the schedule in Excel?

Figure 2. Example of a PRATS session

PRATS produces an Excel spreadsheet file with the following tabs:

- The first tab lists the courses, practical/tutorial sessions and capacities as well as the allocations for all students and includes formulae linking student allocations and student capacities.

- The second tab lists all the courses and additional capacity needed for courses where the number of students registered exceeds the total capacity of all sessions for the course practical/tutorial.
- The third tab list the clashes that cannot be resolved due to a student being registered for two courses that have the same practical/tutorial session with no other options, and the number of students with this clash. An example is illustrated in Table 4. This usually results when students are registered for incorrect combinations. Another reason for this is that the combination was not anticipated, in which case the practical/tutorial for one of the sessions may have to be rescheduled. The subsequent tabs in the spreadsheet list the students for each clash so that they can be contacted and informed. An example is illustrated in Table 5.

Table 4: Example of Spreadsheet Tab Listing Courses with Practical/Tutorial Clashes

Clash	First Course	Second Course	Period	No. of Students
Clash1	MATH196	COMP102	Thurs	1
Clash2	STAT140	COMP106	Wed(am)	3
Clash3	PHYS120	BIMI120	Wed	5
Clash4	MATH144	MATH130	Fri	4
Clash5	ENVS120	BIMI120	Wed	18

Table 5: Example of Student Data for a Clash

Student Number	Surname	Name	Degree
212500323	Mhlongo	Slungile Sunshine	BSCA
213510964	Wanda	Ayanda	BSAPC
212540361	Mvelase	Bonokwakhe Sthembiso	BSCA

The following section describes the performance of PRATS on six sets of data for the College of Agriculture, Engineering and Science.

4. PRATS Performance

During the development phase of PRATS it was tested on data for the first semester of 2013 for which manual schedules had been created. Details of the data sets are listed in Table 6.

Table 6: Data Sets for Semester 1 2013

Data Set	No. of Courses	No. of Sessions	No. of Students	No. of Group Requirements
PMB_S1_2013	18	6	1223	0
WSTVL_S1_2013	17	6	2771	28

It was found that the application of each of the student allocation heuristics separately was not effective. The concept of primary and second heuristics used in Pillay et al. (2009) was introduced in this study. The students to be allocated are firstly sorted for scheduling using the primary heuristic. In the case of ties a secondary heuristic is applied to the tied students to determine the ordering. The combination of the allocation degree as the primary heuristic and total opts as a secondary heuristic was found to be the most effective.

PRATS allocated all practicals/tutorials for all students within the specified capacities for PMB_S1_2013. A clash between two courses, namely Nutrition and Geography with two students registered for this combination was reported. This combination is not permitted as generally Dietetics students register for Nutrition and Geography does not form part of the programme. All student practical/tutorials were also scheduled for WSTVL_S1_2013. The capacity for BIOL103 was exceeded by 1, however it was found that the student in question was registered for both BIOL101 (which was the module required for the programme the student was registered for) and BIOL103. The student was advised to deregister from BIOL103. Three clashes were reported by PRATS. The combinations causing the clashes were not permitted and the students notified.

PRATS was used to schedule the first year practicals/tutorials for the second semester of 2013. The details of both problem instances are listed in Table 7.

Table 7: Data Sets for Semester 2 2013

Data Set	**No. of Courses**	**No. of Sessions**	**No. of Students**	**No. of Group Requirements**
PMB_S2_2013	22	6	1483	5
WSTVL_S2_2013	20	6	2939	9

PRATS performed all the required allocations for PMB_S2_2013 within the specified capacities. Two clashes resulting from illegal combinations was reported. PRATS was not able to allocate the practicals/tutorials for all students for WSTVL_S2_2013. Further investigation revealed that the practical/tutorials not scheduled were those that had only one session option. As students with the highest number of allocations required were given priority for scheduling, this resulted in those students with courses having just one session for the practical/tutorial being scheduled later in the allocation process at which stage there was insufficient capacity for these allocations. This resulted in the introduction of a third student allocation heuristic, namely, the one-option heuristic which gives students with at least one course with one practical/tutorial session priority for allocation. The combination of one-opt as the primary heuristic and allocation degree as the secondary heuristic resulted in all practicals/tutorials being scheduled. Four clashes resulting from students registering for illegal combinations were reported and the students notified.

PRATS was also used to generate schedules for the first semester of 2014. The data sets for both campuses are presented in Table 8.

Table 8: Data Sets for Semester 1 2014

Data Set	**No. of Courses**	**No. of Sessions**	**No. of Students**	**No. of Group Requirements**
PMB_S1_2014	13	6	1436	1
WSTVL_S1_2014	12	6	3548	14

Both the heuristic combinations, i.e. allocation degree as a primary heuristic and total opts as the secondary heuristic and one-option as a primary heuristic and allocation degree as a secondary heuristic are able to allocate all practicals and tutorials for all students. For PMB_S1_2014 one illegal course combination was found and the student notified while there were no clashes for WSTVL_S1_2014.

The runtime of the heuristic approach employed by PRATS is on average a second for all problem instances. Given that this study has revealed that the most appropriate combination of student allocation heuristics used is problem dependent, the option of automating the generation of the student allocation heuristic to use was investigated. This essentially involves implementing a hyper-heuristic for the generation of low-level construction heuristics (Burke et al. 2013). As the best heuristic to use is problem dependent, the generated heuristic is disposable. From previous work done in this domain, it is evident that genetic programming has chiefly been employed for the induction of construction low-level heuristics (Burke et al. 2009; Burke et al. 2013). A genetic programming hyper-heuristic was implemented and tested on the six problem instances. The following section describes the hyper-heuristic and discusses its performance on the six data sets.

5. Generative Constructive Hyper-Heuristic

This section describes the hyper-heuristic used to evolve low-level construction heuristics for the practical and tutorial scheduling problem. Section 5.1 describes the genetic programming system implemented and section 5.2 discusses the performance of the hyper-heuristic in solving the PTSP.

5.1 Genetic Programming Hyper-Heuristic

The hyper-heuristic employs genetic programming to evolve low-level construction heuristics. Genetic programming is an evolutionary algorithm that explores a program space to identify a program which when executed will produce an optimal or near optimal solution (Koza 1992). The generational genetic programming algorithm illustrated in Figure 3 was implemented.

```
Create initial population
Repeat
  Evaluate the population
  Select parents
  Apply genetic operators to parents
Until the termination criteria are met
```

Figure 3. Genetic programming algorithm

Each element of the population is a parse tree representing a low-level construction heuristic. Each parse tree is comprised of elements from the function the terminal sets. The function includes the following operators:

- Arithmetic operators: +, -, *, /. The division operator is protected division which returns a value of 1 if the denominator is zero.
- The *if* operator which performs the standard if-then-else function and has an arity of 3.
- Arithmetic logical operators: <, >, <=, >=, ==, !=. These operators perform the standard arithmetic logical operations. These operators can only be included in the subtree representing the first child of the *if* operator as this is the branch of the tree representing the condition that must be met.

The terminal set is comprised of the low-level student allocation heuristics defined in section 3.1:

- a - Represents the allocation degree heuristic.

- b - Represents the one-option heuristic.
- c - Represents the total opts heuristic.

Figure 4 illustrates examples of population elements. Each element is created by firstly choosing an element from the function set to ensure that trivial trees are not created. The tree is constructed using the grow method (Koza 1992) which involves the random selection of elements from the function and terminal set until the maximum specified depth is reached at which point only elements from the terminal set are chosen.

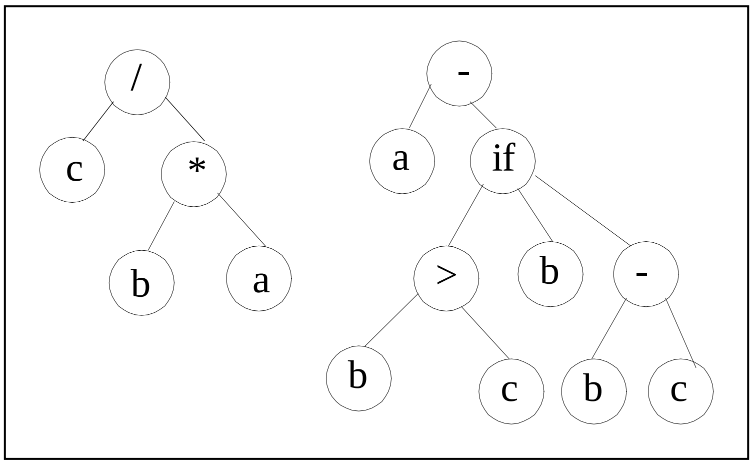


Figure 4. Examples of parse trees representing low-level construction heuristics

Each individual is evaluated by using the individual to create a schedule. This achieved by applying the heuristic to each student producing a numerical value/heuristic which represents the difficulty of scheduling the practicals/tutorials for that student. The students are sorted in descending order according to the heuristic and the allocations for each student performed in order. The fitness of an individual is the number of unallocated practicals and tutorials in the created schedule.

Tournament selection (Koza 1992) is used to choose parents for regeneration. This selection method essentially involves randomly selecting *t* elements of the population and returning the fittest individual as a parent. Selection is with replacement so an individual can be chosen as a parent more than once.

The standard mutation and crossover operators (Koza 1992) are used for regeneration. The mutation operator replaces a randomly selected subtree in the copy of the parent with a newly created subtree. Crossover randomly selects a subtree in each of two parents and the subtrees are swapped to create two offspring.

Performance of the hyper-heuristic on the six data sets is discussed in the following section.

5.2 Hyper-Heuristic Performance

The parameter values used are listed in Table 9. These values were determined empirically.

Table 9. Parameter values

Parameter	**Values**
Population size	50
Maximum tree depth	4
Tournament size	4
Crossover %	50%
Mutation %	50%
Mutation depth	3

Due to the stochastic nature of genetic programming ten runs were performed for each problem. The average runtime for the hyper-heuristic is five seconds. All runs performed were successful at producing schedules with all practicals and tutorials for all students allocated, within the specified capacities, for the six problem instances. Furthermore, heuristics producing the optimal schedule were found in the initial population and further optimization was not needed. The heuristics evolved for each seed were different and there did not appear to be any similarities between the evolved heuristics producing optimal solutions. This will be researched further as part of future work.

6. Conclusion and Future Work

This paper introduces the practical and tutorial scheduling problem and a heuristic approach to solve this problem. Five low-level construction heuristics have been defined for this problem based on human intuition and categorized as student allocation heuristics, course heuristics and session heuristics. The study revealed that it was necessary to combine the student allocation heuristics as primary and secondary heuristics in order to find a solution to the problem. The heuristic approach was incorporated into a tool PRATS which was able to find feasible solutions to six real-world problem instances. It was also found that different student allocation heuristic combinations may be needed to find solutions to different problems. This led to the implementation of a hyper-heuristic to automatically generate a student allocation heuristic. This approach appeared to be effective, finding solutions to all six problems. Future work will investigate further examination of the generated heuristics to identify possible patterns or similarities in the functions. In addition to this the reusability of the optimal heuristics generated will also be investigated.

Acknowledgements. This work is based on the research supported in part by the National Research Foundation of South Africa for the Grant CSUR13091742778. Any opinion, finding and conclusion or recommendation expressed in this material is that of the author(s) and the NRF does not accept any liability in this regard.

7. References

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. &Woodard, J. (2009) Exploring Hyper-Heuristic Methodologies with Genetic Programming. *Computational Intelligence*, 6, 177-201.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. & Qu, R. (2013) Hyper-Heuristics: A Survey of the State of the Art. *Journal of the Operational Research Society*, 1-30.

Carter MW, Laporte G, Lee SY (1996) Examination Timetabling: Algorithmic Strategies and Applications. *Journal of the Operational Research Society*, 47(3), 373-383.

Koza. J.R. (1992) Genetic Programming I : On the Programming of Computers by Means of Natural Selection, MIT Press.

McCollum, B. (2007) A Perspective on Bridging the Gap between Research and Practice in University Timetabling. Practice and Theory of Automated Timetabling VI (eds. E.K.Burke and H.Rudova), Lecture Notes in Computer Science Volume 3867, Springer 2007, pp 3-23.

McCollum, B., McMullan, P., Paechter, B., Lewis, R., Schaerf, A., DiGapsero, L., Parkes, A.J., Qu, R . & Burke, E.K. (2008). Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal of Computing*, 22(1), 120–130.

Muller, T. & Murray, K. (2010) Comprehensive Approach to Student Sectioning. *Annals of Operational Research*, 181, 249-269.

Muller, T., Murray, K. & Schluttenhofer, S. (2007) University Course Timetabling and Student Sectioning System. In Proceedings of the International Conference on Automated Planning and Scheduling, pp. 1-4.

Murray, K. & Muller, T. (2007) Real-Time Student Sectioning. In Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications, pp. 1-3.

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, 482-491.

Pillay, N. (2013) A Survey of School Timetabling. Annals of Operations Research, February 2013, DOI: 10.1007/s10479-013-1321-8.

Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G. & Lee, S.Y. (2009) A Survey off Search Methodologies and Automated System Development for Examination Timetabling. *Journal of Scheduling*, 12(1), 55–89.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Hybrid Local Search for The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

Haroldo Gambini Santos · Janniele Soares · Tùlio Toffolo

Abstract In this work we present a multi-neighborhood, parallel local search approach for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MMRCMPSP). The search in multiple neighborhoods is conducted in parallel with dynamic load balancing among processors. Our solver works with an indirect solution representation and navigates through space of the feasible solutions by combining heuristics and mathematical programming. A perturbation procedure which alters a subset of job modes and still keeps the solution feasible is introduced. Very encouraging results were obtained, improving several best known solutions published at the MISTA Challenge.

Keywords Multi-Mode Resource-Constrained Multi-Project Scheduling Problem · Scheduling · Local Search

1 Introduction

A *Project Scheduling Problem* (PSP), in its general form, consists in scheduling the processing times of *jobs* (or activities) contained in a project. These jobs are interrelated by precedence constraints, that is, a job may require another job to be finished before its start. This class of problems models many situations of practical interest in engineering and management sciences in general.

Santos, H.G.
Computer Science Department, Federal University of Ouro Preto, Brazil
E-mail: haroldo@iceb.ufop.br

Soares, J.A.
Department of Computing and Information Systems, Federal University of Ouro Preto, Brazil
E-mail: janniele@decsi.ufop.br

Toffolo, T.A.M.
Computer Science Department, Federal University of Ouro Preto, Brazil
E-mail: tulio@toffolo.com.br

One well known application of the PSP is the area of Construction Scheduling [4]. For a broad review of this area we refer the reader to [5, 8, 10, 13, 14] and [2].

Recently, with the objective of bridging the gap between theory and practice, the MISTA 2013 [1] challenge was organized. In this challenge, a generalization of the PSP with resource constraints which takes into account several aspects of real world applications was considered: The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MMRCMPSP). Several instances with different characteristics and sizes were proposed. Competing methods were evaluated in a controlled experimental environment. We participated as part of the GOAL team, which proposed an Integer Programming based approach for the MMRCMPSP. Our team was ranked third in this competition.

In this paper we explore the hybridization of Integer Programming and local search heuristics, incorporating some of the best characteristics of our method with some of the efficient local search procedures proposed by other teams.

The paper is organized as follows: Section 2 presents the MMRCMPSP; Section 3 introduces our algorithm and finally, Sections 4 and 5 present, respectively, conclusions and future works.

2 The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

The MMRCMPSP is stated as follows: a set P of projects, each project $p \in P$ consisting of a set $J_p = \{1, \ldots, |J_p|\}$ of non-preemptive jobs, has to be scheduled. Each project p has also a *release time*, that is, a time when its jobs processing may be started. The start and end of a project are delimited by *dummy* jobs 0 and $|J_p| + 1$, respectively the first and last jobs of each project.

To schedule a project means to determine the starting time of all of its jobs, subject to the precedence constraints among them and also their *resource* consumption in face of the available resources. Jobs may consume *local* resources – exclusive resources of a project – and *global resources* – resources shared among all projects. These resources can be either *renewable* – with capacity fixed per time unit during the project duration – or *non-renewable* – with capacity fixed per project duration. Each job may be executed in one or more *execution modes*, each requiring a specific amount of resources consumption and resulting in different durations for a job completion. Note that dummy jobs do not have any resource consumption and their duration is always zero.

A lower bound on a project earliest finish time is the *critical path duration.* The *Critical Path Method* [12] is a tool for general project management that represents the precedence constraints as a network, where each job is a node and arcs connect jobs to its successors and predecessors, and calculates the earliest and latest start and finish times for each job such that the project is not delayed, while observing the precedence constraints. The critical path itself is the sequence of related jobs that cannot be delayed without delaying the whole project, denoted by a path between the two dummy jobs in the network. Thus, the critical path duration is the sum of these jobs durations. To compute a valid critical path based bound for a MMRCMPSP instance one can set job durations to the minimum among all possible execution modes.

Once a project is scheduled, its *makespan* is defined as the difference between the project finish and release times, and the *project delay*, is defined as the difference between the critical path based bound and the actual project duration.

In order to measure the quality of the solutions submitted to the MISTA challenge, an objective function with two components was proposed: to minimize the *total project delay* (TPD) and the *total makespan* (TMS). The TPD is defined as the sum of all projects delays, and the TMS is defined as the time required to finish all projects, i.e., the difference between the maximum finish time of a project and the minimum release time of a project. TPD is the main objective, while TMS is a tie-breaker.

3 The Proposed Algorithm : Overall Working

One fundamental characteristic of our method is that it always navigates in the feasible search space of solutions. To accomplish this we employ an indirect solution representation abstracting starting times of tasks: valid topological orderings are decoded by a constructive algorithm which allocates each task in the sequence as soon as possible, this approach was also used by [3, 6].

The selection of an initial set of valid execution modes for tasks is also computationally challenging: [3, 6] relax this constraint and move it to the objective function. In our approach, the definition of an initial set of modes is carried out by the solution of a binary programming model.

Thus, our solution representation consists in an ordered pair $(\pi, \mathbb{M})$ where π is valid topological sorting of J and $\mathbb{M}$ is a valid set of modes. Since the latter is much harder to determine, the next subsection will describe our approach for this part.

Subsequently, local search is performed in a Variable Neighborhood Descent [7] fashion, iterated with a perturbation of modes. One novel feature of our algorithm is a controlled perturbation in the set of modes, which also involves a binary programming model.

3.1 Initial Feasible Solution

The initial set of selected modes must respect resources constraints. Since the selection of processing modes also determines the duration of tasks, one greedy strategy is to prioritize the selection of fast processing modes. Considering this latter criterion, it is important to observe that it does not guarantees a smaller TPD, since renewable resources constraints can increase the starting times of tasks. Considering non-renewable resources, a greedy strategy can have worse consequences: many combinations of modes may not respect the usage of these resources.

The binary program to select this initial set of modes considers: J jobs with respective processing times p_{jm} and N non renewable resources. Each job has a set M_j of possible modes and the non-renewable resource n consumption of job j in mode m is denoted as r_{jmn}. Thus, the following binary program is solved to select which mode m job j will be allocated, considering its respective decision variables x_{jm} and resource availability q_n for each non renewable resource:

$$
\begin{aligned}
min. : & \\
& \sum_{j \in J} p_{jm}.x_{jm} && && (1)\\
s.t. : & \\
& \sum_{j \in J} \sum_{m \in M_j} r_{jmn} x_{jm} && \leq q_n && \forall n \in N \quad (2)\\
& x_{jm} && \in \{0,1\} && \forall j \in J, m \in M_j \quad (3)
\end{aligned}
$$

Which corresponds to the NP-Hard problem of the 0-1 multidimensional knapsack problem. Fortunately, modern integer programming solvers [9, 11] consistently solve this problem to the optimality considering all instances of the competition, always in a fraction of a second.

3.2 Neighborhoods

In this section we present the neighborhoods used in our search methods. Most of them were proposed or were inspired by the works which won the first two places of the MISTA 2013 challenge [3, 6].

These neighborhoods can be classified in two types: the ones which change modes and the ones which change sequence. The search only operates in the feasible search space: neighbors which disrespect non-renewable resources constraints are ignored and neighbors which disrespect processing dependencies are repaired using a topological order constructed using the invalid order to define priorities.

3.2.1 Change One Mode (COM)

This first neighborhood is based on [3, 6], and receives as a parameter a vector of modes initially allocated to jobs. Since we concentrate in the feasible search space this is a quite restricted neighborhood, since many single mode changes do not produce feasible solutions.

3.2.2 Change Two Modes (CTM)

Similar to the movement change one mode, but explores changes on pairs of modes. This is a significantly larger neighborhood since it has a quadratic size with respect to its input and because valid two mode changes are more common.

3.2.3 Change Three Modes (CTRM)

We observed that an unrestricted search considering all valid change mode triples would be too expensive. At the same time, jobs which are closer in the dependency graph tend to be much more sensible to changes in the mode of their neighbors than changes in modes of distant jobs. So we restricted the involved jobs j_1, j_2 an j_3 to be consecutive in the dependency graph: $(j_1 \rightarrow j_2 \rightarrow j_3)$.

3.2.4 Change Four Modes (CFM)

Just like three mode change this neighborhood restricts neighbors by imposing the same dependency relationships in the dependency graph.

3.2.5 Invert Subsequence (INV)

This movement is based on [6], and receives as a parameters a sequence of jobs and an integer k that determines the size of the subsequences to be inverted. For each position i of the sequence the next $k - 1$ jobs are included in a subsequence and inverted.

Figure 1 shows an example of this movement using subsequence with size $k = 4$ and starting position 2.

s'=INV(s,2,4)

s	2	3	5	8	14	10	7	12	9
s'	2	14	8	5	3	10	7	12	9

Fig. 1 Invert Subsequence

3.2.6 Shift Jobs (SJ)

This movement was proposed by [3], and is responsible for systematically moving forward or backward a job in the sequence in k neighbor positions. The method receives as a parameter a sequence of jobs and a parameter k that determines the maximum distance which job j can be displaced.

Figure 2 shows an example of this movement, shifting the job allocated at position 4 3 positions ahead.

s'=SJ(s,4,3)

s	2	3	5	8	14	10	7	12	9
s'	2	3	5	14	10	7	8	12	9

Fig. 2 Shift Jobs

3.2.7 Swap Jobs (SWJ)

In this neighborhood, which it was based on [3, 6], two jobs in the sequence are swapped. This method receives as a parameter a sequence of jobs and a parameter k which restricts the maximum distance between the two jobs to be swapped.

Figure 3 shows an example of this movement, swapping the job at position 5 with the job currently occupying position 5.

s'=SWJ(s,*3*,5)

s	2	3	5	8	14	10	7	12	9
s'	2	3	14	8	5	10	7	12	9

Fig. 3 Swap Jobs

3.2.8 Compact Project (CP)

This movement is based on the proposal of [3] and tries to accelerate the completion time of a project by shifting tasks of other projects which appear in the mid the project sequence for later processing. The objective is to *shrink* the later portion of the project.

In our implementation a parameter $perc \in (0, 1]$ determines percentage of tasks which will be compressed. These tasks are selected starting from the end of project, i.e: $perc = 0.5$ means that the second half of the tasks in the project sequence will be compressed.

Figure 4 shows an example of this movement, compacting 100% the jobs of project p_1, which is shown in gray.

s'=CP(s,1,1)

s	2	3	5	8	14	10	7	12	9
s'	2	3	5	14	10	9	8	7	12

Fig. 4 Compact projects

3.2.9 Shift Project (SP)

This movement is based on the proposed by [3] and is similar to the shift jobs, but moves forward or backward all jobs of a project p on k positions. The movement receives as parameters a maximum shifting distance k and a parameter p which determines the involved.

Figure 5 shows an example of this movement to the project p_1 (shown in gray) and $k = -2$.

s'=SP(s,1,-2)

s	1	4	3	2	9	7	5	8	6
s'	1	2	4	7	5	3	9	8	6

Fig. 5 Shift a project

3.2.10 Swap Two Projects (SWP)

This movement it is based on the proposals of [3, 6], and is similar to the idea of swapping jobs, but now the swap happens between two projects.

Considering the swap of two projects, p_1 and p_2, a new subsequence is generated as follows: firstly, one identifies the starting position sp of the project which finishes earlier.

In a vector R are stored all jobs before sp that do not belong to projects p_1 or p_2. In a vector D are stored all jobs after sp that do not belong to any of these projects too.

The reconstruction of the sequence is made by allocating all jobs of the vector R. Subsequently, all jobs that belong to the project that finished later, followed by all jobs that belong to the project that finished earlier and finally all the jobs of the vector D.

Figure 6 shows an example of this movement, swapping the projects p_1, dark gray, and p_2, in light gray.

s'=SWP(s,1,2)

s	5	3	9	6	13	1	7	2	25
s'	5	3	6	2	9	13	1	7	25

Fig. 6 Swap two projects

3.3 Perturbation

As observed by [6], in the MMRCMPSP, changes in the mode set appear to have a much more profound effect than changes in the sequence. We developed a perturbation strategy where only a controlled number of modes is randomly changed, keeping the resulting mode set feasible. In our method, a parameter %MC determines the percentage of jobs which will have their modes changed at each application of the perturbation procedure.

This procedures solves exactly the same problem as in subsection 3.1, but with a different objective function. The main idea is to change the mode of some jobs while trying to minimize the *collateral effect* induced by the satisfaction of the non-renewable resources constraints. More specifically, the model allows any other task to have its mode changed too, but tries to minimize these cases. As input this method receives the current mode m_j of each job and a boolean value c_j indicating if one wants to perturb the current solution by changing this job mode. Coefficients p_{jm} of the binary programming model are substituted as follows:

$$p_{jm} = \begin{cases} M & \text{if } c_j \text{ is true and } m \text{ is the current mode of job } j \\ & \text{ or if } c_j \text{ is false and } m \text{ is not the current mode of } j \\ \epsilon & \text{otherwise} \end{cases}$$

Where M represents a sufficiently large constant (500 in our experiments) and ϵ a small one (1 in our experiments). Small random changes with value $r \in \{1, \ldots, 10\}$ are also introduced in p_{jm} to further randomize the procedure.

The perturbation procedure is applied after a complete search on all previously presented neighborhoods is conducted and no improved solution was found.

3.4 Paralell Search

Many neighbors presented before contain a large number of solutions. One parallelization strategy would be an static neighborhood decomposition, where a fixed portion of the neighborhood is sent to a processor. This strategy can introduce a severe imbalance of load among processors, since the computing time to to evaluate different solutions varies significantly, depending on several steps in the decoding of each solution. Thus, we opted for a dynamic load balancing strategy: a pool of neighbors to be evaluated is build and different thread continually request new neighbors to be evaluated until a better solution is found or all neighbors have been processed.

4 Computational Experiments

All algorithms were coded in C++ and the binary programming models were solved by CPLEX 12.6. The code was compiled with GCC 4.7.1 using flag -O3. All tests ran on a computer with an Intel Core i7 processor[1] and 24 Gb of RAM, running OpenSUSE Linux 12.1.

The developed method ran in parallel using 4 threads. Parameter values were obtained after some preliminary empirical evaluation and are presented on Table 1. These parameters correspond respectively the limits used in Neighborhoods Invert Subsequence (INV), Shift Jobs (SJ), Swap Jobs (SWJ), Shift Project (SP),the percentage of Compact Project (CP) and the percentage of mode changes (MC) at each perturbation.

Table 1 Parameters used for tests

Local Search Parameters					
INV	SJ	SWJ	SP	%CP	%MC
3	2	2	5	0.5	0.02

The results of the instance set A, used on the first stage of the competition, were announced during the qualification phase. Results of the instances of the set B and X, on the second and third phase of the competition, were announced during the conference.

Table 2 shows the best results found by the proposed approach, as well the mean and standard deviation, after 10 runs within 300 seconds of runtime. Instances that were better or equal to the results reported in the MISTA 2013 Challenge site are emphasized.

5 Conclusions and Future Works

In this work we presented a hybrid search method which combines a multi-neighborhood parallel local search with mathematical programming. Perturbation is executed in a controlled way, so that the search method always jumps from one feasible solution to another.

[1] the same of the MISTA 2013 Challenge

Table 2 Best and average results after 10 runs of the algorithm sided with best results from MISTA

Inst.	Best		Average		Std.Dev.		MISTA		≤ MISTA?
	TPD	TMS	TPD	TMS	TPD	TMS	TPD	TMS	
A-1	1	23	1	23	0	0	1	23	equal
A-2	2	41	2	41	0	0	2	41	equal
A-3	0	50	0	50	0	0	0	50	equal
A-4	65	42	65	42	0	0	65	42	equal
A-5	157	107	164	109	4	2	153	105	
A-6	153	98	161	102	5	3	147	96	
A-7	620	205	630	204	8	4	596	196	
A-8	300	160	323	161	6	2	302	155	yes
A-9	221	130	227	133	8	3	223	119	yes
A-10	926	324	950	327	18	3	969	314	yes
B-1	294	118	298	121	4	2	349	127	yes
B-2	474	177	482	179	9	2	434	160	
B-3	573	215	595	218	16	2	545	210	
B-4	1304	290	1333	291	19	5	1274	289	
B-5	851	256	873	262	13	4	820	254	
B-6	977	237	1035	244	31	4	912	227	
B-7	817	237	838	239	18	4	792	228	
B-8	3275	570	3361	576	74	6	3176	533	
B-9	4633	812	4782	825	93	11	4192	746	
B-10	3208	465	3284	470	40	4	3249	456	yes
X-1	408	147	417	149	6	2	392	142	
X-2	370	169	381	170	9	2	349	163	
X-3	346	199	354	199	5	3	324	192	
X-4	954	216	991	216	26	3	955	213	yes
X-5	1858	390	1883	393	24	4	1768	374	
X-6	779	249	810	254	22	5	719	232	
X-7	890	237	893	242	3	4	861	237	
X-8	1310	301	1369	306	41	5	1233	283	
X-9	3529	689	3642	703	64	7	3268	643	
X-10	1671	395	1719	404	27	7	1600	381	

There are several points in or algorithm which could be improved. Firstly, our implementation does not considers yet many optimizations described by competing teams of the MISTA Challenge in the serial schedule generation, this would speed up the entire algorithm. Secondly, the perturbation procedure could be improved to consider the history of the search process. As in some implementations of tabu search, the diversification process could consider some form of long term memory.

Nevertheless, the current implementation already outperformed our previous implementation which relied more on integer programming, showing the the combination of local search with some of our already proposed search methods can be very beneficial.

Acknowledgements The authors thank CNPq and FAPEMIG for supporting this research.

References

1. Wauters, T. Kinable, J. Smet, P. Vancroonenburg, W. Berghe, G.V. and Verstichel, J. : MISTA - Multidisciplinary International Scheduling Conference (2013). URL `http://www.schedulingconference.org/`
2. Artigues, C., Demassey, S., Néron, E.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. Scientific and Technical Publisher. Wiley (2013)
3. Asta, S., Karapetyan, D., Kheiri, A., Ozcan, E., Parkes, A.J.: Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem, technical report. Tech. rep., University of Nottingham, School of Computer Science (2014)
4. Callahan, M.T., Quackenbush, D.G., Rowings, J.E.: Construction Project Scheduling. McGraw-Hill (1992)
5. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling: A Research Handbook. Kluwer Academic Publishers, Leuven Belgium (2002)
6. Geiger, M.J.: Iterated Variable Neighborhood Search for the resource constrained multi-mode multi-project scheduling problem. In: Proceedings of the 6th Multidisciplinary International Scheduling Conference (MISTA) (2013)
7. Hansen, P., Mladenović, N.: Variable Neighborhood Search. Computers and Operations Research **24**(11), 1097–1100 (1997)
8. Hartmann, S.: A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics pp. 433–448 (2002)
9. Johnson, E., Nemhauser, G., Savelsbergh, W.: Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. INFORMS Journal on Computing **12** (2000)
10. Józefowska, J., Weglarz, J.: Perspectives in modern project scheduling. International series in operations research & management science. Springer (2006)
11. Jünger, M., Liebling, T., Naddef, D., Nemhauser, G., Pulleyblank, W., Reinelt, G., Rinaldi, G., Wolsey, L.: 50 Years of Integer Programming 1958-2008. Springer (2010)
12. Kelley Jr, J.E., Walker, M.R.: Critical-path planning and scheduling. In: Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference, IRE-AIEE-ACM '59 (Eastern), pp. 160–173. ACM, New York, NY, USA (1959). DOI 10.1145/1460299.1460318
13. Klein, R.: Scheduling of Resource-Constrained Projects. Operations research/computer science interfaces series. Kluwer Academic (2000)
14. Weglarz, J.: Project Scheduling: Recent Models, Algorithms, and Applications. International series in operations research & management science. Kluwer (1999)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Polynomially solvable formulations for a class of nurse rostering problems

Pieter Smet · Peter Brucker [⋆] · Patrick De Causmaecker · Greet Vanden Berghe

Abstract Identifying underlying structures in combinatorial optimisation problems leads to a better understanding of a problem and, consequently, to efficient solution methodologies. The present paper introduces a new network flow formulation for a large class of nurse rostering problems. By solving an integer minimum cost flow problem in a carefully constructed network, nurses' shift schedules can be constructed in polynomial time. The performance of the new formulation is compared with a state of the art algorithm on a benchmark dataset. Computational experiments show that the new formulation performs better in terms of computation time, while still solving the problem to optimality. By identifying inherent combinatorial structures which can be efficiently exploited, insight is gained into the problem's complexity, thereby laying the foundations for a theory of nurse rostering.

Keywords Nurse rostering · Network flows · Mathematical programming

[⋆] Peter Brucker sadly passed away on July 24, 2013. His coauthors dedicate their contribution in this paper to his memory.

P. Smet and G. Vanden Berghe
KU Leuven, Department of Computer Science, CODeS & iMinds - ITEC
Gebroeders De Smetstraat 1, 9000 Gent, Belgium
Tel.: +32 92658704
E-mail: {pieter.smet, greet.vandenberghe}@cs.kuleuven.be

P. De Causmaecker
KU Leuven, Department of Computer Science, CODeS & iMinds - ITEC
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
Tel.: +32 92658704
E-mail: patrick.decausmaecker@kuleuven-kortrijk.be

1 Introduction

Scheduling nurses is a critical process in health care due to the high costs associated with these scarce resources. Nearly sixty years of research has been devoted to solving different variants of this problem, resulting in an equally large variety of solution techniques (Burke et al, 2004; Van den Bergh et al, 2013). Many nurse rostering problems addressed in the literature are complex in nature, dealing with a large variety of organisational and legal constraints (Brucker et al, 2010; Smet et al, 2013). Complex search algorithms have been proposed for dealing with such problems (Burke and Curtois, 2014; Valouxis et al, 2012). However, attention has also been paid to more straightforward, simplified variants of the problem which make abstraction of a large part of the operational complexity arising in practice. Studying the underlying structure of such problems can lead to valuable insights, resulting in improved methodologies for both simplified and complex nurse rostering problems. We revisit problems presented in the literature, and investigate whether they exhibit a combinatorial structure that can be efficiently exploited.

An example of such a combinatorial structure which has been given attention in personnel scheduling research is the use of network flow techniques (Ahuja et al, 1993). A common application of network flows is found in column generation approaches for personnel rostering, where the pricing problem is often modeled as a resource constrained shortest path problem (Jaumard et al, 1998). Networks have also been described to address more general problems: to calculate the size of a workforce (Koop, 1988), to reconstruct nurse rosters from a schedule with disruptions (Moz and Pato, 2004) or to allocate shift types to a fixed days-on roster (Dowsland and Thompson, 2000). Brucker et al (2011) discuss networks for various (sub)problems related to personnel scheduling. Millar and Kiragu (1998) present a mathematical model with an underlying network structure to represent both a cyclic and non-cyclic nurse scheduling problem. Constraints regarding staffing demands and weekends are modelled as side constraints external to the network.

These network flow formulations make a strong abstraction of reality by e.g. assuming equal staffing requirements on all days or full staff availability, considering single shift scenarios and ignoring skill requirements. Furthermore, the number of contractual workforce constraints included in these formulations is typically limited to e.g. only restricting certain shift successions or only limiting the maximum number of consecutive assignments. The present paper fills the existing void by presenting a network flow model incorporating various practical and important nurse rostering constraints. Since there are no side constraints, the underlying structure of network flow problems is kept intact. Thereby, a solution methodology is established for efficiently solving a large class of nurse rostering problems.

The paper is organised as follows. Section 2 presents a detailed classification scheme for nurse rostering problems. Section 3 introduces a new network flow formulation for nurse rostering, along with several extensions. A computational evaluation of the new formulation is presented in Section 4. Finally, Section 5 concludes the paper and identifies areas for future research.

2 A classification scheme for nurse rostering problems

De Causmaecker and Vanden Berghe (2011) present an $\alpha|\beta|\gamma$ classification scheme for practical nurse rostering problems (Table 1). The presented notation allows a wide variety of problem characteristics to be described. In this section, we introduce an extension to this classification scheme which allows detailed elements of nurse rostering problems to be described.

	Personnel constraints		Skill interactions	
α Personnel environment	A	Availability	2, 3, ...	Fixed number
	S	Sequences	N	Variable number
	B	Balance	I	Individual skill definitions
	C	Chaperoning		
	Coverage constraints		Shift type	
β Work characteristics	R	Range	2, 3, ...	Fixed number
	T	Time intervals	N	Variable number
	V	Fluctuating	O	Overlapping
	Objective		Mode	
γ Optimisation objective	P	Personnel constraints	M	Multi-objective
	L	Coverage constraints		
	X	Number of personnel		
	R	Robustness		
	G	General		

Table 1 Classification of nurse rostering problems (De Causmaecker and Vanden Berghe, 2011).

α: Personnel environment

The scheme of De Causmaecker and Vanden Berghe (2011) describes time-related (horizontal) constraints by $\alpha : A$ and $\alpha : S$ for counters and series, respectively. We present the following extensions to these constraint categories:

- $A \in \{a, \overline{a}, \underline{a}, \overline{\underline{a}}, s, \overline{s}, \underline{s}, \overline{\underline{s}}\}$ Type of counter constraint. When $A = a$ there is a constraint on the number of days worked, and when $A = s$ there is a constraint on the number of assignments of a particular shift type. The lines above and under each entry indicate the type of threshold. For example, a means that an exact number of days needs to be worked, $\overline{a}$ means that only an upper bound is specified, $\underline{a}$ refers to only a lower bound, finally, $\overline{\underline{a}}$ means that a range is defined.
- $S \in \{as, \overline{as}, \underline{as}, \overline{\underline{as}}, cs, \overline{cs}, \underline{cs}, \overline{\underline{cs}}, ss\}$ Type of series constraint. When $S = as$ there is a constraint on the number of consecutive days worked, when $S = ss$ there is a constraint on particular shift successions, and when $S = cs$ there is constraint on the number of consecutive assignments of a particular shift type. The threshold for this type of constraint is defined as in category $\alpha : A$.

β: Work characteristics

To detail the type of coverage constraint in a nurse rostering problem, the category $\beta : R$ is extended with the following elements:

- $R \in \left\{d, \overline{d}, \underline{d}, \overline{\underline{d}}\right\}$ Type of coverage constraint. The threshold for this type of constraint corresponds to the threshold definition in category $\alpha : A$.

γ: Optimisation objective

Several objectives can be described in the γ category. We present an extension to the category $\gamma : P$ to differentiate between different types of personnel related objectives:

- $P \in \{\sum wc, \sum px\}$ Objective function. $P = \sum wc$ denotes a weighted sum of soft constraint violations. When $P = \sum px$, the employee preferences are optimised.

While most common time-related constraints for nurse rostering are presented in the extended classification scheme, the use of the $\alpha|\beta|\gamma$ notation presents a flexible framework allowing for future extensions.

3 Network flow models for nurse rostering

3.1 Problem description

The scheduling period T is a set of t days $T = \{1, ..., t\}$. There is a set S of s shift types $S = \{1, ..., s\}$. On each day j and for each shift type k, arbitrary minimum and maximum staffing demands $0 \leq d^l_{jk} \leq d^u_{jk}$ are specified. The workforce N is a heterogeneous set of n nurses $N = \{1, ..., n\}$. Each nurse i has to work exactly a_i days in T. Finally, each nurse i has a preference for working shift type k on day j, expressed as an inversely proportional integer cost c_{ijk}.

Let $\mathcal{P}$ denote the problem of assigning shifts to nurses such that the staffing requirements are satisfied. Each nurse must work exactly the number of specified days and can be assigned to at most one shift per day. The objective is to minimise the costs c_{ijk}.

$\mathcal{P}$ can be formulated as an integer linear program (ILP) with one set of decision variables

$$x_{ijk} = \begin{cases} 1 & \text{if nurse } i \text{ works shift } k \text{ on day } j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{P}: \ min \sum_{i \in N} \sum_{j \in T} \sum_{k \in S} c_{ijk} x_{ijk} \tag{1}$$

$$s.t. \ \sum_{k \in S} x_{ijk} \leq 1 \qquad \forall \ i \in N, j \in T \tag{2}$$

$$d^l_{jk} \leq \sum_{i \in N} x_{ijk} \leq d^u_{jk} \qquad \forall \ j \in T, k \in S \tag{3}$$

$$\sum_{j \in T} \sum_{k \in S} x_{ijk} = a_i \qquad \forall \ i \in N \tag{4}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall \ i \in N, j \in T, k \in S \tag{5}$$

The objective function 1 minimises the sum of costs incurred by the shift assignments. Constraints 2 ensure that at most one shift is assigned per day, per nurse. Constraints 3 model the minimum and maximum staffing demands. Constraints 4 restrict the number of days each nurse should work in the planning period. Finally, constraints 5 bound the decision variables.

Following the extended $\alpha|\beta|\gamma$ notation presented in Section 2, the class of problems we address is in *A(a)NI*$|$*R*($\overline{\underline{d}}$)*VN*$|$*P*($\sum$*px*).

3.2 Network flow formulation

Problem $\mathcal{P}$ can be reformulated as an integer minimum cost network flow problem in a directed network $G = (V, E)$, with V the set of nodes and E the set of arcs. The set V consists of four subsets of nodes.

Shift nodes For each day $j \in T$ and each shift type $k \in S$, a node is created representing the demand on day j for shift type k.
Time nodes For each nurse $i \in N$ and each day $j \in T$, a node is created representing a day on which a nurse can work.
Nurse nodes For each nurse $i \in N$, one node is created.
Other nodes There is one source node s and one sink node f.

Figure 1 shows the structure of the network G. Each shift node has one incoming arc from the source node. Its outgoing arcs are directed towards the time nodes corresponding to the day for which the shift node is defined. Each nurse node only has incoming arcs from time nodes associated with the nurse. Finally, each nurse node has one outgoing arc to the sink node.

Lemma 1 *The number of nodes in G is equal to* $t(s + n) + n + 2$.

Proof The network contains ts shift nodes, nt time nodes, n nurse nodes and two other nodes. □

Lemma 2 *The number of arcs in G is equal to* $t(s + n(s + 1)) + n$.

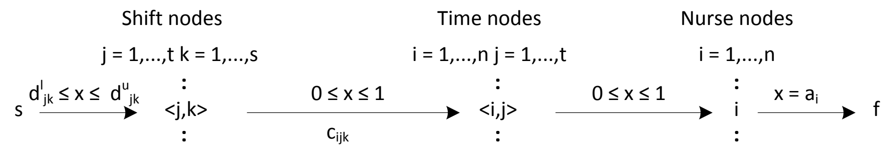


Fig. 1 Network G for problem $\mathcal{P}$. x denotes the flow through an arc.

Proof There are ts arcs going from the source node to the shift nodes. Each shift node has n arcs to time nodes. There are ts shift nodes, so in total tsn arcs go from shift nodes to time nodes. Each time node has one outgoing arc to a nurse node. With nt time nodes, nt arcs exist between the time nodes and the nurse nodes. Finally, there are n arcs between the nurse nodes and the sink node. □

Flow costs are only defined on the arcs between the shift nodes and the time nodes, representing the cost c_{ijk} of assigning a nurse i to shift type k on day j. All nodes, except the source and sink nodes, are transshipment nodes. The supply in the source node is $\sum_{i \in N} a_i$, corresponding to the total number of days the nurses can work according to their contracts. The supply in the sink node is equal to $\sum_{i \in N} -a_i$.

Lower and upper bounds on the capacity of the arcs are appropriately defined to correctly represent problem $\mathcal{P}$. The arcs between the source node and the shift nodes have a lower (upper) bound equal to the minimum (maximum) staffing demand. Arcs between the nurse nodes and sink node have a lower and upper bound equal to the required number of days worked. All other arcs require a flow of either 0 or 1.

Theorem 1 *An optimal integer minimum cost flow in the network G corresponds to an optimal solution for problem $\mathcal{P}$.*

Proof Due to the construction of network G, a minimum cost solution respecting the capacity and demand constraints can be converted to a solution for problem $\mathcal{P}$. A flow on an arc between a time node defined for nurse i, day j and a nurse node defined for nurse i, corresponds to a working day for nurse i. By forcing a flow of a_i in the arc between the nurse node i and the sink node, nurse i will work exactly a_i days. Shift assignments are determined by flows in the arcs between the shift nodes and the time nodes. A flow from the shift node associated with day j, shift k to the time node associated with day j, nurse i, corresponds to nurse i working shift k on day j, thereby incurring cost c_{ijk}. The flow conservation constraints ensure that at least d^l_{jk}, and at most d^u_{jk} units of flow will be divided among the arcs leaving the associated shift node, thereby fulfilling the staffing demands. Since there is an upper bound of one on the arcs between the shift nodes and time nodes, a nurse cannot be assigned more than one shift per day. □

3.3 Extensions

Several elements can be added to the definition of problem $\mathcal{P}$, which can also be included in the network formulation.

3.3.1 Unavailabilities

A *shift unavailability* prevents the assignment of shift type k on day j. A set of shift unavailabilities $\bar{S}_{ij} \subseteq S$ can be defined, containing the shifts for which nurse i is unavailable on day j. This is enforced by adding constraints (6) in the ILP model.

$$\sum_{k \in \bar{S}_{ij}} x_{ijk} = 0, \forall\ i \in N, j \in T \tag{6}$$

Shift unavailabilities can be modeled in network G by setting the capacity upper bound to zero on the arcs going from the shift node associated with each shift in $\bar{S}_{ij}$ to the corresponding time nodes.

This type of unavailability can also be used to include qualification requirements for particular shifts. For example, when one head nurse is required during the day shift, a dedicated *head nurse-day shift* can be created. The capacity upper bound on the arcs going from the associated shift nodes should be zero, except for the arcs to the time nodes defined for the actual head nurses.

A *day unavailability* forbids the assignment of any shift on day j. Again, for each nurse i, a set of day unavailabilities $\bar{T}_i \subseteq T$ can be defined. Constraints (7) model these unavailabilities in the ILP model.

$$\sum_{k \in S} x_{ijk} = 0, \forall\ i \in N, j \in \bar{T}_i \tag{7}$$

In the network G, day unavailabilities are enforced by changing the capacity upper bound to zero on the arcs going from the relevant time nodes to the corresponding nurse nodes.

3.3.2 Hard preferences

Hard preferences, either for working days or for particular shifts, can be modeled in a similar way as the unavailabilities. An assignment of shift k on day j for nurse i can be fixed by adding constraint (8) to the ILP model.

$$x_{ijk} = 1 \tag{8}$$

For a fixed day-on assignment on day j to nurse i, constraint (9) should be added to the ILP model.

$$\sum_{k \in S} x_{ijk} = 1 \tag{9}$$

In network G, instead of setting the capacity upper bound on selected arcs to zero, the capacity lower bound is set to one, thereby forcing a flow through the arcs and consequently ensuring a working day or shift.

3.3.3 Daily employment cost

There exist cases in which a cost c_i is incurred for each day nurse i works in the planning period. The objective function in the ILP includes an additional term to represent these costs (expression (10)).

$$\sum_{i \in N} \sum_{j \in T} \sum_{k \in S} c_{ijk} x_{ijk} + \sum_{i \in N} \sum_{j \in T} c_i \sum_{k \in S} x_{ijk} \tag{10}$$

This extension can be modeled in network G by adding a flow cost equal to c_i on the arcs from the nurse nodes to the sink node. Since each unit of flow through these arcs represents one day of labour, a flow cost corresponds to the cost c_i.

3.3.4 Ranged constraint

Problem $\mathcal{P}$ requires nurse i to work exactly a_i days. This constraint can be relaxed such that nurse i works between a_i^l and a_i^u days. In the ILP, constraints (4) are replaced by constraints (11).

$$a_i^l \leq \sum_{j \in T} \sum_{k \in S} x_{ijk} \leq a_i^u, \forall\ i \in N \tag{11}$$

This relaxation is included in the network flow model by transforming the network G to a circulation network G' by adding one arc from the sink node to the source node. There is no cost associated with this arc, and the capacity is only bounded below by zero. All nodes become transshipment nodes. According to Theorem 1, an integer minimum flow in G' again corresponds to an optimal solution for problem $\mathcal{P}$ with constraints (11).

3.3.5 Weighted constraint

Consider the modification of problem $\mathcal{P}$ such that for each nurse $i \in N$ only an upper bound a_i on the number of days worked is imposed, which can be violated at the cost of a penalty w_i per additional day worked. This is modelled by replacing constraints (4) with constraints (12) in the ILP.

$$\sum_{j \in T} \sum_{k \in S} x_{ijk} \leq a_i + p_i, \forall\ i \in N \tag{12}$$

The variable p_i represents the number of days nurse i works over the allowed maximum. Violations of this constraint are minimised by optimising objective function (13).

$$\sum_{i \in N} \sum_{j \in T} \sum_{k \in S} c_{ijk} x_{ijk} + \sum_{i \in N} w_i p_i \tag{13}$$

In the network flow formulation, one additional transshipment node is created for each nurse: a constraint node. Each of these new nodes is connected with the nurse node of the corresponding nurse, and the sink node. Both arcs have positive infinite capacity. By adding a flow cost equal to w_i on the arcs between the nurse nodes and the constraint nodes, the penalty for additional days worked is counted. Figure 2 shows the modified network.

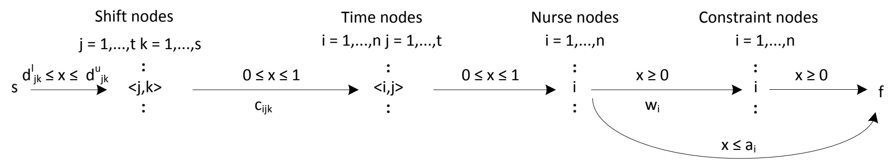


Fig. 2 Network G for problem $\mathcal{P}$ with weighted constraint violation. x denotes the flow through an arc.

To construct a solution for this extended problem description, the network G depicted in Figure 2 is first transformed to a circulation network G', such that all nodes become transshipment nodes. According to Theorem 1, an integer minimum flow in G' again corresponds to an optimal solution for problem $\mathcal{P}$ with the weighted constraint.

4 Computational analysis

4.1 Applying the network flow model

The effectiveness of the new network flow formulation is evaluated by analysing a series of computational experiments on the NSPLib benchmark dataset (Vanhoucke and Maenhout, 2007). The dataset consists of different constraint sets that can be combined with any of the 29,160 problem instances. By omitting the constraint on forbidden shift sequences from the NPSLib problem description, a subset of instances (case 1 constraint set for the 7-day instances, and case 9 for the 28-day instances) corresponds to problem $\mathcal{P}$.

As in problem $\mathcal{P}$, an assignment cost c_{ijk} is defined for each nurse, shift, day combination. Since NSPLib uses a dummy shift k' to represent a day-off, costs of working shifts are scaled relative to the cost of the dummy shift. Let $c_{ijk'}$ be the preference cost of nurse i for a day-off on day j. To correctly incorporate the use of the dummy shift in the network G, the original costs c_{ijk} are transformed to modified costs c'_{ijk} by applying equation 14.

$$c'_{ijk} = c_{ijk} - c_{ijk'} \quad \forall i \in N, j \in T, k \in S \tag{14}$$

It is clear that a problem instance with costs c'_{ijk} has the same optimal assignments as a problem with costs c_{ijk} since the relative differences in preference remain the same. The only difference is that c'_{ijk} can be less than zero, which in general does not influence (optimal) choices made by algorithms. Since after applying equation 14, the costs associated with days-off are zero, their assignment can be omitted from any shift assignment model. The costs associated with the other shifts will determine whether a particular shift type assignment is preferable to a day-off.

4.2 Performance evaluation

We first present a comparison in terms of size for both the ILP formulation and the network flow formulation (NF). Table 2 shows, for the different instances in NSPLib, the number of variables and constraints in the ILP model and the number of nodes and arcs in the network G.

		ILP		**NF**	
Days	**Nurses**	**Variables**	**Constraints**	**Nodes**	**Arcs**
7	25	700	228	230	928
	50	1400	428	430	1828
	75	2100	628	630	2728
	100	2800	828	830	3628
28	30	3360	982	984	4342
	60	6720	1852	1854	8572

Table 2 Size comparison of ILP and network flow models.

We performed a series of experiments with the ILP formulation and network flow formulation to solve instances from NSPLib. The experiments were carried out on an Intel Core i5 CPU at 2.5GHz with 4GB RAM operating on Windows 7, using a single thread. All algorithms were coded in C++. IBM ILOG CPLEX 12.5 was used to solve the ILP formulation. The network flow formulation was solved with the network simplex algorithm in LEMON 1.3.

Table 3 compares the solution costs and computation times in seconds for the ILP formulation (ILP) and the network flow formulation (NF). These values are averages over all instances, grouped per number of days and number of nurses.

Both the ILP and network flow formulations obtain optimal solutions for all instances, while requiring very little computation time. The reported calculation times are plotted in Figure 3 as functions of problem size, determined by the number of days and nurses. For both approaches, the trend shows that an increasing problem size, and thus an increasing number of variables, constraints or network dimensions, leads to longer calculation times. However, for the network flow formulation, the required calculation time is up to a magnitude lower than for the ILP formulation, thereby demonstrating the ad-

		ILP		NF	
Days	**Nurses**	**Avg. cost**	**Time(s)**	**Avg. cost**	**Time(s)**
7	25	245.41	0.0206	245.41	0.0014
	50	489.77	0.0324	489.77	0.0031
	75	740.11	0.0447	740.11	0.0062
	100	1191.19	0.0579	1191.19	0.0102
28	30	1422.32	0.0685	1422.32	0.0134
	60	2915.64	0.1406	2915.64	0.0406

Table 3 Comparison of the ILP and network flow formulations.

vantage of exploiting the problem's underlying combinatorial structure with well known efficient algorithms.

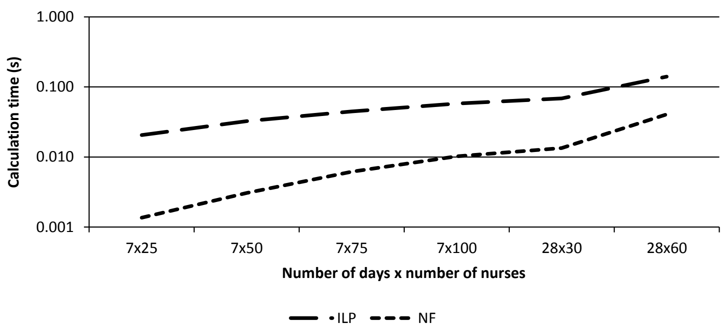


Fig. 3 Required calculation times in function of problem size.

5 Conclusions and future work

By reformulating nurse rostering problems as integer minimum cost flow problems, we identified a class of problems in $A(a)NI|R(\underline{\overline{d}})VN|P(\sum px)$ that can be solved in polynomial time. Within this problem class, several variants are introduced which can be modeled by making minor modifications to the presented flow network, while still preserving its combinatorial structure. The contribution lies in this new formulation by which a large class of problems can be solved in polynomial time.

Computational experiments demonstrated the effectiveness of the new formulation on a benchmark dataset from the literature. Compared to solving an ILP formulation with a state of the art mathematical solver, a network simplex algorithm required almost ten times less computation time for solving the integer minimum cost flow problem in the presented flow network.

The challenge of identifying efficiently exploitable combinatorial structures for more complex problems, incorporating other practical constraints, remains an important research area. Such results lead to establishing a theory of personnel scheduling which is severely lacking in the academic community. Understanding various problems' structure and complexity supports the study of more complex nurse rostering problems arising in practice.

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network Flows: Theory, Algorithms, and Applications. Prentice Hall

Brucker P, Burke EK, Curtois T, Qu R, Vanden Berghe G (2010) Adaptive construction of nurse schedules: A shift sequence based approach. Journal of Heuristics 16(4):559–573

Brucker P, Qu R, Burke EK (2011) Personnel scheduling: Models and complexity. European Journal of Operational Research 210(3):467 – 473

Burke EK, Curtois T (2014) New approaches to nurse rostering benchmark instances. European Journal of Operational Research 237(1):71 – 81

Burke EK, De Causmaecker P, Vanden Berghe G, Van Landeghem H (2004) The state of the art of nurse rostering. Journal of Scheduling 7(6):441–499

De Causmaecker P, Vanden Berghe G (2011) A categorisation of nurse rostering problems. Journal of Scheduling 14(1):3–16

Dowsland K, Thompson J (2000) Solving a nurse scheduling problem with knapsacks, networks and tabu search. Journal of the Operational Research Society pp 825–833

Jaumard B, Semet F, Vovor T (1998) A generalized linear programming model for nurse scheduling. European Journal of Operational Research 107(1):1 – 18

Koop GJ (1988) Multiple shift workforce lower bounds. Management Science 34(10):1221–1230

Millar HH, Kiragu M (1998) Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. European Journal of Operational Research 104(3):582 – 592

Moz M, Pato M (2004) Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. Annals of Operations Research 128:179–197

Smet P, De Causmaecker P, Bilgin B, Vanden Berghe G (2013) Nurse rostering: A complex example of personnel scheduling with perspectives. In: Uyar AS, Ozcan E, Urquhart N (eds) Automated Scheduling and Planning, Studies in Computational Intelligence, vol 505, Springer Berlin Heidelberg, pp 129–153

Valouxis C, Gogos C, Goulas G, Alefragis P, Housos E (2012) A systematic two phase approach for the nurse rostering problem. European Journal of Operational Research 219(2):425 – 433

Van den Bergh J, Beliën J, De Bruecker P, Demeulemeester E, De Boeck L (2013) Personnel scheduling: A literature review. European Journal of Operational Research 226(3):367 – 385

Vanhoucke M, Maenhout B (2007) NSPLib - a tool to evaluate (meta-) heuristic procedures. In: Brailsford S, Harper P (eds) Operational research for health policy: making better decisions, proceedings of the 31st meeting of the European working group on operational research applied to health services, pp 151–165

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Branch-and-Price and Improved Bounds to the Traveling Umpire Problem

**Túlio A. M. Toffolo · Sam Van Malderen ·
Tony Wauters · Greet Vanden Berghe**

Abstract The present paper proposes a branch-and-price approach to the Traveling Umpire Problem (TUP). In this hard combinatorial optimization problem, umpires (or referees) have to be assigned to games in a double round robin tournament. The objective is to obtain a solution with minimum total travel distance over all umpires, while respecting several hard constraints. Dantzig-Wolfe decomposition is applied to an existing Integer Programming formulation to be used in a branch-and-price framework. The pricing problems are solved using a specialized branch-and-bound algorithm, which applies multiple pruning techniques. Two branching strategies (best-first and depth-first) were employed and result in many improved lower bounds compared to the previous best known. In addition, five new best solutions were found and four instances with 16 teams were proven to be infeasible.

Keywords Traveling Umpire Problem · Branch and Price · Column Generation · Decomposition Strategies · Integer Programming

1 Introduction

The Traveling Umpire Problem (TUP) is a sports timetabling problem that considers the assignment of n umpires (or referees) to games in a double round robin tournament (e.g. a baseball championship). The tournament schedule is given as input with $4n - 2$ rounds (or slots), where the $2n$ teams play twice against each other; once in their home venue and once away. The objective is to minimize the total travel distance of all umpires. In order to obtain a fair assignment, several hard constraints are imposed:

$a)$ every game in the tournament is officiated by exactly one umpire.

Túlio A. M. Toffolo[1,2] · Sam Van Malderen[1] · Tony Wauters[1] · Greet Vanden Berghe[1]
[1] KU Leuven, Department of Computer Science, CODeS & iMinds-ITEC
[2] Federal University of Ouro Preto - Brazil, Department of Computing
Emails: tulio.toffolo@kuleuven.be, sam.vanmalderen@cs.kuleuven.be, tony.wauters@cs.kuleuven.be and greet.vandenberghe@cs.kuleuven.be

$b)$ every umpire must work in every round.
$c)$ every umpire should visit the home of every team at least once.
$d)$ no umpire is in the same venue more than once in any $q_1 = n - d_1$ consecutive rounds.
$e)$ no umpire officiates a game with the same team more than once in any $q_2 = \lfloor \frac{n}{2} \rfloor - d_2$ consecutive rounds (this constraint is similar to the previous one, but also takes the 'away team' into consideration).

The values d_1 and d_2 range from 0 to n and 0 to $\lfloor \frac{n}{2} \rfloor$, respectively.

The Traveling Umpire Problem (TUP) was first introduced by Trick and Yildiz (2007). Their work was extended by Trick and Yildiz (2011), where a Benders cuts guided large neighborhood search is proposed. These papers also provide an Integer Programming (IP) and Constraint Programming (CP) formulation for the problem. A greedy matching heuristic and a simulated annealing approach with a two-exchange neighbourhood are described by Trick et al (2012). Trick and Yildiz (2012) present a Genetic Algorithm (GA) with a locally optimized crossover procedure. A stronger IP formulation and a relax-and-fix heuristic are proposed in (de Oliveira et al, 2013), which improve both lower and upper bounds. Wauters et al (2014) present an enhanced iterative deepening search with leaf node improvements (IDLI), an iterated local search (ILS) and a new decomposition based lower bound methodology. Many improved solutions and lower bounds were found.

In this work, we present a branch-and-price approach to the TUP. By applying the Dantzig-Wolfe decomposition on an existing formulation of the problem, we obtain a Restricted Master Problem (RMP) and pricing subproblems. The RMP is a set partition problem, and its relaxation can be solved by linear programming algorithms such as Simplex. The pricing subproblems are solved by a specialized branch-and-bound. The branch-and-price can be seen as a branch-and-bound employing a column generation scheme to solve the relaxation in each node. We considered two branching and node selection strategies, one for improving the lower bounds and another for obtaining feasible solutions.

The following section presents the formulation introduced by de Oliveira et al (2013) for the TUP. Section 3 details the reformulation of the original model. The strategies considered in the branch-and-price framework are discussed in Section 4. Section 5 presents computational experiments considering both lower and upper bounds and, finally, Section 6 summarizes the conclusions and proposes future work.

2 Integer Programming Formulation for the TUP Problem

The first formulation for the TUP was proposed by Trick and Yildiz (2007). This formulation was then improved by de Oliveira et al (2013). We apply the Dantzig-Wolfe decomposition on the latter model. Following, we present this formulation. For that, consider the following input data:

U :	set of umpires, such that $U = \{1, ..., n\}$;
T :	set of teams, such that $T = \{1, ..., 2n\}$;
R :	set of rounds, such that $R = \{1, ..., 4n - 2\}$;

$H(r)$:	set of teams i hosting a game in round r;
$\delta(i)$:	set of rounds $r \in R$ in which the team i is hosting a game;
$A(i,r)$:	function that returns the team playing against team i in round r;
d_{ij} :	distance between the home of teams i and j;
$CV(r)$:	set of rounds $\{r, \dots, r+q_1-1\} \in R$ defined for $r \in \{1, \dots, \lvert R\rvert - q_1 - 1\}$;
$CT(r)$:	set of rounds $\{r, \dots, r+q_2-1\} \in R$ defined for $r \in \{1, \dots, \lvert R\rvert - q_2 - 1\}$;

The decision variables are:

$$x_{ijru} = \begin{cases} 1 & \text{if umpire } u \text{ is assigned to venue } i \text{ in round } r \text{ and to } j \text{ in round } r+1 \\ 0 & \text{otherwise} \end{cases}$$

The formulation is presented by constraints (1)-(9).

$$\text{min} \qquad \sum_{i \in T} \sum_{j \in T} \sum_{r \in R} \sum_{u \in U} d_{ij} x_{ijru} \tag{1}$$

$$\text{s.t.} \qquad \sum_{u \in U} \sum_{j \in T} \mathrm{x}(i,j,r,u) = 1 \quad \forall i \in T,\ r \in \delta(i) \tag{2}$$

$$\sum_{i \in H(r)} \sum_{j \in T} \mathrm{x}(i,j,r,u) = 1 \quad \forall r \in R,\ u \in U \tag{3}$$

$$\sum_{r \in \delta(i)} \sum_{j \in T} \mathrm{x}(i,j,r,u) \geq 1 \quad \forall i \in T,\ u \in U \tag{4}$$

$$\sum_{\substack{c \in CV(r): \\ c \in \delta(i)}} \sum_{j \in T} \mathrm{x}(i,j,c,u) \leq 1 \quad \begin{array}{l} \forall i \in T,\ u \in U,\ r \in R: \\ r < \lvert R\rvert - q_1 - 1 \end{array} \tag{5}$$

$$\sum_{c \in CT(r)} \sum_{j \in T} \left(\mathrm{x}(i,j,c,u) + \sum_{\substack{k \in T: \\ A(k,c)=i}} \mathrm{x}(k,j,c,u) \right) \leq 1 \quad \begin{array}{l} \forall i \in T,\ u \in U,\ r \in R: \\ r < \lvert R\rvert - q_2 - 1 \end{array} \tag{6}$$

$$\sum_{j \in T} \mathrm{x}(i,j,r,u) = 0 \quad \begin{array}{l} \forall i \in T,\ u \in U, \\ r \in R \backslash \delta(i) \end{array} \tag{7}$$

$$\sum_{j \in T} x_{jiru} - \sum_{j \in T} x_{ij(r+1)u} = 0 \quad \begin{array}{l} \forall i \in T,\ u \in U,\ r \in R: \\ r < \lvert R\rvert - 1 \end{array} \tag{8}$$

$$x_{ijru} \in \{0,1\} \quad \begin{array}{l} \forall i \in T,\ j \in T,\ r \in R, \\ u \in U \end{array} \tag{9}$$

where

$$x(i,j,r,u) = \begin{cases} x_{ijru} & \text{if } r = 1 \\ x_{ji(r-1)u} & \text{otherwise} \end{cases}$$

Constraints (2) and (3) ascertain that every game can be officiated by only one umpire and that every umpire may officiate only one game per round, respectively. Constraints (4) state that every umpire should visit every team at least once during the season. Constraints (5) and (6) specify that every umpire must wait $q_1 - 1$ days to revisit the same home location and that every umpire must wait $q_2 - 1$ days to revisit the same team, respectively. Constraints (7) enforce that an umpire can only travel to the home location of a team if that team hosts a game in that certain round. If an umpire is at the location of a team in round r, the umpire must leave from the same location in round $r + 1$ as specified by constraints (8). Finally, the objective function, given by equation (1), is to minimize the total travel distances of the umpires.

3 Dantzig-Wolfe Reformulation

In order to obtain stronger bounds, we reformulate the model presented in the previous section by applying the Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960). The original problem is decomposed into a master problem and n pricing problems, one for each umpire.

Figure 1 shows the structure of the linear program (LP) of a TUP instance considering the formulation presented in Section 2. This figure presents the coefficient matrix of the original LP (left image) and the same LP after sorting the rows and columns by umpire (right image). The dots indicate non-zero coefficients in the constraint matrix. The required block structure for the Dantzig-Wolfe decomposition is easily identified in the right image (sorted model). In this image, each square block forms a pricing problem containing the constraints and variables corresponding to a single umpire.

In formulation (1)-(9), constraints (3)-(9) are umpire-oriented and form the pricing problems. The remaining constraints, given by equation (2), are the coupling (or linking) constraints. These constraints correspond to the wide block at the bottom of the sorted LP in Figure 1.

The pricing problem can be stated as the problem of finding the optimal schedule for one umpire with the consideration of dual costs.

The master problem is a set partition problem, whose formulation is given by equations (10)-(13). In this formulation, Ω is the set of columns (possible schedules for the umpires), Ω_u represents the subset of Ω containing all columns of umpire $u \in U$, d_s is the cost (travel distance) of column $s \in \Omega$, λ_s is a binary variable that indicates whether the column $s \in \Omega$ is selected or not and, finally, a_{irs} is a binary coefficient denoting whether the umpire is assigned to game hosted by team $i \in T$ in round $r \in R$ in column $s \in \Omega$. Constraints (11) guarantee that only one column is chosen per umpire while constraints (12) are the coupling constraints inherited from the original problem (2), and ensure that each game in each round is officiated by exactly one umpire.

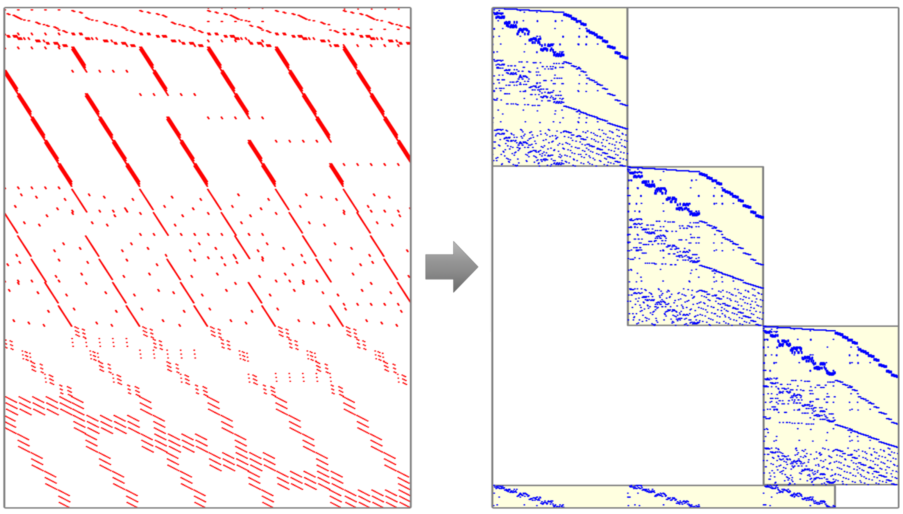


Fig. 1 Representation of the applied decomposition

$$\begin{aligned}
\text{min} \quad & \sum_{u\in U}\sum_{s\in \Omega_u} d_s\lambda_s & & (10)\\
\text{s.t.} \quad & \sum_{s\in \Omega_u} \lambda_s = 1 & \forall u \in U & \quad (11)\\
& \sum_{u\in U}\sum_{s\in \Omega_u} a_{irs}\lambda_s = 1 & \forall i \in T,\ \forall r \in R & \quad (12)\\
& \lambda_s \in \{0,1\} & \forall u \in U,\ \forall s \in \Omega_u & \quad (13)
\end{aligned}$$

The column generation approach (Lübbecke and Desrosiers, 2005; Vanderbeck and Wolsey, 2010) is solved iteratively. The linear relaxation of the master problem is solved first. In every iterations, the pricing problems are solved to obtain reduced cost columns. A reduced cost column for umpire u is a column $s \in \Omega_u$ for which $v_u + \sum_{i\in T}\sum_{r\in R} a_{irs}w_{ir} > d_s$, where v_u and w_{ir} represent the dual variables corresponding to constraints (11) and (12), respectively. If such columns are found, they are added to the master problem, which is subsequently re-solved. The algorithm continues until no reduced cost columns exists, in which case the relaxation of the reduced master problem is solved.

3.1 Symmetry breaking

In order to speed up the pricing solver, we preallocate the games assigned to the umpires in the first round. This reduces symmetry (Yildiz, 2008) in the original problem, as otherwise the umpires would have similar coefficients in the constraint matrix. Preallocation can be easily achieved by adding constraints (14) to the formulation (1)-(9). In these constraints, we use the notation $H_k(r)$ to represent the home team of the k-th game of round r, with the games in lexicographical order.

$$\sum_{j \in T} x_{ij1u} = 1 \quad \forall u \in U, i = H_u(1) \tag{14}$$

Constraints (14) are umpire-oriented and can be included in the pricing problems of the column generation scheme. Adding these constraints results in a reduction of the pricing problem size by one round.

3.2 Specialized pricing problem solver

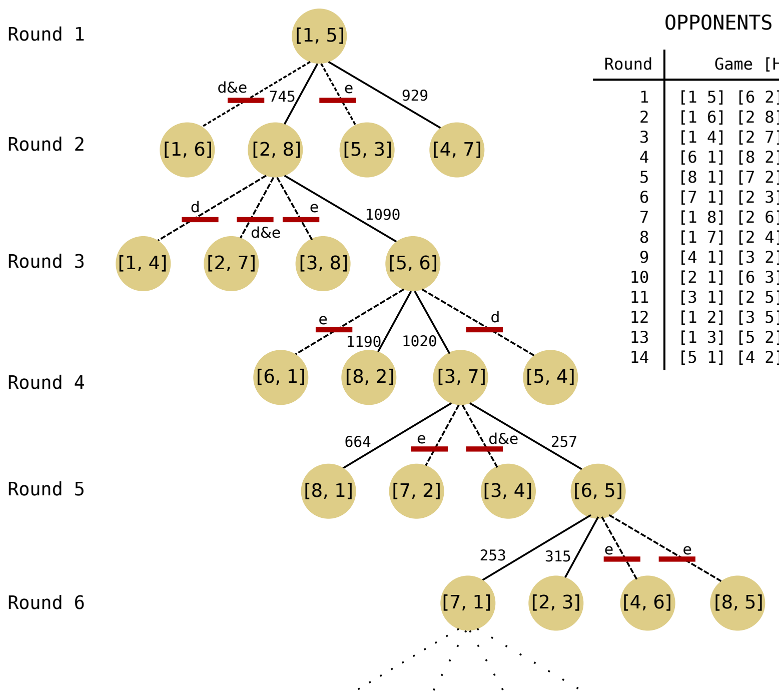
A branch-and-bound pricing solver is used to generate a predefined maximum number c of reduced cost columns. Starting in the first round, the algorithm assigns games to the umpire, round after round until the last round. An assigment of a game to an umpire in a round is feasible if (i) the umpire did not visit the same location in the previous $q_1 - 1$ rounds and (ii) the umpire did not officiate any of the teams during the previous $q_2 - 1$ rounds. Whenever multiple games can be assigned in a round, the algorithm chooses the assignment incurring the smallest increase in the travel distance.

Figure 2 shows an example of the branch-and-bound procedure for an 8-team (4-umpire) problem instance. The table inside the figure shows the considered game schedule (opponents matrix). The example considers the pricing for the first umpire using parameter values $q_1 = 4$ and $q_2 = 2$. As detailed in section 3.1, the assignment in the first round is fixed.

The umpire can neither officiate game `[5,3]` nor game `[1,6]` in the second round due to constraint e (presented in section 1), since a game played by teams 1 and 5 has already been officiated by the umpire during the first round. Moreover, the umpire cannot officiate game `[1,6]` due to constraint d, since the home location of team 1 has already been visited in the previous round. The only possibilities left in round two are game `[2,8]` and game `[4,7]`. The branch-and-bound prefers to assign the umpire to game `[2,8]` because the travel distance between the home location of teams one and two is smaller than the distance between the home locations of teams one and four.

If no valid assignment can be found in a certain round, the procedure returns to the previous round and chooses the game with the second smallest travel distance. This procedure continues until a valid assignment has been found in the last round. If the resulting solution does not violate constraint c, it is feasible and serves as an upper bound for pruning when exploring the rest of the search tree.

We implemented several extensions to improve the performance of the branch-and-bound algorithm. In section 3.2.1, we explain these pruning strategies.



OPPONENTS MATRIX

Round	Game [Home Away]
1	[1 5] [6 2] [7 3] [4 8]
2	[1 6] [2 8] [5 3] [4 7]
3	[1 4] [2 7] [3 8] [5 6]
4	[6 1] [8 2] [3 7] [5 4]
5	[8 1] [7 2] [3 4] [6 5]
6	[7 1] [2 3] [4 6] [8 5]
7	[1 8] [2 6] [4 3] [7 5]
8	[1 7] [2 4] [3 6] [5 8]
9	[4 1] [3 2] [5 7] [6 8]
10	[2 1] [6 3] [4 5] [8 7]
11	[3 1] [2 5] [8 4] [7 6]
12	[1 2] [3 5] [7 4] [8 6]
13	[1 3] [5 2] [6 4] [8 7]
14	[5 1] [4 2] [8 3] [6 7]

Fig. 2 Specialized branch-and-bound example for a 8-team (4-umpire) problem instance.

3.2.1 Pruning the search tree

Multiple strategies exist to prune unfavorable parts of the search tree. First of all, the branch-and-bound algorithm prunes the parts of the search tree where no optimal solution can reside based on the lower and upper bound on the travel distance. Once the branch-and-bound algorithm has obtained a feasible solution, it can be used as an upper bound on the minimum travel distance of the umpire.

For each game in every round, a shortest path exists to any of the games in the last round. The shortest path serves as a lower bound for the branch-and-bound procedure.When trying to assign a game in a round, the algorithm evaluates whether the current travel distance together with the lower bound exceeds the currently best known upper bound. If so, the branch-and-bound need not consider that assignment anymore, since it will not improve the current upper bound.

It is impossible to evaluate constraint c before a complete path has been generated for the umpire. Nevertheless, a second pruning strategy is possible. If in a certain round, the number of unvisited home locations exceeds the remaining number of rounds, it is impossible to obtain a solution satisfying constraint c, given the assignments in the previous rounds. The branch-and- bound algorithm should therefore return to a previous round and explore other assigments.

4 Branch-and-Price

Section 3 presented the column generation scheme. Since this algorithm only solves the LP-relaxed version of the problem, it may be necessary to branch on the variables to find an integer solution. In that case, we apply branch-and-price (Barnhart et al, 1998; Vanderbeck, 2000), which is a variant of branch-and-bound where the relaxation is solved by column generation in each node of the search tree.

The branch-and-price algorithm branches on the variables x_{ijru} from the original formulation. Since the branching tree is too large, we consider two different strategies for branching, each one pursuing a different goal. The first branching strategy aims at providing good lower bounds by conducting a best-first search (BFS) in the branching tree. The second strategy executes depth-first search (DFS) and focuses on finding integral solutions.

In each iteration, the BFS strategy selects variables for branching based on the following criterion: variables with the most fractional value of the earliest available round are selected first. Fixing variables of the earliest available round impacts the performance of the pricing solver considerably. The specialized branch-and-bound constructs the solution from the first to the last round, in lexicographical order. Hence, if a variable of the last round were selected first, processing time may be wasted searching in infeasible subtrees. Since the fixation of a variable renders several subtrees infeasible, it is better to detect the infeasibility as soon as possible during the branching process. If this strategy were not used, the detection of infeasible subtrees would be delayed, consuming a considerable amount of processing time.

The DFS strategy aims to obtain feasible solutions as soon as possible. Therefore, in each node the variable with least fractional value of the earliest possible round is selected to be branched first. By doing so in a depth-first search manner, the fixations are directed to iteratively build a feasible solution using the information provided by the column generation.

5 Computational Experiments

The approach applies SCIP/GCG (Achterberg, 2009). This open source framework provides a well structured platform for developing branch-and-price algorithms. The branching scheme, node rules and pricing solver were coded in Java, using Java Native Interface to exchange information between Java and C. CPLEX was used to solve the linear relaxation of the Restricted Master Problem.

The experiments were executed on a Intel(R) Xeon(R) CPU E5-2650 @ 2.60GHz computer with 128Gb of RAM memory running Linux Mint 16. CPLEX version 12.6 and Java Virtual Machine 1.7 were used.

The benchmark set used within the experiments is available online [1], together with the currently best known solution values in literature. We also consider the most recent bounds found by Wauters et al (2014) for comparison. Further information about the instances can be found in (Trick et al, 2012).

[1] `http://mat.gsia.cmu.edu/TUP/`, last accessed June 8, 2014

The discussion of experiments focuses on two evaluations. First we consider the dual bounds obtained by the BFS branching scheme in the branch-and-price. Afterwards we present the feasible solutions obtained by the DFS branching strategy. In both situations, we compare our results with the best known in the literature.

5.1 BFS Strategy

The results of the branch-and-price with the BFS strategy are presented in Table 1. This table shows the lower bound (LB_0) obtained with the column generation (in the root node), the best lower bound obtained (LB) and the best lower bound found in the literature. The table also shows the time in which the lower bound was found by the branch-and-price algorithm. The last column presents the gap between the best known bound and the obtained bound. The cells marked with $\circledast$ indicate some improvement over the best known bound. Considering the imposed time limit of 3 hours, no feasible solutions were found for any of the instances with more than 10 teams with the BFS strategy. We omitted the results for the small instances, as they can be easily solved to optimality in few seconds.

Table 1 shows that the column generation approach already improves 8 best known lower bounds. By applying the branch-and-price with BFS, 15 other instances have their best known dual bound improved. This result corroborates the expected strong bound from column generation.

Table 1 also shows the influence of the pricing solver on the total processing time of the column generation approach. Consider, for example, the difference in time required for solving the column generation in the root bound (given by column LB_0) for instances '16A-7,2' and '16A-7,3'. Column generation for instance '16A-7,2' required much more computation time than for '16-A-7,3'. This is mainly due to the value $q_2 = 2$ in the first instance, which is less constrained than the second one, with $q_2 = 3$. Small values of q_2 negatively impact the performance of our specialized branch-and-bound, since it provides fewer pruning opportunities.

5.2 DFS Strategy

The results of the branch-and-price with the DFS strategy are presented in Table 2. This table shows the value of the first feasible solution found (UB_0), the best solution found by the branch-and-price (UB) and the best solution in the literature. The table also shows the total runtime to find the solutions with the branch-and-price and the gap between the best known solution and the obtained solution. The experiments were restricted to 3 hours of processing time. As in Table 1, cells marked with $\circledast$ indicate some improvement over the best known solution. It is important to note that the bounds for these instances have been updated repeatedly over the years. Best bounds were hard to trace.

Table 2 shows that the DFS strategy provides feasible solutions in a small amount of time for most instances. Even considering the total available runtime, the branch-and-price was able to improve five upper bounds. For 7 instances, the branch-and-price was not able to produce feasible solutions within the time limit.

Table 1 Experiments with the BFS strategy in branch-and-price

Inst.	q_1, q_2	Bounds		Time (seconds)		LB*	gap(%)
		LB_0	LB	LB_0	LB		
14	7, 3	156439.3	157812.8	42.1	10500.0	159797	1.24
14	6, 3	154439.9	155570.4	40.8	10560.0	156551	0.63
14	5, 3	152941.3	$\circledast$ 153759.6	50.3	10740.0	153066	-0.45
14A	7, 3	149992.7	151243.5	40.7	10740.0	153199	1.28
14A	6, 3	148168.7	149285.4	45.5	10680.0	150998	1.13
14A	5, 3	147097.5	147966.4	47.8	10620.0	148299	0.22
14B	7, 3	149767.0	$\circledast$ 151165.8	43.5	10620.0	151059	-0.07
14B	6, 3	148243.9	149208.6	49.6	10620.0	149267	0.04
14B	5, 3	146846.2	$\circledast$ 147638.3	55.7	10800.0	147534	-0.07
14C	7, 3	148613.2	150101.6	44.6	10380.0	151581	0.98
14C	6, 3	146774.6	147820.0	47.7	10320.0	148728	0.61
14C	5, 3	145794.4	146622.1	49.5	10620.0	146764	0.10
16	8, 4	184187.6	$\circledast$ 193457.1	172.0	10260.0	185939	-3.89
16	8, 2	$\circledast$ 155045.2	$\circledast$ 155045.2	7092.0	7092.0	151481	-2.82
16	7, 3	158257.4	$\circledast$ 158586.0	10500.0	10500.0	158480	-0.07
16	7, 2	$\circledast$ 148341.8	$\circledast$ 148341.8	10102.0	10102.0	147138	-0.81
16A	8, 4	$\circledast$ 198969.7	$\circledast$ 200648.5	172.0	10260.0	185119	-11.28
16A	8, 2	$\circledast$ 166575.5	$\circledast$ 166624.1	5403.0	10410.0	162788	-2.77
16A	7, 3	170575.1	172420.1	371.0	10560.0	172964	0.31
16A	7, 2	161571.2	161571.2	7476.0	7476.0	161640	0.04
16B	8, 4	207505.4	$\circledast$ 209346.5	202.0	10440.0	208418	-4.55
16B	8, 2	$\circledast$ 169363.4	$\circledast$ 170092.6	5162.0	10162.0	167768	-1.37
16B	7, 3	170632.5	172058.0	880.0	10560.0	173023	0.56
16B	7, 2	163539.7	163649.6	9021.2	11298.3	164012	0.29
16C	8, 4	$\circledast$ 200682.6	$\circledast$ 205643.8	234.0	10380.0	188561	-8.31
16C	8, 2	$\circledast$ 168783.6	$\circledast$ 168783.6	7380.0	7380.0	166001	-1.77
16C	7, 3	171216.0	$\circledast$ 171767.6	449.0	10740.0	171377	-0.23
16C	7, 2	$\circledast$ 163850.8	$\circledast$ 163850.8	10578.0	10578.0	163305	-0.33

Considering that the developed approach tends to perform better on more constrained instances, one would probably expect better results for the very constrained '16-8,4', '16A-8,4', '16B-8,4' and '16C-8,4' instances. The branch-and-price was not able to find any feasible solution after several hours of processing time. This result, coupled with the fact that there are no known solution for these instances, motivated us to investigate the strong indication that they may be infeasible. In the next section we discuss over the infeasibility of these instances.

5.3 A note on the feasibility of TUP instances

It was already shown that TUP instances with $q1 > n$ and $q2 > \lfloor \frac{n}{2} \rfloor$ are infeasible (Yildiz, 2008). Instance '12-6,3' was also proven infeasible. This instance belongs to the special class of TUP instances with constraint values $q1 = n$ and $q2 = \lfloor \frac{n}{2} \rfloor$, further denoted as $\mathbf{TUP^0_0}$, referring to $\mathbf{TUP^{d_1}_{d_2}}$ with $d_1 = 0$ and $d_2 = 0$. Other instances from $\mathbf{TUP^0_0}$ with $n \leq 7$ were shown to contain at least one feasible solution. No feasible solution was found for $\mathbf{TUP^0_0}$ instances with $n > 7$.

An adapted version of the branch-and-bound procedure presented in Section 3.2, considering all umpires simultaneously, enables proving that instance '16-8,4', be-

Table 2 Experiments with the DFS strategy in branch-and-price

Inst.	q_1, q_2	Bounds		Time (seconds)		UB*	gap(%)
		UB_0	UB	UB_0	UB		
6	3, 1	14077	14077	2.0	2.0	14077	0.00
6A	3, 1	16918	15457	1.3	1.8	15457	0.00
6B	3, 1	16716	16716	2.2	2.2	16716	0.00
6C	3, 1	14396	14396	1.8	1.8	14396	0.00
10	5, 2	49154	48942	7.6	8.1	48942	0.00
10A	5, 2	46551	46551	7.6	7.6	46551	0.00
10B	5, 2	45609	45609	5.7	5.7	45609	0.00
10C	5, 2	43149	43149	6.9	6.9	43149	0.00
14	7, 3	174373	166942	84.0	546.6	164440	1.50
14	6, 3	163488	159808	88.7	2462.4	159505	1.67
14	5, 3	156406	$\circledast$ 155392	99.9	1214.8	155439	-0.03
14A	7, 3	172737	160856	91.4	7500.1	158760	1.30
14A	6, 3	160599	154637	89.7	2813.5	153216	0.92
14A	5, 3	157249	150386	933.6	4110.2	149331	0.07
14B	7, 3	170180	162677	88.3	1560.1	157884	2.95
14B	6, 3	164212	155817	94.0	5662.4	152740	1.97
14B	5, 3	154425	149866	100.8	1579.2	149621	0.16
14C	7, 3	173962	159815	87.2	6071.9	154913	3.07
14C	6, 3	155918	152696	93.2	6877.4	150858	1.20
14C	5, 3	155218	$\circledast$ 149482	108.6	9218.9	149662	-0.12
16	8, 4	-	-	-	-	-	-
16	8, 2	162720	161999	6612.6	9918.9	160705	0.80
16	7, 3	176576	170293	950.8	7799.8	168860	0.84
16	7, 2	-	-	-	-	153978	-
16A	8, 4	-	-	-	-	-	-
16A	8, 2	175796	$\circledast$ 171882	7065.8	8016.7	172966	-0.63
16A	7, 3	190715	187686	1020.5	2979.9	179960	4.12
16A	7, 2	165931	165766	9562.6	9759.0	164620	0.69
16B	8, 4	-	-	-	-	-	-
16B	8, 2	189564	$\circledast$ 180728	9283.3	10717.5	180888	-0.09
16B	7, 3	192188	186429	1176.3	1378.1	181565	2.61
16B	7, 2	-	-	-	-	170194	-
16C	8, 4	-	-	-	-	-	-
16C	8, 2	191461	$\circledast$ 179939	8949.0	9285.8	180221	-0.16
16C	7, 3	191859	187310	822.4	2234.5	184181	1.67
16C	7, 2	-	-	-	-	169184	-

longing to $\mathbf{TUP}_0^0$, has no feasible solution. At the same time, instance '16A-8,4', instance '16B-8,4' and instance '16C-8,4' are proven to be infeasible since their opponents matrix is equal to that of '16-8,4'. Table 3 summarizes the feasibility of the instances from $\mathbf{TUP}_0^0$ up to $n = 10$. The table reports for each instance, the feasibility, the number of nodes and time (in milliseconds) needed by the branch-and-bound to prove (in)feasibility. The feasibility of instances from $\mathbf{TUP}_0^0$ with $n > 8$ remains unknown after 48 hours of running time of the branch-and-bound algorithm.

6 Conclusions and future work

This work introduced a branch-and-price approach to the Traveling Umpire Problem, devoting attention to both computation of strong dual bounds and production of good

Table 3 Feasibility of instances from $\mathbf{TUP}_0^0$ up to $n = 10$, and the number of nodes and time (ms) needed to prove (in)feasiblity.

Inst.	q_1, q_2	Feasibility	Nodes	Time(ms)
4	2,1	feasible	12	1
6	3,1	feasible	35	2
8	4,2	feasible	$1,129$	5
10	5,2	feasible	$27,179$	63
12	6,3	infeasible	$901,228$	309
14	7,3	feasible	$172,552$	77
16	8,4	**infeasible**	$35,696 \times 10^6$	$3h$
18	9,4	unknown	-	$48h$
20	10,5	unknown	-	$48h$

feasible solutions. We presented a pricing solver and branching rules, optimized to speed up the resolution of the pricing problem.

We were able to improve several lower bounds. In addition, five improved solutions have been obtained. We also employed a modified version of the pricing solver to prove infeasibility of some instances. Tight runtime limits are sufficient for the branch-and-price to generate competitive feasible solutions for the most constrained problem instances.

As suggestions for future work, we point at the development of heuristics that use the information from the column generation approach to produce good solutions. In addition, other branching rules and approaches to speed up the resolution of the pricing problem may yield further improvements.

7 Acknowledgements

Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX (http://comex.ulb.ac.be).

References

Achterberg T (2009) Scip: Solving constraint integer programs. Mathematical Programming Computation 1(1):1–41

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: column generation for solving huge integer programs. Operations Research 46:316–329

Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. Operations Research 8(1):101–111

Lübbecke ME, Desrosiers J (2005) Selected Topics in Column Generation. Operations Research 53(6):1007–1023

de Oliveira L, de Souza CC, Yunes T (2013) Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic. European Journal of Operational Research In press

Trick MA, Yildiz H (2007) Bender's cuts guided large neighborhood search for the traveling umpire problem. In: Van Hentenryck P, Wolsey L (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, no. 4510 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp 332–345

Trick MA, Yildiz H (2011) Benders' cuts guided large neighborhood search for the traveling umpire problem. Naval Research Logistics (NRL) 58(8):771 – 781

Trick MA, Yildiz H (2012) Locally optimized crossover for the traveling umpire problem. European Journal of Operational Research 216(2):286 – 292

Trick MA, Yildiz H, Yunes T (2012) Scheduling major league baseball umpires and the traveling umpire problem. Interfaces 42:232 – 244

Vanderbeck F (2000) On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Operations Research 48(1):111–128

Vanderbeck F, Wolsey L (2010) Reformulation and decomposition of integer programs. In: Jünger M, Liebling TM, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA (eds) 50 Years of Integer Programming 1958-2008, Springer Berlin Heidelberg, pp 431–502

Wauters T, Malderen SV, Vanden Berghe G (2014) Decomposition and local search based methods for the traveling umpire problem. European Journal of Operational Research

Yildiz H (2008) Methodologies and applications for scheduling, routing & related problems. PhD thesis, Carnegie Mellon University

Extended Abstracts

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A self-generating memetic algorithm for examination timetabling

**Cevriye Altıntaş · Shahriar Asta ·
Ender Özcan · Tuncay Yiğit**

1 Introduction

Timetabling problems encompasses educational timetabling, nurse rostering, transportation timetabling and so on. The educational timetabling is one of the most widely studied timetabling problems including high school timetabling, university course and examination timetabling. In this study, we focus on the examination timetabling problem, which is one of the most important and repetitive administrative activities that occur in the educational institutions. In the last ten years or so, many methodologies have been developed to solve the examination timetabling problem. An examination timetabling problem consists of the designation of a set of exams to a given set of timeslots subject to various practical constraints. The generated timetable must satisfy all the hard constraints of a problem is called a feasible timetable. The hard constraints can not be violated. Soft constraints represent preferences that can be violated, but in many cases solution approaches attempt to reduce the number of such violations as much as they can to improve the quality of a generated timetable further. More on examination timetabling can be found in Qu et al. 2009 [13]. This study presents a self-generating multimeme algorithm for solving an examination timetabling problem at Suleyman Demirel University (SDU). Unlike previous multimeme algorithms, each meme in the proposed algorithm encodes a score as a performance indicator of the associated operator. Those scores are then used in the process of choosing operators to create/modify

C.Altıntaş and T.Yiğit
Suleyman Demirel University, Department of Computer Engineering
Tel.: +90-246-211 1378
Fax: +90-246-211 1378
E-mail: cevriyealtintas,tuncayyigit@sdu.edu.tr

S. Asta and E. Özcan
University of Nottingham, School of Computer Science
Jubilee Campus, Nottingham NG8 1BB UK
E-mail: sba,exo@cs.nott.ac.uk

new candidate solutions, self-adaptively. The results obtained on some SDU and ITC2007 problem instances indicate that the proposed approach performs reasonably well.

2 Problem Definition

Suleyman Demirel University (SDU), located in Turkey also deals with the examination timetabling issue a couple of times in a year. This problem is not that different than the examination timetabling problems faced by the other educational institutions across the world. Recently, the second International Timetabling Competition (ITC 2007) was organised [8] with the goal of providing a set of real world problem instances and determining the state-of-the-art for educational timetabling. One of the competition tracks was on examination timetabling and the instances used at the competition turned into a benchmark. SDU examination timetabling problem is formulated in the same way as in the ITC2007. The SDU problems instances used in this study will be publicly provided extending the ITC2007 benchmark instances. The properties of each SDU instance is summarised in Table 1. SDU instances do not have any room hard constraints.

Table 1 The characteristics of the SDU examination timetabling problem instances, where HC indicates the number of hard constraints and Density is the conflict density in percentage.

Problem	Density	Students	Exams	Rooms	Periods	Period HC
SDU01	3.24	10953	212	17	50	142
SDU02	5.08	11012	236	26	61	123
SDU03	1.37	24867	430	29	80	317
SDU04	12.60	8028	166	18	59	61
SDU05	3.59	12091	269	33	46	173

3 Proposed Approach

A generic *Memetic Algorithm* is an evolutionary algorithm which makes heavy use of hill climbing as introduced by Moscato in [9]. The main components of an MA are mutation, crossover and hill climbing. In this study, we describe a novel “Self-Generating Multimeme Algorithm” (SGMA) that manages 6 mutation, 2 crossover and 2 hill climbing operators. The initial population is formed using multiple constructive heuristics with the goal of generating feasible initial solutions. The main feature of the proposed algorithm is that each meme encodes a score as a performance indicator of the associated operator. During the evolutionary process, when it is time to apply an operator of certain type, e.g., mutation, one of the operators is selected and employed randomly using roulette wheel selection based on the scores of operators of that type.

Table 2 Best result obtained from SGMA for each SDU instance.

Problem	SDU01	SDU02	SDU03	SDU04	SDU05
Score	760	5880	210	20000	30

4 Experimental Results

The performance of a self-generating multimeme algorithm for the examination timetabling problem is investigated on a subset of ITC2007 and SDU instances. Each experiment is repeated 10 times and a run is terminated after 325 seconds complying with the ITC2007 competition rules. We have used a 2 Core Duo 3.16 GHz (2 GB RAM) machine during our experiments. Feasible solutions are obtained for all problem instances used during the experiments. Table 2 provides the best results obtained by SGMA over 10 runs for the SDU instances. Similarly, Table 3 presents a comparison between our approach and some selected previously proposed approaches on six ITC2007 benchmark instances based on the best result that each approach achieves. SGMA performs reasonably well in the overall. It is not the best approach, but it performs potentially better than some other memetic approaches [11]. We will be implementing different types of memetic algorithms and testing them on all ITC2007 and SDU instances as future work.

References

1. Abdul-Rahman, S., Bargiela, A., Burke, E.K., Özcan, E., McCollum, B.: Linear Combination of Heuristic Orderings in Constructing Examination Timetables. European Journal of Operational Research, **232**(2), 287–297 (2014)
2. Atsuta, M., Nonobe, K., Ibaraki, T.: Itc2007 track 1: An approach using general csp solver. http://www.cs.qub.ac.uk/itc2007
3. De Smet, G.: Itc2007 - examination track. In: Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, 19-22, August (2008)
4. Demeester, P., Bilgin, B., De Causmaecker, P., Vanden Berghe, G.: A hyperheuristic approach to examination timetabling problems:benchmarks and a new problem from practice. Journal of Scheduling, **15**(1), 83–103 (2012)
5. Gogos, C., Alefragis, P., Housos, E.: A multi-staged algorithmic process for the solution of the examination timetabling problem. In: Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, pp. 19–22, August (2008)
6. Gogos, C., Alefragis, P., Housos, E.: An improved multi-staged algorithmic process for the solution of the examination timetabling problem. Annals of Operation Research **3**, 1–3, (2010)
7. McCollum, B., McMullan, P., Parkes, A.J., Burke, E.K., Abdullah, S.: An extended great deluge approach to the examination timetabling problem. In: The 4th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA09), Dublin, (2009)
8. McCollum, B., McMullan, P., Parkes, A.J., Burke, E.K., Qu, R.: A new model for automated examination timetabling. Annals of Operations Research **194**(1), 291–315 (2012)
9. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, *Caltech Concurrent Computation Program Report* 826, California Institute of Technology (1989)
10. Müller, T.: Itc2007 solver description: A hybrid approach. Annals of Operations Research **172**(1), 429–446 (2009)

Table 3 Performance comparison of our approach to the previously proposed approaches based on the best results (scores) obtained in 10 runs over the ITC2007 benchmark instances.

	Exam 1		Exam 2		Exam 3	
Ranking	Approach	Score	Approach	Score	Approach	Score
1st	Müller[10]	4370	Gogos[6]	385	Gogos[6]	8996
2nd	McCollum[7]	4633	Müller[10]	400	McCollum[7]	9064
3rd	Gogos[6]	4775	McCollum[7]	405	Müller[10]	10049
4th	**SGMA**	**5626**	Demeester[4]	515	Gogos[5]	13771
5th	Gogos[5]	5905	**SGMA**	**616**	Pillay[12]	15917
6th	Demeester[4]	6060	De Smet[3]	623	Atsuta[2]	17669
7th	De Smet[3]	6670	Gogos[5]	1008	Rahman[1]	19098
8th	Atsuta[2]	8006	Pillay[12]	2886	**SGMA**	**19617**
9th	Rahman[1]	11060	Rahman[1]	3133	Demeester[4]	23580
10th	Pillay[12]	12035	Atsuta[2]	3470	De Smet[3]	x
	Exam 4		Exam 5		Exam 6	
Ranking	Approach	Score	Approach	Score	Approach	Score
1st	McCollum[7]	15663	Gogos[6]	2929	Gogos[6]	25740
2nd	Gogos[6]	16204	Müller[10]	2988	McCollum[7]	25880
3rd	Müller[10]	18141	McCollum[7]	3042	Müller[10]	26585
4th	Gogos[5]	18674	De Smet[3]	3847	Demeester[4]	27605
5th	Rahman[1]	20830	Gogos[5]	4139	Gogos[5]	27640
6th	Atsuta[2]	22559	Atsuta[2]	4638	De Smet[3]	27815
7th	Pillay[12]	23582	Demeester[4]	4855	Rahman[1]	28330
8th	**SGMA**	**30010**	**SGMA**	**5002**	Atsuta[2]	29155
9th	Demeester[4]	x	Pillay[12]	6860	Pillay[12]	32250
10th	De Smet[3]	x	Rahman[1]	7975	**SGMA**	**33085**

11. Özcan, E., Asta, S. and Altintas, C.: Memetic algorithms for cross-domain heuristic search. In: The 13th Annual Workshop on Computational Intelligence (UKCI), pp. 175–182 (2013).
12. Pillay, N.: A developmental approach to the examination timetabling problem. In: Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, pp. 19–22, August (2008)
13. Qu, R., Burke, E.K., McCollum, B., Merlot, L., Lee, S.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling **12**(1), 55–89 (2009)

Exam timetabling at Université de Technologie de Compiègne: a memetic approach

Taha Arbaoui · Jean-Paul Boufflet · Kewei Hu · Aziz Moukrim

1 Introduction and problem description

The exam timetabling problem at Université de Technologie de Compiègne (UTC) has some of the usual hard and soft constraints introduced in the second International Timetabling Competition ITC2007. For the sake of simplicity, we use the same terminology for these constraints in the sequel. The other constraints, however, fall into the scope of some of the potential extension of the ITC2007 problem (McCollum et al. 2012).

A timetable is considered as feasible by the practitioner if all exams are assigned to a room and a period while respecting the hard constraints. The quality of the solution is measured using soft constraints. Despite the allowance of scheduling exams in overlapping periods, examination rooms which are spread on different sites cannot be used twice at two overlapping periods. Moreover, rooms have a list of allowed periods.

Contrary to the ITC2007 problem, splitting exams between rooms is permitted. Thus, there are two types of exams: *splittable exams* and *non-splittable exams*. Each exam disposes of a list of allowed rooms and periods. As a result, for an exam to be assigned to room r and period p, they should both be allowed for the exam and room r must be available in the same period as well.

The hard constraints are the following:

- A student cannot sit two exams at the same period or at two overlapping periods.
- An exam must be assigned to a unique period.
- A non-splittable exam must be assigned to a unique room.
- The duration of the exam must be less than or equal to the duration of the period in which it is assigned.

Taha Arbaoui (✉) · Jean-Paul Boufflet · Kewei Hu · Aziz Moukrim
Université de Technologie de Compiègne
Laboratoire Heudiasyc, UMR 7253 CNRS 60205 Compiègne, France
E-mail: taha.arbaoui@hds.utc.fr
jean-paul.boufflet@hds.utc.fr
kewei.hu@etu.utc.fr
aziz.moukrim@hds.utc.fr

Instance	n^E	n^S	n^P	n^R	Conflict density
S2011	141	2322	32	8	0.30
F2011	119	2388	24	9	0.32
S2012	142	2412	36	9	0.31
F2012	117	2296	26	8	0.30
S2011noTC	122	1988	32	8	0.34
F2011noTC	102	2089	24	8	0.33
S2012noTC	123	2057	36	8	0.35
F2012noTC	100	2098	26	8	0.32

Table 1 Characteristics of UTC instances

- The capacity of any room should not be exceeded at any period.
- The sum of the parts of a splittable exam should be equal to the total number of enrolled students.
- A room can be used only at one period of a set of overlapping periods.
- Each exam should be assigned only once.

A solution is considered to be feasible when all the hard constraints are satisfied. On the other hand, when a soft constraint is not satisfied, a penalty is applied. The soft constraints used to measure the quality of the solution differ from one institution to another. A quick look at the ITC2007 benchmark shows that the Front Load penalty is not as important as the penalty for the *Two In a Row*, *Two In a Day* and *Period Spread*. However, this is not the case in our university. Due to the limited time given to professors after the exams to mark them, the practitioner informed us the most important penalty to minimize is the Front Load penalty (minimize the number of big exams planned at the end of the examination session). The following definitions describe briefly the soft constraints used by the practitioner:

Two In a Row: Examinations of a student allocated back to back in the same day should be avoided.
Two In a Day: Examinations of student scheduled in the same day but not back to back should be avoided.
Front Load: Large-size exams should be assigned before a certain period.

Note that ITC2007 differs in both the hard and the soft constraints. Our problem does not consider all the soft constraints used in the ITC2007 problem (*e.g.* Room and Period penalty). On the other part, ITC2007 lacks some of the hard constraints considered in our problem. For example, the overlapping periods and splitting the exams do not exist in ITC2007.

Table 1 presents the characteristics of four instances relative to four semesters in UTC. Column *instance* reports the labels where “S” stands for the spring semester and “F” stands for the fall semester. In UTC, the curriculum is split into two parts. The first part, called “Tronc Commun” (TC), is done in the first two years. The second part is a three-year specialization, at the end of which the student holds an engineering degree. Exams associated to the courses in TC usually constitutes a set of large exams. To measure the impact of these large exams, we studied another type of instances that does not contain the “TC” exams.

Columns n^E, n^S, n^P, n^R and *Conflict density* show for each instance the number of exams, the number of students, the number of periods, the number of rooms and the density of conflict graph, respectively.

2 The memetic algorithm

Evolutionary algorithms, such as Genetic Algorithms (Corne et al. 1994) and Memetic algorithms (Burke et al. 1996), have proved to be effective approaches to tackle exam timetabling problems (Qu et al. 2009). A memetic algorithm (MA) can be viewed as a genetic algorithm in which the key feature is the coupling of the global optimization and local optimization. The global optimization is managed by the crossover procedure while the local optimization is performed using the local search as mutation operators. We proposed a memetic algorithm to solve our problem.

Exams are labelled by integers and a *chromosome* is a permutation of these integers. A decoding procedure is needed to transform this *indirect encoding* into a solution. We used the *First Fit Decoding* (FFD) procedure, inspired from the Bin Packing first fit heuristic (Johnson 1974): exams in the permutation are taken in turn and assigned to the first period and room that respect the hard constraints. The decoding procedure does not always lead to a feasible solution. When it is the case, a *Repairing Method* (RM) is performed until a feasible solution is reached. The chromosome is then updated to the new permutation. The cost of a chromosome is assessed while decoding.

The population is initialized by generating fifty chromosomes at random. Five chromosomes are improved using a rapid destruction-construction local search. The idea is to remove randomly a number of exams, and to reinsert them using the following *Best Insertion* (BI) strategy: allocate an exam to the period and the room that minimize the penalty of the soft constraints.

We applied the Linear Order Crossover (LOX) (Dahal et al. 2007) on parents selected using the following Binary Tournament strategy: two couples of chromosomes are selected at random. The best of the first couple is the first parent and the best of the second couple is the second parent. LOX then randomly selects one of the parents and defines two indices on its permutation. Exams between the two indices are then given to the child and the rest of child is completed from the remaining parent.

The child is mutated with a certain probability. We used three different local searches as mutation operators. The operators are chosen at random. If an operator fails to improve the solution, one of the remaining operators is then applied. The mutation is stopped when none of the operators improves the chromosome.

The mutation operators used are Hill-Climbing (HC), Exam Swap (SWAP) and Light Destruction/Construction (LDC). A conflict graph in which nodes are exams and edges represent incompatibilities between exams is used for Light Destruction/Construction. We briefly describe the mutation operators as follows:

HC: a random order of exams is considered. Next, exams are removed one after another in the defined order, and then inserted back in the solution using BI.

SWAP: two periods are randomly selected. The exams assigned to these periods are swapped. The new solution is accepted *iff* all the hard constraints are respected and the quality of the solution is improved.

LDC: a random exam is removed from the solution. Some of its adjacents in the conflict graph are also removed. These exams are then shuffled and reinserted using BI. If some exams are left unscheduled, the last best permutation of the current chromosome is restored and LDC is reapplied.

Instance	MA	Previous approach	noTC
P2011	5462	10926	3896
A2011	3400	7501	2020
P2012	3450	12008	2586
A2012	3241	9266	1671

Table 2 Results obtained by MA. "noTC" presents the results of MA when the exams of "Tronc Commun" are removed.

The population is updated by replacing an existing chromosome having the same quality of the solution or by replacing the worst chromosome of the population if the new chromosome is better.

3 Preliminary results

Table 2 presents the preliminary results obtained using the memetic algorithm. Results show that MA improved the quality of solutions compared to the previous approach used by the practitioner. The cost of the solutions has been more than halved.

In order to help the practitioner determine the exams involved in most of the solution penalty, we removed the exams of the core curriculum (TC) to measure their impact. The results show that "TC" exams highly contribute to the cost of the solutions on the different instances. This allows us to investigate a two-stage approach aiming at giving more priority to "TC" exams in the planning. In fact, they share students with many other exams which leads to a bigger conflict in the examination session. Taking these exams in priority will effectively lead to high-quality solutions.

References

Edmund K. Burke, James P Newall, and Rupert F Weare. A memetic algorithm for university exam timetabling. In *Lecture notes in computer science: Practice and Theory of Automated Timetabling I: selected papers from the 1st international conference*, volume 1153, pages 241–250. Springer, 1996.

Dave Corne, Peter Ross, and Hsiao-Lan Fang. Evolutionary timetabling: Practice, prospects and work in progress. In *UK planning and Scheduling SIG Workshop*, 1994.

Keshav P. Dahal, Kay Chen Tan, and Peter I. Cowling, editors. *Evolutionary Scheduling*, volume 49 of *Studies in Computational Intelligence*. Springer, 2007.

David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256 – 278, 1974.

Barry McCollum, Paul McMullan, Andrew Parkes, Edmund K. Burke, and Rong Qu. A New Model for Automated Examination Timetabling. *Annals of Operations Research*, 194: 291–315, 2012.

Rong Qu, Edmund Burke, Barry McCollum, Liam T. G. Merlot, and S. Y. Lee. A Survey of Search Methodologies and Automated System Development for Examination Timetabling. *Journal of Scheduling*, 12(1):55–89, 2009.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A Tensor-based Approach to Nurse Rostering

Shahriar Asta · Ender Özcan

1 Introduction

Hyper-heuristics are high level improvement search methodologies exploring space of heuristics [4]. According to [5], hyper-heuristics can be categorized in many ways. A hyper-heuristic either selects from a set of available low level heuristics or generates new heuristics from components of existing low level heuristics to solve a problem, leading to a distinction between *selection* and *generation* hyper-heuristics, respectively. Also, depending on the availability of feedback from the search process, hyper-heuristics can be categorized as *learning* and *non-learning* methods. The learning hyper-heuristics can further be categorized into *online* and *offline* methodologies. The online hyper-heuristics learn *while* solving a problem whereas the offline hyper-heuristics process collected data gathered from training instances prior to solving the problem.

Nurse rostering is a highly constrained scheduling problem which was proven to be NP-hard (Karp, 1972) in its simplified form. Solving a nurse rostering problem requires assignment of shifts to a set of nurses so that 1) the minimum staff requirements are fulfilled and 2) the nurses' contracts are respected [3]. The problem can be represented as a constraint optimisation problem using 5-tuples: (i) set of nurses, (ii) set of days (periods) including the relevant bits from the previous and upcoming schedule, (iii) set of shift types, (iv) set of skill types and (v) constraints.

In this study, a novel selection hyper-heuristic approach is employed to tackle the nurse rostering problem. The proposed framework is a single point based search algorithm which fits best in the online learning selection hyper-heuristic category, even if it is slightly different than the other online learning

S. Asta, E. Özcan
ASAP Research Group
School of Computer Science
University of Nottingham
NG8 1BB, Nottingham, UK
E-mail: sba,exo@cs.nott.ac.uk

selection hyper-heuristics. A selection hyper-heuristic has two main components: heuristic selection and move acceptance method. While the task of the heuristic selection is to select low level heuristics based on a strategy, the acceptance method decides whether or not the solution produced by the selected heuristic shall be accepted. Over the years many heuristic selection and move acceptance methods have been proposed. Examples of heuristics selection strategies are Simple Random (SR) and Random Gradient (RG)[7], Choice Function (CF) [7], Reinforcement Learning (RL) [11] and Tabu Search (TS) [6]. Improvement Only (IO), Improvement Equal (IE), Naive Acceptance (NA), Simulated Annealing(SA) and Late Acceptance (LA) are few examples of move acceptance mechanisms.

In our proposed approach, the trace of a selection hyper-heuristic combining random heuristic selection with a Naive Acceptance (NA) method is represented as a 3-rd order tensor. Factorization of this tensor results in basic factors, which are then analysed and used to partition the low level heuristics into two halves. The first half of the low level heuristics are considered to perform well with NA, while the remaining half are associated with the Improvement Equal (IE) acceptance method. The proposed hyper-heuristic approach then periodically switches between the two move acceptance methods mixing only the relevant low level heuristics associated with them.

2 A Novel Hyper-heuristic Approach: Tensors for Analysing the Space of Heuristics

Tensors are multidimensional arrays. The order of a tensor is the number of dimensions it covers. Tensor decomposition (a.k.a tensor factorization) reduces the dimensionality of the original tensor while keeping the multi-dimensional nature of the data. This helps with preserving the correlation between various modes of data. As well as being somewhat immune to noise, the tensor factors are useful for generalization purposes. The literature is rich of a range of tensor factorization methods such as Higher Order SVD (HOSVD) [10] and Parallel Factor (a.k.a PARAFAC or CANDECOMP or CP) [9]. These methods are often generalization of the Singular Value Decomposition (SVD) method to higher dimensions.

In this study, the search history of a simple random hyper-heuristic with naive acceptance is represented as a 3-rd order tensor. The first two dimensions of this tensor correspond to the current and previous heuristic indexes, called by the underlying hyper-heuristic, while the iteration is projected to the 3rd dimension. The goal is to hybridise multiple hyper-heuristics (SR-IE and SR-NA) and use the most suitable one depending on the randomly chosen low level heuristic at each iterative step under a selection hyper-heuristic framework. This process requires partitioning of heuristics and assigning each group to the best hyper-heuristic. Using the CP decomposition method, the tensor is reduced to a basic factor (frame), interpreting which reveals the pairs of heuristics performing well together under naive acceptance. Analysing these basic factors, scores are associated to each low level heuristic where half of the heuristics which rank highest form a group of heuristics performing well under Naive Acceptance (NA) mechanism. The remaining half form a secondary group which are then applied together with the Improvement Equal (IE) acceptance mechanism. This online learning process is employed for a fixed time in each run on a given problem instance. Then the second stage starts operating. The novel hyper-heuristic approach periodically switches between the two hyper-heuristics in fixed periods and the low level heuristic sets associated with them. Thus, a Tensor-based Hyper-heuristic (THY) emerges.

3 Empirical Results

Hyper-heuristics Flexible Framework (HyFlex) is a Java based software library for the implementation and comparison of various hyper/meta-heuristics across different problem domains [12]. One of those problem domains is the nurse rostering (Personnel Scheduling) implementing 12 low level heuristics of various types: 1 mutation, 5 hill climbing, 3 ruin and re-create and 3 crossover heuristics. Some previously proposed hyper-heuristics do not discriminate the nature of low level heuristics while some others do take that into account in their design. THY is of former type. Since the crossover heuristics are binary operators requiring additional maintenance (for the second argument), for simplicity, they are discarded by THY. Using the HyFlex v1.0 Java library [1], THY is implemented for nurse rostering and tested on 8 benchmark instances [2].

Table 1 The comparison of solutions obtained by previously proposed approaches [8] to the ones found by THY across 31 runs for some nurse rostering benchmark problem instances. BKN denotes the best known solution.

		Our Approach			SS [2]	VDS [1]
Problem	BKN	Best	Avr.	Time (s)	Best	Best
BCV-3.46.1	3280	3284	3312.23	354	3351	-
BCV-A.12.1	1294	1384	1643.90	435	1600	-
BCV-A.12.2	1875	1940	2152.03	277	2180	-
Ikegami 3d1	2	13	19.45	258	-	13
Ikegami 3d1.1	3	15	21.32	409	-	14
Ikegami 3d1.2	3	16	21.74	318	-	9
ORTEC01	270	300	338.74	390	-	465
ORTEC02	270	300	325.97	258	-	510

The best and average costs obtained by THY for each instance are presented in Table 1. The performance of THY is compared to the performance of two previously proposed approaches; scatter search (SS) [2] and variable depth search (VDS) [1] based on the best results achieved by each algorithm. THY performs better than SS on the selected BCV instances. It is also better

[1] http://hyflex.org/
[2] http://cs.nott.ac.uk/~tec/NRP/

than VDS on the ORTEC instances. However, VDS performs slightly better on the selected Ikegami instances. In the overall, the initial experiments show that the tensor-based approach to nurse rostering successfully produces high quality solutions in a reasonable amount of time.

References

1. Burke, E.K., Curtois, T.: New approaches to nurse rostering benchmark instances. European Journal of Operational Research **237**(1), 71 – 81 (2014)
2. Burke, E.K., Curtois, T., Qu, R., Berghe, G.V.: A scatter search methodology for the nurse rostering problem. JORS **61**(11), 1667–1679 (2010)
3. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. J. of Scheduling **7**(6), 441–499 (2004)
4. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society **64**(12), 1695–1724 (2013)
5. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A classification of hyper-heuristics approaches. In: M. Gendreau, J.Y. Potvin (eds.) Handbook of Metaheuristics, *International Series in Operations Research & Management Science*, vol. 57, 2nd edn., chap. 15, pp. 449–468. Springer (2010)
6. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics **9**(6), 451–470 (2003)
7. Cowling, P., Kendall, G., Soubeiga, E.: A parameter-free hyperheuristic for scheduling a sales summit. In: Proceedings of the 4th Metaheuristic International Conference, MIC 2001, pp. 127–131 (2001)
8. Curtois, T.: Published results on employee scheduling instances. `http://www.cs.nott.ac.uk/~tec/NRP/`
9. Harshman, R.A.: PARAFAC: Methods of three-way factor analysis and multidimensional scaling according to the principle of proportional profiles. Ph.D. thesis, University of California, Los Angeles, CA (1976)
10. Lathauwer, L.D., Moor, B.D., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**(4), 1253–1278 (2000)
11. Nareyek, A.: Metaheuristics. chap. Choosing Search Heuristics by Non-stationary Reinforcement Learning, pp. 523–544. Kluwer Academic Publishers, Norwell, MA, USA (2004). URL `http://dl.acm.org/citation.cfm?id=982409.982435`
12. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A., Petrovic, S., Burke, E.: Hyflex: A benchmark framework for cross-domain heuristic search. In: J.K. Hao, M. Middendorf (eds.) European Conference on Evolutionary Computation in Combinatorial Optimisation, EvoCOP '12., *LNCS*, vol. 7245, pp. 136–147. Springer, Heidelberg (2012)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

The Effects of the Planning Horizon on Heathrow TSAT Allocation

Jason A. D. Atkin · Geert De Maere · Edmund K. Burke

1 Introduction and Problem Description

London Heathrow is an extremely popular two-runway airport. The runway forms the major throughput limitation for the departure system and thus increased departure delay is always expected at busy times of the day, when the demand exceeds the available capacity. A Collaborative Decision Making system has been developed by NATS and Heathrow Airport Ltd (HAL) to share information between the various partners at the airport (e.g. airlines, ground handlers, airport staff and the NATS controllers). TSAT (Target Start-up Approval Time) allocation algorithms run within the CDM system, predicting take-off times for aircraft and allocating appropriate times (the TSATs) at which each aircraft will be able to push back and start its engines. The predicted take-off times are provided to Eurocontrol, contributing to airspace capacity improvements. TSAT allocation aims to absorb at the stands some of the delay that aircraft would otherwise experience at the runway. Even a two minute reduction in fuel burn for all aircraft would save over £13M per year, along with the consequent reductions in emissions.

Minimum separations must be attained between any aircraft which use a runway. Re-sequencing can avoid larger separation requirements and improve runway utilisation and delays [3,4]. Due to the unusual constraints at Heathrow, the take-off sequencing can be very sensitive to the mix of aircraft

Jason A.D. Atkin
ASAP research group, School of Computer Science, University of Nottingham, NG81BB, Nottingham, UK
E-mail: jason.atkin@nottingham.ac.uk

Geert De Maere
ASAP research group, School of Computer Science, University of Nottingham, NG81BB, Nottingham, UK

Edmund K. Burke
Department of Computing and Mathematics, University of Stirling, Cottrell Building, Stirling, FK94LA, UK

which are available: only one runway can be used for take-offs at any time; downstream capacity constraints increase separations on some routes; and the airport is running close to capacity. It has been common to release aircraft from the stands as soon as possible, providing the maximal sequencing choice at the runway. Stand delays must not decrease the runway throughput.

The TSAT allocation algorithms predict the take-off sequence which a good controller would achieve at the runway and are described in [3]. Delays around the stands can be considerable, since aircraft can block each other, and have to be considered, resulting in a combination of two sequencing operations: once at the runway and one at the stands [3]. Once a take-off time has been predicted for each aircraft, an ideal pushback time is determined by considering the estimated taxi time to reach the runway from the allocated stand, the expected delays around the stands, and an 'ideal' runway hold. This 'ideal' runway hold adds slack to allow for deviations from predicted taxi times and to provide a pool of aircraft for the runway controller to choose from.

The TSAT allocation system relies upon the early provision of planned earliest pushback times (called TOBTs, Target Off-Block Times) from airlines in order to predict the earliest take-off time for each aircraft. As it is made aware of more aircraft over time, better sequences may be found, fitting new aircraft into the sequence and making small changes. It was observed in [2] that the planning horizon has a huge effect upon the runway sequencing efficiency at Heathrow. The aim of this research is to discover the extent to which this also applies when sequencing is performed at the stands in order to allocate stand holds and whether a lack of early information is likely to release inappropriate aircraft to the runway, leading to poor sequencing, and/or increased fuel burn.

2 Experimental details and results

The same model, parameters and objectives have been used in this research as in [4], except for the addition of a cost for larger runway delays to model the fact that a runway controller is likely to allow aircraft which have been waiting at the runway for a long time to take off earlier. This will be discussed in more detail in the full paper. The primary objective of the algorithm is to meet take-off time windows, where possible, and the secondary objective is to achieve low delay sequences which avoid excessive inequity between aircraft. The trade-off between these objectives was considered in [1].

Eighteen Heathrow datasets, provided by NATS, were tested, containing 105 consecutive take-offs each, from the same runway. The first five take-off times were fixed, to provide a take-off history. An iterated simulation was performed whereby the algorithm considered the aircraft it knew about, predicted take-off times and allocated TSATs, then advanced the time by 60 seconds, adding in new aircraft, then repeated this process. The historic pushback times were used for the TOBTs and were provided to the system PH seconds before TOBT, where PH is the planning horizon being investigated. TSATs were

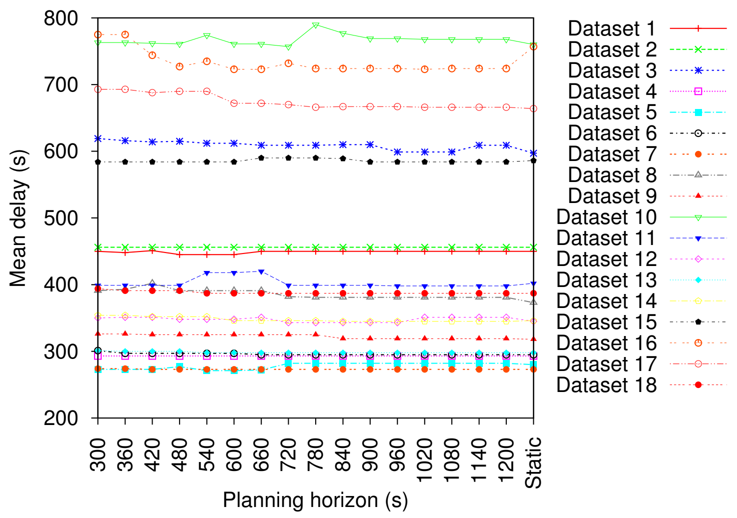


Fig. 1 Graph of mean delay (stand hold + runway delay) vs planning horizon for all 18 datasets

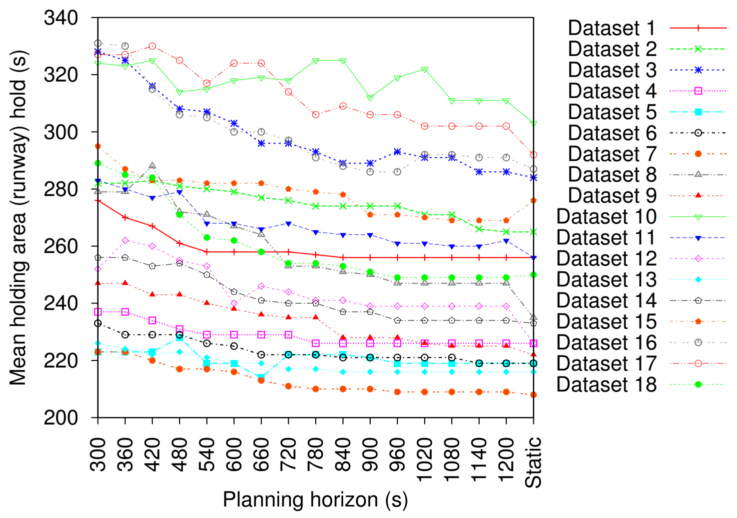


Fig. 2 Graph of mean runway hold vs planning horizon for all 18 datasets

frozen 300 seconds before the TSAT time. No uncertainty was assumed in the predictions.

Figure 1 shows how the mean delay (on the y axis) changes as the planning horizon increases (from left to right) and compares it against the delay in the static problem (on the far right, considering all aircraft simultaneously). The delay is not greatly affected by a low planning horizon, since the system still has the potential to change the take-off sequence once the aircraft have left

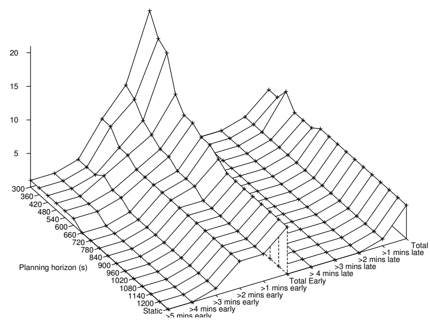


Fig. 3 Deviation of aircraft from ideal TSATs, Dataset 1

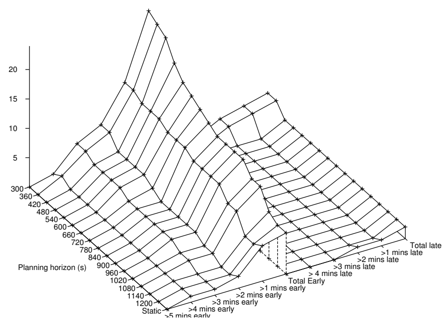


Fig. 4 Deviation of aircraft from ideal TSATs, Dataset 2

the stands. In contrast, the runway delay is shown in Figure 2 and is much larger for lower planning horizons. This indicates that significant re-sequencing is being performed after the aircraft leave the stands and that the lack of information is leading to stand holds which would have been larger otherwise, leading to unnecessary fuel burn.

Figures 3 and 4 show, for two datasets, the deviation of the allocated TSAT from the ideal TSAT which would have been allocated based upon the final take-off sequence which was achieved. The delays around the stands mean that ideal stand holds could not be allocated to all aircraft even in the static case (the line closest to the reader). It can be observed that the system is applying shorter stand holds than ideal to some of the aircraft and that it is very rare for aircraft to be released later than ideal.

This research shows that late information from airlines may adversely affect the potential benefits of the TSAT allocation system, leading to increased airline costs and emissions. The complete consideration of the full results, the effects of the ideal stand hold, the time at which the TSAT is frozen, and the function to limit excessive runway hold are left for the full paper.

References

1. Atkin, J.A.D., Burke, E.K., Greenwood, J.S.: The TSAT Allocation System at London Heathrow: The Relationship Between Slot Compliance, Throughput and Equity. Public Transport **2**(3), 173–198 (2010)
2. Atkin, J.A.D., Burke, E.K., Greenwood, J.S., Reeson, D.: The Effect of the Planning Horizon and Freezing Time on Take-off Sequencing. In: Proceedings of the 2nd International Conference on Research in Air Transportation (ICRAT 2006), Belgrade, Serbia and Montenegro (2006)
3. Atkin, J.A.D., Burke, E.K., Greenwood, J.S., Reeson, D.: Hybrid meta-heuristics to aid runway scheduling at London Heathrow airport. Transportation Science **41**(1), 90–106 (2007)
4. Atkin, J.A.D., De Maere, G., Burke, E.K., Greenwood, J.S.: Addressing the Pushback Time Allocation Problem at Heathrow Airport. Transportation Science **47**(4), 584–602 (2013)

Dantzig-Wolfe decomposition of Meeting planning problems

Niels-Christian Fink Bagger · **Matilda Camitz** ·
Thomas Stidsen

Abstract Planning meetings of groups of persons is an activity which secretaries all over the world performs every day. For the cases where persons should participate in several meetings, this can actually become a hard planning problem to solve. In this abstract we will briefly present a Branch & Price approach for a general setup. A complete article about this approach has been submitted [1].

Keywords Meeting planning, Branch & Price

1 Introduction

Meeting planning *can* be a non-trivial task, if enough people are involved and if several of the people have to participate in several meetings. In this case, the planning problem becomes NP-hard [1].

2 Dantzig-Wolfe decomposition

The overall idea in this abstract is to apply Dantzig-Wolfe decomposition and create a master optimization problem and a sub optimization problem, where the differences between the different versions of meeting planning problems are ”hidden” in the sub-problem.

The two optimization problems, for which we will apply Column Generation, inside a Branch & Bound algorihm (i.e. Branch & Price) are now:

- A master problem which plans the meetings according to person schedules
- A sub-problem, per person, which generates new person schedules for each person.

Unfortunately we are not able to present the sub-problem models in this abstract, since these models are rather large and we have limited space here.

2.1 Master Problem

The master problem assume to have, for each person, a number of meeting plans. Each meeting plan specify *when* a meeting with the person is possible, but not which meeting. Assume that there are a number of persons e who has to participate in some meetings g. The meetings can occur in a number of timeslots b but naturally each

Niels-Christian Fink Bagger, Camilla Camitz · Thomas Stidsen
Technical University of Denmark
E-mail: thst@dtu.dk

person can at most participate in *one* meeting in each timeslot. The binary variable $x_{g,b}$ defines if meeting g takes place in timeslot b and there is a gain $\alpha_{g,b} > 0$ by having meeting g in time slot b. Whether a person should participate in a meeting is defined by an incidence matrix A_g^e which is 1 if person e is part of the group g constituting the meeting. The The master problem will attempt to select the best meeting plans $\lambda^{e,p}$ for each person e and plan p. A meeting plan is defined by an incidence matrix $M^{e,p,b}$ which is 1 if person e in plan p can have a meeting in timeslot b. The cost of a meeting plan is $\beta^{e,p} < 0$. The full master problem is now:

$$\max \quad \sum_{g,b} \alpha_{g,b} \cdot x_{g,b} + \sum_{e,p \in P_e} \beta^{e,p} \cdot \lambda^{e,p} \tag{1}$$

$$\text{s.t.} \quad \sum_{b} x_{g,b} \leq 1 \quad \forall g \tag{2}$$

$$\sum_{g} A_g^e \cdot x_{g,b} - \sum_{p \in P_e} M^{e,p,b} \cdot \lambda^{e,p} \leq 0 \quad \forall e,b \tag{3}$$

$$\sum_{p \in P_e} \lambda^{e,p} \leq 1 \quad \forall e \tag{4}$$

$$x_{g,b}, \lambda^{e,p} \in \{0,1\} \tag{5}$$

2.2 Sub-problem

The sub-problem generates meeting plans for each person. For different types of meeting problems, different requirements can be put here. Due to space limitations, it is impossible to include the three different sub-problems of the test problems in this abstract and we refer to the full article [1].

3 Tests

We test the approach on three different problems:

- Parent-teacher meetings at high-schools
- Supervisor-student meetings at high-schools
- Exam planning at high-schools

The developed Branch & Price algorithm is compared to two alternative approaches: ALNS and MIP.

ALNS (Adaptive Large Neighborhood Search) is a relatively new meta-heuristic, see [2]. An ALNS can be considered to be a special type of hyper heuristic and ALNS algorithms has been very successful in the area of Vehicle Routing. An ALNS has been developed for the Parent-teacher meeting problem and the Supervisor-student meeting problem, see [3].

MIP (Mixed Integer Programming) is a direct model of the two different problems, which is solved directly in the Gurobi solver [4].

3.1 Parent-teacher meetings

In Danish high-schools (9’th to 12’th grade), the school will typically arrange 2 meetings pr. year between the student and parents, and the teachers which they wish to meet. The objective is to minimize the time for the schedules of the students. The overall results when testing on 100 real-world problems are shown below in Table 1. Unfortunately, because of space restrictions we are not able to describe the details for each of the 100 real-world problems. Statistically, the number of meetings varies between 23 and 1187, the number of timeslots varies between 8 and 51 and the number of persons varies between 21 and 307. The largest dataset contains 779 meetings spread over 51 timeslots for 291 persons.

Table 1 Summary of results for parent-teacher meetings. 'Best obj' denotes the amount of instances where the algorithm provided the best objective value (including draws). 'Best UB' denotes the amount of instances where the algorithm found the best upper bound (including draws). Columns 'Gap $\leq q$ shows the amount of instances for which the respective algorithm provided a gap $\leq q$. 'Avg. Gap to best UB' is found for each algorithm by finding the best available UB for each instance, calculating the gap to the solution provided, and averaging these gaps.

	Best obj	Best UB	Gap = 0%	Gap ≤ 2%	Gap ≤ 5%	Avg. Gap to best UB
ALNS	46	-	-	-	-	2.31%
MIP	21	19	17	23	35	9.37%
B&P	54	94	16	54	92	2.32%

3.2 Supervisor-student meetings

In the third year of high-school, the students have to write a bigger assignment, for which they will get two supervisors in different subjects. Again the job is to plan the meetings between the teachers and the students. The overall results after testing on 100 real-life datasets are shown in Table 2 below. Unfortunately, because of space restrictions we are not able to describe the details for each of the 100 real-world problems. Statistically, the number of meetings varies between 21 and 303, the number of timeslots varies between 8 and 102 and the number of persons varies between 29 and 367. The largest dataset contains 258 meetings spread over 74 timeslots for 288 persons.

Table 2 Summary of results for supervisor-student meetings. Columns are equivalent to those in Table 1.

	Best obj	Best UB	Gap = 0%	Gap ≤ 2%	Gap ≤ 5%	Avg. Gap to best UB
ALNS	37	-	-	-	-	1.26%
MIP	16	14	10	22	42	7.13%
B&P	68	95	23	80	97	1.15%

3.3 Exam planning

Exam planning is the problem of scheduling the exams such that no student will have to go to two exams on the same day and that as many students as possible will have good preparation time between the exams. Just solving the relaxed probelm using Column Generation is so slow that full problems cannot be solved. Hence we will not present any results here.

3.4 Conclusion

Many different time-table problems can be considerede to belong to the meeting planning problem category. The particular Dantzig-Wolfe decomposition approach works fine on two of the test problems, but not on the last problem. We conclude that the method works well, if there are not too many persons involved and if each person has a significant number of meetings, i.e. more than say 3.

References

1. Niels-Christian Fink Bagger et. al., ”A Mathematical Programming Approach to the Generalized Meeting Planning Problem”, submitted to Computers & OR
2. Stefan Ropke and Pisinger, David, ”An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows”, TRANSPORTATION SCIENCE, Vol. 40, No. 4, pp. 455-472, 2006
3. Simon Kristiansen, Matias Sorensen, Michael Herold and Thomas Stidsen, ”The Consultation Timetabling Problem at Danish High Schools”, Journal of Heuristics, Vol. 19, No. 3, pp. 465-495, 2013
4. Gurobi Optimization, Inc., ”Gurobi Optimizer Reference Manual”, 2012.

Room Allocation Optimisation at the Technical University of Denmark

Niels-Christian Fink Bagger · Jesper Larsen · Thomas Stidsen

Keywords University Course Timetabling · Room Allocation · Mathematical Programming · Heuristics · Matheuristics

1 Introduction

As at many other universities the Technical University of Denmark (DTU) faces the challenge of solving a case of the curriculum based university course timetabling problem (CUCTT) multiple times a year. However, there are some slight modifications to the CUCTT problem usually described in the literature. One of the major difference is that the assignment of the courses to specific time slots are predetermined and cannot be subject to changes. This is a decision made by the administration since this takes away the issue of course collisions, e.g. when two courses sharing a student are allocated at overlapping time slots, since the students are to ensure by themselves that their courses do not overlap. The problem was first considered in the masters' thesis [1] and the project here is an extension of the work done in that thesis.

2 Objectives

For each course a predetermined set of events will occur during the semester, e.g. lectures, laboratory exercises, tutorials and so on. Each event may be allocated to more than one room. It is allowed to allocate the events such that the sum of the capacities of the allocated rooms does not accommodate all the students attending the event and an objective is to minimize this number of unallocated students. Some of the lecturers prefers to give their lectures

Niels-Christian Fink Bagger, Jesper Larsen, Thomas Stidsen
Technical University of Denmark
Tel.: +45 45 25 25 42
E-mail: nbag@dtu.dk

in specific rooms. An objective is to minimise the geographical distance between the allocated rooms and the requested rooms of the events. A highly prioritised soft constraint is that events from the same course occurring the same day must be allocated as close to each other as possible with respect to their geographical location and if an event is split into multiple rooms, these must also be as close to each other as possible in the geographical sense. The difference between this soft constraint and the *Room Stability* considered in the curriculum timetabling problem of ITC2007 [2] is that in ITC2007 the number of different rooms that a course or event is assigned to is penalised whereas here the actual physical distance between the rooms are penalised. As an example of the difference between the two penalties consider Fig. 1.

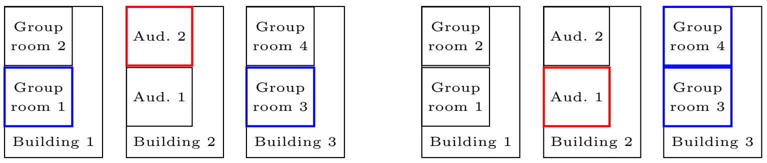


Fig. 1 Example of a course consisting of two events; a lecture followed by group exercises. The auditorium coloured in red is the selected room for the lecture and the two group rooms marked in blue are the selected rooms for the group exercises. The two have the same *Room Stability* penalty, however the solution to the right is considered to be better than the solution to the left since the allocated rooms are closer to each other.

3 Solution Approach

The Mixed Integer Program (MIP) formulation describing the problem is given in Fig. 2.

The following sets and parameters are given as input to the model:

- E, R, C are sets of events, rooms and courses respectively
- $\mathcal{C}$ is every distinct pair of events which are from the same course and occurs on the same day of the week
- $\mathcal{R}$ is every distinct pair of rooms
- χ is every distinct pair of events which are overlapping in time but not from the same course.
- P_r is the capacity of room $r \in R$
- $F_{e,r}$ is 1 if event $e \in E$ is allowed to be scheduled in room $r \in R$ and 0 otherwise
- $R_e^{\max}$ is the maximum number of rooms that event $e \in E$ is allowed to be split into.
- The parameters α_e, $\gamma_{e,r}$, $\zeta_{r,r'}^{e}$ and $\zeta_{r,r'}^{e,e'}$ for penalising respectively, the number of unseated students of event $e \in E$, the allocation of event $e \in E$ to room $r \in R$, the allocation of event $e \in E$ to both room $r \in R$ and $r' \in R$ and the allocation of events $e \in E$ and $e' \in E$ to rooms $r \in R$ and $r' \in R$

The binary variable $x_{e,r}$ takes value 1 if event $e \in E$ is scheduled in course $r \in R$ and 0 otherwise. The variable y_e will take the value of the number of students from event $e \in E$ that are not seated in the solution. The variable $t^e_{r,r'}$ will take value 1 if event $e \in E$ is scheduled in both room $r \in R$ and in room $r' \in R$. The variable $u^{e,e'}_{r,r'}$ will take value 1 if event $e \in E$ is scheduled in room $r \in R$ and event $e' \in E$ is scheduled in room $r' \in R$.

$$
\begin{array}{llll}
\min & \sum_{e \in E} \alpha_e \cdot y_e + \sum_{e \in E, r \in R} \gamma_{e,r} \cdot x_{e,r} + \sum_{\substack{e \in E,\\ (r,r') \in \mathcal{R}}} \zeta^e_{r,r'} \cdot t^e_{r,r'} + \sum_{\substack{(e,e)' \in \mathcal{C}\\ (r,r') \in \mathcal{R}}} \zeta^{e,e'}_{r,r'} \cdot u^{e,e'}_{r,r'} & & (1)\\
\text{s.t.} & x_{e,r} \leq F_{e,r} & \forall e \in E, r \in R & (2)\\
& \sum_{r \in R} x_{e,r} \leq R^{\max}_e & \forall e \in E & (3)\\
& \sum_{r \in R} P_r \cdot x_{e,r} + y_e \geq S_e & \forall e \in E & (4)\\
& x_{e,r} + x_{e',r} \leq 1 & \forall (e,e') \in \chi, r \in R & (5)\\
& x_{e,r} + x_{e,r'} - t^e_{r,r'} \geq 1 & \forall e \in E, (r,r') \in \mathcal{R} & (6)\\
& x_{e,r} + x_{e',r'} - u^{e,e'}_{r,r'} \leq 1 & \forall (e,e') \in \mathcal{C}, (r,r') \in \mathcal{R} & (7)\\
& x_{e,r} \in \mathbb{B} & \forall e \in E, r \in R & \\
& y_e \geq 0 & \forall e \in E & \\
& t^e_{r,r'} \geq 0 & \forall e \in E, (r,r') \in \mathcal{R} & \\
& u^{e,e'}_{r,r'} \geq 0 & \forall (e,e') \in \mathcal{C}, (r,r') \in \mathcal{R} &
\end{array}
$$

Fig. 2 Model of the room allocation problem.

The description of the objective and the constraints follows:

(1) The weighted sum of all the soft constraints
(2) Event $e \in E$ can only be scheduled into room $r \in R$ if it is marked as feasible
(3) Event $e \in E$ can at most be put into $R^{\max}_e$ rooms
(4) If event $e \in E$ is put into rooms where the total capacity is less than the number of students S_e then y_e is given a lower bound of the difference, i.e. the number of students which are unallocated
(5) Two events, $e \in E$ and $e' \in E$, which are overlapping cannot be scheduled in the same time slot
(6) If an event $e \in E$ is allocated to both room $r \in R$ and $r' \in R$ then $t^e_{r,r'}$ is given a lower bound of 1
(7) If event $e \in E$ is allocated to room $r \in R$ and event $e' \in E$ is allocated to room $r' \in R$ then $u^{e,e'}_{r,r'}$ is given a lower bound of 1

3.1 Greedy Graph Colouring Constructive (Mat)Heuristic

Since the problem has similarities to graph colouring and the k–clique problem it seems natural to implement algorithms inspired from solution approaches to these problems. This is for instance done for the examination timetabling problem in [4] showing good results. The idea of ordering events due to "difficulty" is adopted.

The heuristic works by iteratively making a lexicographic order of the unscheduled events by "difficulties" and preferences and then picking the best allocation for the first event in the list. The "difficulty" of an event is based on the hard constraints of the mathematical model that the event is part of, i.e. how difficult it is expected to be to schedule. Since different planners have different priorities of the preferences the heuristic will adapt the choice of the ordering during the search. This is done by assigning a score $s_{i,j}$ for each pair of soft or hard constraints i and j taking an initial value of 1 indicating how well the algorithm performs when the events are ordered lexicographically by i before j and then updated in each iteration as a result of the performance. The ordering of the constraints is done by the algorithm described in Algorithm 1.

Algorithm 1: OrderList

```
   Input: An unordered list of the indces of the constraints $L$
   Output: An ordered list $O$
1  Pick the first element in $L$, remove it from $L$ and insert it in $O$
2  while $L \neq \emptyset$ do
3      Pick the first element $i$ in $L$ and remove it from $L$
4      foreach $j \in O$ in the given order do
5          Pick a random number $r \in [0, 1]$
6          if $r \leq \frac{s_{i,j}}{s_{i,j}+s_{j,i}}$ then
7              Insert $i$ in $O$ right before $j$
8              Exit the foreach-loop
9      If $i$ did not get inserted then insert $i$ last in $O$
```

After the constraints have been ordered the events are put into a sorted list in the given lexicographical order. Then the first event from the list is taken out and allocated to a room which decreases the objective value. If such a room cannot be found then the next event in the list is chosen. The algorithm stops the first time an event is allocated a room and then the constraint list is reordered for the next iteration of the heuristic.

This heuristic has also been extended into a matheuristic. The basic idea is the same but instead of considering only one event of the time the matheuristic chooses the first k events in the ordered list. Then the given MIP model is solved to find the best allocation for the k events where previously allocated events are fixed at their location and unallocated events, which are not one of the k chosen events, are ignored.

4 Results

For comparison the heuristic and the matheuristic has been implemented in C# an tested on five real-life data sets from DTU. The model has also been tested by a direct implementation in Gurobi[3] v. 5.5.1. The time limit has been set to fifteen minutes and an Overview of the results can be seen in Table 1.

	H		MH		GRB		
Name	Time	UB	Time	UB	Time	UB	LB
DataSet1	6	4016031	91	1408853	900	1505480	1155852
DataSet2	7	7057059	109	2857753	900	3055835	1668848
DataSet3	14	3290617	153	2039970	900	1938341	1500971
DataSet4	5	8083225	90	7254350	900	6629019	5934471
DataSet5	3	4193801	64	1582210	900	1525580	1451168
Average	7	5328147	101	3028627	900	2930851	2342262

Table 1 Comparison of the heuristic (H), matheuristic (MH) and Gurobi (GRB). UB is the obtained value, Time is the time that the algorithm spent in seconds, LB is the lower bound.

From Table 1 it can be seen that the heuristic is outperformed by both the matheuristic and Gurobi in terms of the obtained solution. The matheuristic and Gurobi are very close, however the matheuristic obtains the solutions within a much smaller amount of time and since it is a constructive heuristic it can potentially get a better solution if some improvement step is implemented. The use of a MIP solver inside the constructive heuristic is an easy way to extend the heuristic and in this case improves the performance significantly. However the project is still in a very preliminary state and other heuristics known for performing well from the literature needs to be implemented for comparison.

References

1. Bærentsen, R.: Optimization of room-allocation at the technical university of denmark. Masters' thesis, Technical University of Denmark
2. Gasparo, L.d., McCollum, B., Schaerf, A.: The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Tech. rep. (2007). Http://www.cs.qub.ac.uk/itc2007/curriculmcourse/report/curriculumtechreport.pdf
3. Gurobi Optimization, Inc.: Gurobi Optimizer Reference Manual (2014). URL http://www.gurobi.com
4. Sabar, N.R., Ayob, M., Qu, R., Kendall, G.: A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence **37**(1), 1–11 (2012)

Modeling and Solving a Real-Life Multi-Skill Shift Design Problem

Alex Bonutti · Fabio De Cesco · Nysret Musliu · Andrea Schaerf

1 Introduction

Shift design is an important phase within the workforce management process (see, e.g., [2]). According to labor regulations, in all industrial sectors, shifts must include breaks for employees for both resting and eating. Therefore, breaks must be taken into account when designing shifts, in order to be able to meet precisely the staffing requirements and thus ensure the desired service level. For this reason, we address the *shift and break* design problem, rather than shift design only.

In this work, we propose a novel multi-skill formulation of the problem arising from a few practical cases. In addition, we propose a new search method based on Simulated Annealing (SA), that, differently from previous approaches (see [3]), solves the overall problem as a single-stage procedure. The core of the method is a composite neighborhood that includes at the same time changes in the staffing of shifts, the shape of the shifts, and the position of the breaks.

The experimentation, which is still on going, makes use of statistically-principled techniques for the tuning of the numerous control knobs. Indeed, they include both the standard parameters of SA and the distributions for the selection of the candidate moves out of the composite neighborhood.

2 Problem formulation

The problem formulation includes the following main entities:

A. Bonutti and F. De Cesco
EasyStaff s.r.l., Via Adriatica, 278 - 33030 Campoformido (UD), Italy.
E-mail: {alex,fabio}@easystaff.it
N. Musliu
DBAI, Technische Universität Wien, Austria. E-mail: musliu@dbai.tuwien.ac.at
A. Schaerf
DIEGM, University of Udine, Via delle Scienze 206, Udine, Italy. E-mail: schaerf@uniud.it

Planning Granularity: The length of the indivisible step for the planning (typically 10' or 15'), used to divide the planning period into discrete *timeslots*.

Planning Horizon: Number of days of planning (typically 7 or 14). The schedule is assumed cyclic, so that the shifts that cross the midnight of the last day contribute to fulfill the requirements of the morning of the first day.

Requirements: The requirements specify for each skill, for each timeslot of each day of the planning horizon, the number of required workers. This number is not strict, so that overstaffing and understaffing is allowed, but penalized in the objective function.

Shift types: Each shift belongs to a shift type, which sets several constraints on the shape of the shift:
- Minimum and maximum length
- Minimum and maximum start time
- Granularity of the shift length (a multiple of the Planning Granularity)
- Break presence (Boolean-valued)

For shift types with break, the following additional data is included:
- Break length
- Minimum distance of the break from the beginning of the shift
- Minimum distance of the break from the end of the shift
- Minimum start time of the break
- Maximum end time of the break

In our setting, we consider only the break for the meals (that normally last 1 hour), therefore there is at most one break for each shift. Shorter breaks for resting are not planned, but rather assumed to be taken deliberately by the operator according to the current situation.

The problem consists in selecting a set of shifts (belonging to the given types), and the corresponding staffing level for each skill, so as to minimize the cost of the following objectives:

1. understaffing and overstaffing for each timeslot;
2. number of different shifts;
3. average length of shifts.

For the third objective, for each skill we average the length of the shifts (multiplied by the number of operators) and we penalize the discrepancy with respect to a given range (see [4] for the details).

Our current problem is an extension of the shift design problem solved in [4] and [2]. In the current formulation employees can have different qualifications and the workforce requirements for each skill should be fulfilled. Additionally, the scheduling of lunch break has not been considered in [2,4].

The skills of employees have been considered in shift scheduling by several researchers. Bhulai *et al.* [1] for example investigate scheduling of shifts in multi-skill call centers by an approach that is based on a heuristic method and an integer programming model. Quimper and Rousseau [5] investigate modeling of the regulations for the multiple activity shift scheduling problem by using regular and context-free languages and solved the overall problem with Large Neighborhood Search. Generation of large number of breaks per shifts has been also recently considered in the literature. We refer the reader to [3] for a review of previous work on this problem.

3 Search method

We propose a simulated annealing approach that uses a neighborhood relation that extends and adapts to the case with breaks the one proposed in [2] for simple shift design. In details, we consider the union of the following basic neighborhoods:

1. Shrink or enlarge a shift by one timeslot.
2. Add, remove, or transfer to another shift a worker from a shift.
3. Move the break forward or backward by one timeslot.

For neighborhoods 1 and 3, the move can be selected only if it lead to a shift that is still compliant with its given type.

4 Current and future work

The current work involves the collection of a set of real case instances, and an experimentation on these instances, with the following objectives:

– Define the suitable weights of the objectives in order to obtain the results that better fit with costumers' needs.
– Set the parameters to their best values (with the given statistical confidence) of the search method to obtain good results of the available instances and robust performances for future unforeseen instances.

Acknowledgements Nysret Musliu has been supported for this work by the Austrian Science Fund (FWF): P24814-N23.

References

1. Sandjai Bhulai, Ger Koole, and Auke Pot. Simple methods for shift scheduling in multiskill call centers. *Manufacturing & Service Operations Management*, 10(3):411–420, 2008.
2. 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.
3. Luca Di Gaspero, Johannes Gärtner, Nysret Musliu, Andrea Schaerf, Werner Schafhauser, and Wolfgang Slany. Automated shift design and break scheduling. In *Automated Scheduling and Planning*, pages 109–127. Springer, 2013.
4. Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. Local search for shift design. *European Journal of Operational Research*, 153(1):51–64, 2004.
5. Claude-Guy Quimper and Louis-Martin Rousseau. A large neighbourhood search approach to the multi-activity shift scheduling problem. *Journal of Heuristics*, 16(3):373–391, 2010.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Airport Ground Movement: Real World Data Sets and Approaches to Handling Uncertainty

Alexander E.I. Brownlee · Jason A.D. Atkin · John R. Woodward · Una Benlic · Edmund K. Burke

Keywords airport operations · ground movement · data sets · uncertainty

1 Abstract

Two related topics are considered in this presentation, both concerning the ground movement of aircraft. The first describes the collection of data from publicly available websites and the second discusses the issue of uncertainty in this problem.

The airport ground movement problem [1] connects together the problems of runway scheduling and gate allocation, which are often tackled separately in the research literature. The overall problem involves allocating routes for aircraft to take as they proceed along the taxiways between the runways and the gates (stands), and timings or orders for them to take them. The aim is to find a schedule that reduces delays, reduces the fuel burn associated with taxiing, and is resilient to last-minute changes. This represents a challenging problem because there are typically several pinch points where congestion is more likely to occur, and the uncertainty inherent in aircraft landing times, pushback times and taxi speeds means that routes need to be constantly updated to reflect the current situation. In addition, any solution method must be efficient enough that it can be executed within a couple of minutes, at most, to accommodate incoming, changing data. Furthermore, once an aircraft has had a route allocated and commenced its movement, it is undesirable (and

A. Brownlee, J. Woodward, U. Benlic, E. Burke
Division of Computer Science and Mathematics
University of Stirling, UK
E-mail: sbr/jrw/ube@cs.stir.ac.uk, e.k.burke@stir.ac.uk

J. Atkin
School of Computer Science
University of Nottingham, UK
E-mail: Jason.Atkin@nottingham.ac.uk

avoided at most airports) for a new route to be assigned. This means that any route allocation should be robust enough to cope with uncertainties that arise during the ground movements such as variations in taxi speed.

Innovation in this area is potentially limited by the difficulty in accessing real-world data sets. While some freely-available toy problems exist in the literature, none truly reflect the inherent complexity of operations at a real airport. Existing works have made use of data provided through partnerships with airports, but typically this is the subject of a non-disclosure agreement. This means that there are currently no up-to-date common datasets for researchers to compare approaches on. This acts as a barrier to new researchers entering the field, who would have to develop working relationships with airport staff to obtain relevant data, and we intend to resolve this problem.

This work explores several freely available sources of data related to airport ground movement, considering the ways in which these can be combined to confront this challenge. Initial attempts have also been made to quantify and address the important issue of uncertainty in taxi time estimation. The freely available data sets include layout information derived from open street map (OSM) (www.openstreetmap.org), which is free to use under an open licence, and the NATS Aeronautical Information Service (www.ais.org.uk). A tool has been developed and made available to generate layouts taken from raw data downloaded from these sites. Some example layouts for UK airports have also been made available. Furthermore, the work has explored the use of live flight track information taken from the site Flight Radar 24 (FR24) (http://www.flightradar24.com). These tracks are available for the majority of flights at most European and US airports and are detailed enough to allow analysis of the real movements of aircraft at an airport. However, due to measurement errors the recorded data appear to be precise but is not actually accurate (for example on occasions an aircraft is recorded as travelling along a path parallel to the actual runway rather than along it). In order to retain as much valuable data as possible, these paths can often be "repaired" by applying a linear transformation to the recorded trajectory and forcing the path of the data to coincide with the closest runway. To recover the most likely path, the raw data is snapped to the taxiways, from which taxi speeds and routes taken can be analysed. A tool has also been made available to conduct this snapping process from raw GPS track coordinates obtained from a site such as FR24.

Existing work [3] has quantified the uncertainty in terms of taxi time estimation using existing taxi time modelling. This work has expanded upon that by using the real flight data taken from FR24. The QPPTW algorithm (quickest path problem with time windows) is an existing state of the art approach for optimising the ground movement problem [2]. This finds the shortest path for a given aircraft given the routes which have already been allocated to other aircraft. However, the algorithm assumes that taxi time estimates are known exactly, which may not be the case. One approach for extending this work is to add a buffer time (slack) to aircraft movements to allow for variations in taxi speed. However, this impacts on the overall capacity of the airport and can add unnecessary delay to aircraft movements. An existing approach to handling the similar problem of flow shop scheduling with uncertain (fuzzy) processing times can be adapted to the ground movement problem. Some preliminary results are presented exploring and contrasting the impact on final taxi-times and ground movement efficiency of adding padding and applying the fuzzy approach.

Acknowledgements This work was funded by the UK EPSRC [grant number EP/H004424/2].

References

1. Atkin, J.A.D., Burke, E.K., Ravizza, S.: The Airport Ground Movement Problem: Past and Current Research and Future Directions. In: 4th International Conference on Research in Air Transportation, pp. 131–138 (2010)
2. Ravizza, S., Atkin, J.A.D., Burke, E.K.: A more realistic approach for airport ground movement optimisation with stand holding. Journal of Scheduling (2013). DOI 10.1007/s10951-013-0323-3.
3. Ravizza, S., Chen, J., Atkin, J.A., Stewart, P., Burke, E.K.: Aircraft taxi time prediction: Comparisons and insights. Applied Soft Computing **14**(Part C), 397 – 406 (2014). DOI 10.1016/j.asoc.2013.10.004.

Scheduling Air Traffic Controllers

**R.Conniss · T.Curtois · S.Petrovic ·
E.Burke**

Keywords Personnel Scheduling, Rostering, Heuristics

1 Introduction

The effective rostering of Air Traffic Controllers is a complex and under researched area of the personnel scheduling literature. An effective method to produce real world rosters for controllers requires the ability to model shifts, breaks, multiple tasks and their associated qualifications, to rotate staff through all the tasks for which they are qualified to maintain skill levels, the requirement to train staff whilst continuing normal operations and an ability to reroster in the event of unexpected events. Examples in the literature that examine some of these components include shift scheduling [5], break planning [2] [6] and multi skilled staff [4] [3].

We shall present an algorithm that can effectively model many of the features of the ATC rostering problem, and produce useful real world rosters for operational use.

R.Conniss
OMIS, NUBS, University of Nottingham, UK
E-mail: psxrc@nottingham.ac.uk

T.Curtois
ASAP Research Group, School of Computer Science, University of Nottingham, UK
E-mail: tim.curtois@nottingham.ac.uk

S.Petrovic
OMIS, NUBS, University of Nottingham, UK
E-mail: Sanja.Petrovic@nottingham.ac.uk

E.Burke
University of Stirling E-mail: e.k.burke@stir.ac.uk

2 Problem Description

Creating a feasible roster for controllers depends on similar hard constraints to other classes of rostering problem. Controllers have limits on their maximum shift length, how long they can work without a break during a given shift and minimum time off between shifts. Although the initial focus of the research was on RAF controllers, the working limits of civilian controllers have been used in the model for two reasons. Firstly, military controllers are not subject to well defined working time rules, due to the nature of military service. Secondly, the Civil Aviation Authority publishes UK wide rules for all civilian controllers in the Scheme for the Regulation of Air Traffic Controller's Hours (SRATCOH) [1].

For a controller to be assigned to a console position, or task, the controller must hold the correct qualification to staff the task. Each controller will hold some subset of all available qualifications and some will be fully qualified for all tasks. Qualifications are obtained by completing on the job (OJT) training under the supervision of an instructor. Whilst this is similar to the use and acquisition of qualifications in other industries, such as Nurse Rostering, the difference here is that controllers must maintain familiarity, or currency, in all the positions for which they are qualified. Currency is measured in days since a controller last worked in a particular position. This restriction requires that controllers must regularly work in all relevant positions, failure to do so results in the suspension of that qualification and leads to an enforced period of retraining, with a final competency check, before the qualification can be reactivated. Once a controller has worked productively in a position i.e. they have controlled a reasonable number of aircraft, the currency value is reset to zero. The objective function for the model is therefore to maximise the sum of currency for all assignments of a controller to a position.

Maintaining currency for all controllers is the central goal for any useful rostering approach. Other considerations such as minimising the cost of staffing a shift can be ignored, as staff are salaried by rank or grade and minimum staffing levels for each unit are determined by the relevant regulating authority. This means that the objective of the solution is to prevent controllers from violating currency limits, over time. This can be best achieved by forcing controllers to vary the positions worked as often as is reasonable.

Our model divides time into 30 minute blocks, and a controller should not work for more than 2 hours, followed by at least a 30 minute break as defined by SRATCOH. Staff can not directly swap positions because for each change of controller in a position a handover briefing is required, which requires that the relieving controller must have been unassigned in the previous time block. Although it would be valid to have controller A work for 2 hours, be relived by controller B for 30 minutes and for A to return to their original position, this increases the number of staff on a given shift . Ideally, B should stay in the position and A should be assigned another task, to maintain their currency.

3 Solution Approach

To produce valid rosters in reasonable time, a constructive, greedy heuristic algorithm has been developed, to meet the above requirements. The process begins with a set of input parameters based on controller availability, qualifications, demand for tasks and currency values. Every assignment in the model is defined as a tuple containing values for controller, position and time (c, p, t).

The algorithm is an iterative procedure which selects the first assignment from a sorted list of valid assignments at each step and appends this to the solution. If the list is empty the algorithm discards the last assignment to the solution, backtracks and restarts. The solution is a list of assignments, and it's index relates to the position and time slot that requires staffing i.e. solution(1) is for position(1), time slot(1), solution(2) is for position(2), time slot(1) etc.

The list of valid assignments is first sorted by the currency value of controllers, for the required position at each index of the solution, to maximise total currency for the roster. However, this ordering alone can cause problems in the final roster. Consider the situation where controller A is the least current in position APP. She is assigned to APP in the first 4 time slots and replaced by controller B in the 5th. The ordering by currency of valid assignments would mean that A would be reassigned to APP in the 6th time slot, to maximise currency. This is a poor approach, as B is not given enough time to reset his currency and would require a larger pool of available controllers to satisfy the staffing requirements. An additional preference rule is applied to the list that attempts to keep a controller in a position for as long as possible, subject to break rules, and reduces the possibility of having 2 controller changes in a position in consecutive time slots.

4 Experimental Results

A series of 240 single day rosters have so far been produced, with progressively more constrained starting conditions (fewer available controllers) and arbitrarily limited to a 60 minute time limit for a solution. Th algorithm is written in Python 2.7 and run on a consumer grade i7 desktop PC. The final 30 experiments with the fewest number of least qualified controllers (14 controllers to 11 positions) managed to produce valid rosters in an average time of 26 minutes. Table 1 shows an example of the algorithm output with controllers {A, B, ..., Z} assigned to positions {SUP, APP, ..., BKN SRA} for each timeslot {1, 2, ..., 18}. The score of 3,557 is the total sum of currency for all assignments, with a theoretical maximum value of 5,940. This upper value could never be achieved as it would require no controller to have worked in any position for 30 days, which is completely unrealistic.

To take controller M as an example, table 1 shows them assigned to four different positions for the day, therefore maximising the chance that they reset their currency in those positions. There are several other examples of con-

Table 1 Example roster: Controllers assigned to positions for given time slot. Currency score = 3557

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
SUP	V	V	V	V	T	T	T	T	V	V	V	V	T	T	T	T	V	V
APP	S	S	S	X	X	X	K	K	K	K	Y	Y	Y	Y	S	S	S	S
CWL DIR	T	T	T	O	O	O	O	X	X	X	X	K	K	K	K	Y	Y	Y
BKN DIR	O	O	Y	Y	Y	H	H	H	H	S	S	S	S	O	O	O	O	X
CWL DEPS	H	H	H	H	S	S	S	S	O	O	O	O	X	X	X	X	Z	Z
BKH DEPS	X	X	Z	Z	Z	Z	Y	Y	Y	T	T	Z	Z	Z	Z	H	H	H
ADC	M	M	B	B	B	B	E	E	E	E	H	H	H	H	E	E	E	E
GND	Y	E	E	E	E	W	W	W	W	Q	Q	Q	Q	M	M	M	M	T
PAR	W	W	W	W	Q	Q	Q	Q	M	M	M	M	B	B	B	B	K	K
CWL SRA	Q	Q	Q	M	M	M	M	B	B	B	B	E	E	V	V	W	W	W
BKN SRA	B	K	K	K	K	V	V	Z	Z	Z	W	W	W	W	Q	Q	Q	Q

trollers assigned to multiple positions in this roster, implying that the solution is of sufficient quality for use.

5 Future Work

There are several obvious extensions to the model that will require further investigation. Including training plans for new controllers is obvious next step. Other possible areas for research include:

– Allowing for breaks of different length e.g. lunch hours.
– Encoding personal preference for tasks/shifts.
– Embedding fairness into the rosters i.e evenly distributing time assigned to positions.
– Producing extended days rosters with overlapping shifts.
– Generalising the model for use in other industries e.g. airport security.

References

1. Authority, C.A.: Cap 670: Air traffic services safety requirements (2003)
2. Di Gaspero, L., Gärtner, J., Musliu, N.: A hybrid LS-CP solver for the shifts and breaks design problem. Hybrid . . . pp. 46–61 (2010). URL http://link.springer.com/chapter/10.1007/978-3-642-16054-7_4
3. Li, H., Womer, K.: Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. Journal of Scheduling **12**(3), 281–298 (2008). DOI 10.1007/s10951-008-0079-3. URL http://link.springer.com/10.1007/s10951-008-0079-3
4. Quimper, C.G., Rousseau, L.M.: A large neighbourhood search approach to the multi-activity shift scheduling problem. Journal of Heuristics **16**(3), 373–392 (2009). DOI 10.1007/s10732-009-9106-6. URL http://link.springer.com/10.1007/s10732-009-9106-6
5. Rekik, M., Cordeau, J., Soumis, F.: Implicit shift scheduling with multiple breaks and work stretch duration restrictions. Journal of scheduling (2010). URL http://link.springer.com/article/10.1007/s10951-009-0114-z
6. Widl, M., Musliu, N.: An improved memetic algorithm for break scheduling. Hybrid Metaheuristics pp. 133–147 (2010). URL http://link.springer.com/chapter/10.1007/978-3-642-16054-7_10

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A Matheuristic Approach for the High School Timetabling Problem

Árton P. Dorneles · Olinto C.B. Araújo · Luciana S. Buriol

The school timetabling is a classic optimization problem that consists of scheduling a set of class-teacher meetings in a prefixed period of time, satisfying requirements of different types. Due to the combinatorial nature of this problem, solving medium and large instances of timetabling to optimality is a challenging task. When resources are tight, it is often difficult to find even a feasible solution. In this study, we propose a matheuristic approach for the high school timetabling problem that combines a fix-and-optimize heuristic with a variable neighborhood descent method. The experimental results show that our approach provides high quality feasible solutions in a short computational time when compared with results obtained with the mixed integer programming solver CPLEX. In addition, we have improved best known solutions of 7 out of 12 instances from the literature. Among them, three are new optimal solutions for classical instances that have been available since 2000.

Keywords High School Timetabling · Matheuristics · Fix-and-optimize · Variable Neighborhood Descent

Árton P. Dorneles · Luciana S. Buriol
Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre - Brazil
E-mail: arton.dorneles@inf.ufrgs.br
E-mail: buriol@inf.ufrgs.br

Olinto C.B. Araújo
Colégio Técnico Industrial de Santa Maria, Universidade Federal de Santa Maria, Santa Maria - Brazil
E-mail: olinto@ctism.ufsm.br

FIFA Ranking and World Cup Football Groups: Quantitative Methods for a Fairer System

Guillermo Durán · Sebastián Cea · Mario Guajardo · Denis Sauré and Gonzalo Zamorano

Abstract The team seeding and final group distributions for the final tournament of the 2014 World Cup football competition have provoked serious criticism in the international football community. Much of the discontent has been directed at the formula underlying the FIFA ranking used to designate the 8 seeded teams. Particularly surprising for many in the football world

Keywords FIFA Ranking · Integer Programming · Numerical Simulations · World Cup Football

First author is partly financed by project nos. ANPCyT PICT-2012-1324 (Argentina), CONICET PIP 112-200901-00178 (Argentina), UBACyT 20020100100980 (Argentina), and FONDECyT 1140787 (Chile), and by the Complex Engineering Systems Institute (ISCI, Chile).

Guillermo Durán
Instituto de Cálculo and Departamento de Matemática, FCEN, UBA, Argentina; Departamento de Ingeniería Industrial, FCFM, Universidad de Chile, Chile.
E-mail: gduran@dm.uba.ar

Sebastián Cea
Departamento de Ingeniería Industrial, FCFM, Universidad de Chile, Chile.
E-mail: seba.cea.b@gmail.com

Mario Guajardo
Department of Business and Management Science, NHH Norwegian School of Economics, N-5045 Bergen, Norway.
E-mail: mario.guajardo@nhh.no

Denis Sauré
Departamento de Ingeniería Industrial, FCFM, Universidad de Chile, Chile.
E-mail: dsaure@dii.uchile.cl

Gonzalo Zamorano
Departamento de Ingeniería Industrial, FCFM, Universidad de Chile, Chile.
E-mail: gonzaloz@dii.uchile.cl

was that the seeds include national sides like Colombia and Switzerland while undisputed powerhouses such as Italy and Holland were left out.

Another focus of negative comment in sports media around the world is the heavily geographical criteria used by FIFA to determine the makeup of the 4 ”pots” for the draw procedure that defined the composition of the team groups. Such criteria tend to result in groups that are highly uneven, and this is clearly reflected in the 2014 World Cup assignments. While one group contains football powers Uruguay, Italy and England and a second group includes Spain, Holland and Chile, others are globally much weaker such as the one lumping third-seeded Argentina in with Bosnia, Iran and Nigeria, three countries with relatively little football tradition.

This study has two principal goals. The first is to make a series of adjustments to the current FIFA ranking formula, thus producing a new ranking that resolves the main problems in the official version. Among the more significant deficiencies corrected are the lack of due consideration for the home-away status of matches, the fact that a team is better off not playing friendlies than playing them and winning, the fact that deciding a friendly by a penalty shootout is more advantageous because neither team has anything to lose, and the reality that a team has more to gain by defeating San Marino (the weakest team in the ranking) at home in a World Cup or UEFA European Championship qualification match than tying with Spain (the strongest team in the ranking) in a friendly away game. To develop a formula more in keeping with what football experts expect from a team ranking, a number of numerical simulations are presented that correct key parameters accordingly.

The second goal of this article is to address the problem of generating more equitable team groups. To this end we first calculate the ”strength” of each national team by adjusting our proposed new FIFA ranking to take into account its historical performance in every World Cup and continental cup tournament. Based on this adjusted ranking we then assign the 8 seeded teams (i.e., the 8 strongest) to pot 1 and then define the other 3 pots as follows: pot 2 contains the countries ranked 9th to 16th, pot 3 contains the countries ranked 17th to 24th and pot 4 contains the countries ranked 25th to 32th. Finally, we design an integer linear programming model that allocates one team from each pot to the four groups. FIFA’s geographical conditions limiting the number of European teams to no more than 2 per group and the number from any other continent to no more than 1 per group are included in the model as constraints. The objective function minimizes the difference between the final strength of the strongest and the weakest groups, the strength of a group being defined as the sum of the individual strengths of the group’s teams. The idea is to achieve a set of groups that are as balanced as possible. To incorporate FIFA’s wish that the final result contain an element of randomness, that is, that it not be derived deterministically, the N best solutions are generated (where N is between 5 and 10) rather than just the optimal solution, and the definitive one is then decided by a weighted draw.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Predictive scheduling for optimal cloud configuration

Michael G. Epitropakis ·
Andrea Bracciali · Marco Aldinucci ·
Emily Potts · Edmund K. Burke

1 Introduction

Cloud Computing provides availability of a large amount of resources in a seamless and tailored way to a vast public of users, following the paradigm of computation as a utility, which, like all the more traditional utilities, comes with an associated cost. Users can access resources through facilitated interfaces supported by the vast majority of service providers. However, non-trivial Cloud Computing usually requires advanced skills to configure and manage the computation in an optimal way. Configuration and management of the available resources might include either selection of "optimal" parameters for the utilized software in order to efficiently use the available resources, or selection of the "optimal" hardware resources in order to minimize renting cost and maximize efficiency.

Very relevant is the trade-off between efficiency and costs, which is also typically quite difficult to manage, because of intrinsic unpredictability of computation and the consequent lack of accurate cost models for these architectures. Even if the unit cost of different architecture components, e.g. bandwidth, different classes of virtual machines and storage, is known, determining beforehand the exact cost of running a complex parallel or distributed application over different, possibly dynamic, configurations is currently an open problem.

Several approaches address the problem as a multi-objective optimisation problem on (very) simplified *"theoretical models"* of the architecture. These models are general and typically unrelated to the specific performances of the

Michael G. Epitropakis · Andrea Bracciali · Emily Potts · Edmund K. Burke
Computing Science and Mathematics,
University of Stirling, UK.
E-mail: {mge,abb,epo}@cs.stir.ac.uk, e.k.burke@stir.ac.uk

Marco Aldinucci
Computer Science Department,
University of Torino, Italy.
E-mail: aldinuc@di.unito.it

architecture for the specific *problem at hand*. For instance, in [8] different virtual machines are abstracted in terms of their brute Gb-per-second theoretical processing capability.

Another class of approaches is based on *"experimental models"*, which build on top of knowledge about past executions. For instance, in [6] past executions are clustered in classes of problems, whose optimal time and cost configuration can be inferred from past experience. Then, mapping the *problem at hand* to one of such classes gives indications about how to configure Cloud Computing architectures for best time and cost performances.

A critical aspect of the above modelling approaches is the difficulty, i.e. the cost ultimately, of collecting suitable information for devising accurate and specific cost models for the *problem at hand*. Theoretical approaches by-pass such costs, losing accuracy. Experimental approaches increase accuracy, but at the cost of accumulating enough experience and hence potentially undermining the cost-optimisation goals. Furthermore, properly mapping the specific problem under consideration to the so-far available experience base might be difficult and introduces back a loss of accuracy.

Bridging the two approaches, we investigate the possibility of building a model from *limited experimental information* on a specific problem, on top of which we build a "theoretical" model. The model, then, reflects some specific information about the *problem at hand*, hence improving domain knowledge with respect to the theoretical approaches, but the price needed to accumulate large experimental knowledge is waived.

On top of our model, we perform state of the art evolutionary multi-objective optimisation to discover optimal configurations. Multi-objective optimization formulations have been well established and widely adapted in various fields of Software Engineering [5], such as Requirements Engineering [10], Software Testing [3, 9] and Cloud Computing [4].

2 Combining models and experimental data for optimised cost predictions

We focus on (simplistic) variants of the *MapReduce* programming model [1], as provided by the Amazon EC2 Cloud. MapReduce is very relevant in Cloud Computing as it provides a high-level programming interface easing the development of efficient, scalable and robust applications and services.

We initially consider three benchmark problems, one CPU bounded, one I/O bounded and one making a large use of bandwidth. These broad categories characterise several aspects of Cloud Computing, other choices are possible. For each benchmark, a minimal configuration of the problem, a single task, is executed in one instance, i.e. a virtual machine, provided by EC2. Small, medium and large instances are available with different performances and costs. We hence use *averaged* minimal tasks, i.e. uniform representatives of tasks computational requirements, to *experimentally* gauge instances performances on the specific problem at hand. Then, we develop a *theoretical* model that gives an approximation of how performances scale up when many tasks are executed on combinations of several instances of different kinds. The models are based on theoretical limits of performance scaling, and may consider, where appropriate, the cost of distributing data and collecting results, again inferred from simple (simplistic) experimental measurements, the costs associated to storage, and specific costing policies, such as pricing per temporal intervals of a given duration, and other problem- and architecture-specific information.

On top of our model, we formulate a multi-objective optimisation to tackle resource allocation and task scheduling. We use multi-objective optimisation analysis to determine optimal configurations of resources (number and kind of instances) and schedule of tasks. More precisely, we aim at minimising both *execution costs* for the MapReduce execution, and its *completion time* (makespan). Although, we focus on these two objectives, the formulation is general and capable of handling more objectives, based on the optimisation needs of the user.

In the current study, we utilise two well known evolutionary multi-objective algorithms that have demonstrated efficient behaviour in Search-based Software Engineering optimisation problems, namely the Non-dominated Sorting Genetic Algorithm (NSGA-II) [2], and the Two-Archive algorithm (TAEA) [7]. NSGA-II uses a fast non-dominated sorting procedure to enhance the diversity and the quality of the resulting solutions, while TAEA incorporates two archives with different properties to save potentially good solutions. The former aims at enhancing the diversity of the solutions found, while the latter aims at enhancing the convergence of the algorithm to the real Pareto front. In the current setting the representation of a solution encodes both the resource allocation and task scheduling goals, while both algorithms have been used with their default search and parameter settings.

An extensive empirical verification of the proposed methodology has been designed. The algorithmic part of the methodology can be enriched by developing an adaptive hyper-heuristic version of the algorithms mentioned above to enhance their searching abilities and their performance in terms of locating optimal and diverse solutions.

A key challenge for the proposed methodology is to enable the user to make fast and cost/performance optimal decisions when configuring Cloud Computing services for novel MapReduce based problems. User decisions will be less dependent on costly and hard-to-generalise past experimental information than the decisions based on traditional "experimental approaches", and a bit more informed on the problem at hand than the decisions based on "theoretical approaches".

Acknowledgements E.K. Burke and M.G. Epitropakis would like to thank EPSRC for their support for this work through grant EP/J017515/1.

References

1. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6, OSDI'04, pp. 10–10. USENIX Association, Berkeley, CA, USA (2004)
2. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)
3. Harman, M.: Making the case for morto: Multi objective regression test optimization. In: Fourth International IEEE Conference on Software Testing, Verification and Validation, ICST 2012, Berlin, Germany, 21-25 March, 2011, Workshop Proceedings, pp. 111–114. IEEE Computer Society (2011)
4. Harman, M., Lakhotia, K., Singer, J., White, D.R., Yoo, S.: Cloud engineering is search based software engineering too. Journal of Systems and Software **86**(9), 2225–2241 (2013)
5. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys **45**(1), 11:1–11:61 (2012)
6. Lama, P., Zhou, X.: Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In: Proceedings of the 9th International Conference on Autonomic Computing, ICAC '12, pp. 63–72. ACM, New York, NY, USA (2012)
7. Praditwong, K., Yao, X.: A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm. In: Y. Wang, Y.m. Cheung, H. Liu (eds.) Computational Intelligence and Security, *Lecture Notes in Computer Science*, vol. 4456, pp. 95–104. Springer Berlin Heidelberg (2007)
8. Tsai, J.T., Fang, J.C., Chou, J.H.: Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Computers & Operations Research **40**(12), 3045–3055 (2013)
9. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification & Reliability **22**(2), 67–120 (2012)
10. Zhang, Y., Finkelstein, A., Harman, M.: Search based requirements optimisation: Existing work and challenges. In: B. Paech, C. Rolland (eds.) Requirements Engineering: Foundation for Software Quality, *Lecture Notes in Computer Science*, vol. 5025, pp. 88–94. Springer Berlin Heidelberg (2008)

Personalized nurse rostering through linear programming *

Han Hoogeveen Tim van Weelden
Department of Information and Computing Sciences
Utrecht University
P.O. Box 80089, 3508 TB Utrecht, The Netherlands.
j.a.hoogeveen@uu.nl, timvanweelden@gmail.com

1 Introduction

Our purpose is finding good, personalized rosters for the personnel of the Cardiothoracic Department of UMC Utrecht. The department contains several treatment rooms at different care levels. Taking care of the patients is done 24/7 by approximately 52 nursing employees, each of which is available for a given number of hours, which differs per nurse. There are four types of nurses: student nurses, basic nurses, Medium Care nurses, and Senior nurses. The student nurses are trainees, who work mostly in the day-shift. The basic nurses have a basic qualification only; they can work in all shifts. Next to the basic qualification, the MC and senior nurses have an MC/senior qualification.

The day is divided in three shifts: day (early) shift, late shift, and night shift. For all shifts in the planning period, we know the necessary number of nurses per qualification.

The nurses each have there own preferences regarding their schedule; the hospital tries to obey these as much as possible. Rosters are created per 6-week period. Currently, this is done by hand, since the planning system in use is not able to find reasonable rosters. We present an ILP-based algorithm to solve this problem; it is based on the approach in [2] that was used to solve a simpler problem without qualifications. Moreover, we present a repair heuristic to adapt to changing nurse availability.

Our contribution. We describe an algorithm that can find near-optimal solutions for real-life rostering problems while taking personal preferences into account. This hot topic (in the Netherlands) has not been well-studied in the literature (see [1,2]).

*The working paper this abstract is based can be found at http://www.staff.science.uu.nl/ hooge109/Weelden.pdf

2 Constraints

The goal is to find a feasible roster for each employee such that there are enough nurses with the right qualifications in each shift, and nurse preferences are satisfied as much as possible. Having surplus employees is allowed, preferably well-spread over the day-shifts, but never for night-shifts. Moreover, the presence of student nurses should be evenly divided over the day-shifts.

The individual rosters have to obey several hard constraints. First of all, the number of workshifts should equal the appointment size, but a deviation of 1 is allowed; moreover, every employee should work at most 6 days per week. Next, the night-shifts must be rostered in prespecified blocks. There should be at most 4 night-shifts per 6 weeks, and after a night-shift the nurse must have at least two days off. Furthermore, a nurse must have the same shift on Saturday and Sunday. Moreover, days off must come in blocks of size two or more. Because of healthy rostering, a late-shift cannot be followed by a day-shift. Finally, senior nurses are entitled to two 'quality days'.

Furthermore, there are many soft constraints concerning individual rosters, which are used to determine the quality of the individual rosters. In general, the number of shifts per week should be the same each week, and the night-shifts and weekend-shifts should be well spread over the 6-week period. Examples of personal preferences are to have a regular or occasional day off, to have a specific shift on a day (some nurses perform self-scheduling), to have as many/few late/night shifts as possible, to have as many days off after night-shift as possible, and to have bounds on the number of consecutive shifts.

3 Solution approach

The basic idea (see [2]) is to formulate the problem as an ILP. For each nurse, we construct a set of rosters; each one satisfies all hard constraints issued by both the hospital and the nurse. In constrast to [1], we generate these up to 200.000 rosters per nurse beforehand. For each roster we compute its cost on basis of how well it respects the personal preferences of that nurse. The cost is then scaled to $[0, 1]$ and the ones with cost > 0.5 are removed.

For each roster $s \in S$ we introduce a binary variable x_s indicating whether s is chosen. Roster s contains information as to which nurse it belongs and which shifts it covers. Using these variables, we formulate the constraint that each nurse gets one roster, and that for each combination of a shift and a qualification, we meet the minimum occupancy. We further have variables measuring shortage, surplus, and excessive surplus per shift per qualification; these are penalized in the objective function. The objective is to minimize the total cost of the chosen rosters plus the total shortage, surplus, and excessive surplus penalty.

As the ILP cannot handle these large numbers of variables, we restrict ourselves to a selection of these rosters, for which we then solve the ILP. To make this selection, we solve the LP-relaxation through revised simplex; we put each variable that gets selected in the revised simplex in a column pool. Furthermore, we add variables to this pool that have small reduced cost based on the dual multipliers of the optimum solution. Moreover, we find additional columns for the pool by applying small mutations. Finally, we let the ILP run for this pool of rosters for 15 minutes. In the resulting solution almost all occupancy constraints are met (sometimes one weekend night-shift needs one more nurse) with good rosters (most of the soft preferences were fulfilled, but sometimes there was a single day off).

We then implemented a set of heuristics for repairing roster problems, which also can be used to resolve new requests. The philosophy behind these was to make small changes, where each change would resolve an 'important' problem at the expense of creating an 'unimportant' problem. In this way, we can also resolve requests like people who should cooperate regularly. Furthermore, in this way we can determine consecutive 6-weeks rosters independently, which are then glued together.

References

[1] F. He and R. Qu (2012). A constraint programming based column generation approach to nurse rostering problems. *Computers & Operations Research, 39*, pp. 3331–3343.

[2] H. Hoogeveen and E. Penninkx (2007). Finding near-optimal rosters using column generation. Technical Report UU-CS-2007-002, Department of Information and Computing Sciences, Utrecht University. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.8622&rep=rep1&type=pdf

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

The Impact of Reserve Duties on Personnel Roster Robustness: An Empirical Investigation

Jonas Ingels · Broos Maenhout

Keywords Personnel scheduling · Robustness · Discrete-event simulation · Branch-and-price

1 Introduction

Robustness in personnel scheduling is an important aspect in organisations for a variety of reasons, among which service level objectives, cost minimisation [6] and an increasing focus on satisfying individual employee preferences [2]. First, every organisation wants to sustain a certain *service level* towards its customers. This service level objective requires organisational flexibility to deal with unexpected events, which can be caused by uncertainty of capacity, demand and arrival [3]. Unexpected events that require roster changes are called disruptions, with which organisations can cope through a variety of policies, including reserve duty conversion, overtime, assignment of employees with a day off and acceptance that customer demand cannot be fully met [10]. Reserve duty conversion is the conversion of a reserve duty into a working duty. Second, *personnel costs* represent a large portion of the operating costs of an organisation and even a small percentage-decrease can result in a significant

Jonas Ingels
Faculty of Economics and Business Administration, Ghent University
Tweekerkenstraat 2, 9000 Gent (Belgium)
Tel.: +32-9-264 89 81
Fax: +32-9-264 42 86
E-mail: Jonas.Ingels@Ugent.be

Broos Maenhout
Faculty of Economics and Business Administration, Ghent University
Tweekerkenstraat 2, 9000 Gent (Belgium)
Tel.: +32-9-264 98 32
Fax: +32-9-264 42 86
E-mail: Broos.Maenhout@Ugent.be

cost reduction [3]. However, disruptions can cause important deviations between the planned and actual costs of the personnel roster. Planned costs are the estimated costs when the personnel roster is composed, while the actual costs are the costs incurred during the execution of the roster. If we make abstraction of uncertainty and variability during the construction of the personnel roster, the planned costs are minimal but this may lead to significantly higher actual costs.
Third, the adaptations to the original personnel roster, which are required by disruptions in order to restore feasibility, potentially lead to a lower *personnel satisfaction*. This objective is an important objective in the personnel scheduling literature [2,11,6] and should therefore be taken into account.
Hence, it is important for organisations to consider uncertainty, which helps them to construct rosters that are more robust. These rosters enable organisations to better achieve their desired service level objectives, cost objectives and employee satisfaction objectives.
In this paper, we evaluate the impact of reserve duties on the robustness of personnel rosters through discrete-event simulation and aim to derive insight into how reserve duties help organisations construct robust rosters.

2 Problem Definition

Personnel scheduling consists of three phases [9]: the strategic, tactical and operational phase. The strategic phase focuses on the long term, during which the personnel mix is determined through hiring, firing and training. The decisions made during this phase serve as input to the tactical phase. This medium-term phase encompasses the construction of a personnel roster for a mid-term period, which, in turn, serves as input to the operational phase. In this phase allocations are made for the next 24-hour period.
Since robustness has both a proactive and a reactive component [8], we examine the relationship between the tactical and operational phase. Thus, the goal is to construct a tactical roster that can absorb disruptions during the operational phase. Nevertheless, a roster that fully absorbs all disruptions is costly, which is why an efficient reactive mechanism is crucial to overcome disruptions with as few changes to the original roster as possible. In order to achieve this, we focus on the assignment of reserve duties during the tactical phase and the conversion of these reserve duties during the operational phase.

2.1 Tactical phase

In the tactical phase, we assume that all input is known and deterministic. Given the estimated daily shift requirements, we create individual personnel schedules, which consist of a collection of 3 possible assignments: a working duty, a reserve duty and a day off.
In order to obtain general results, this paper investigates a general personnel

scheduling problem. Therefore, the personnel information, objectives and constraints are all general and common in literature [4]. Our tactical model can be classified as $ASB\ N|RV\ 3|LRG$ [5].

2.2 Operational phase

In the operational phase, the allocations for the next 24 hours are performed. Due to uncertainty, these allocations may differ from the original assignments made in the tactical roster. Since our focus is to study the impact of reserve duties on the robustness of personnel rosters, we only explore the conversion of reserve duties into working duties to adhere to the original roster while trying to maintain the service level at minimal cost and minimal preference violations.

3 Methodology

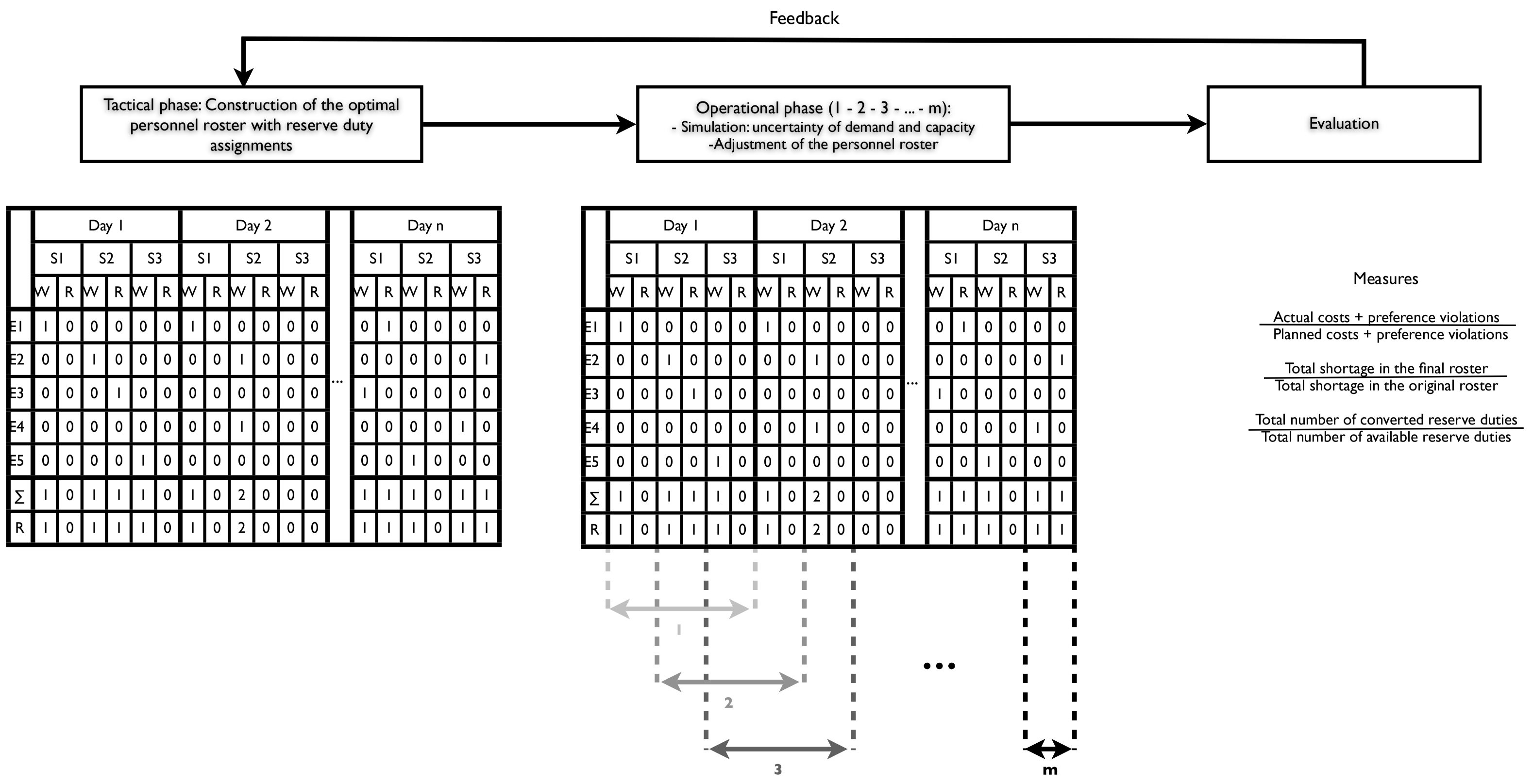
Our research methodology is composed out of three different steps, which are explained below. Figure 1 gives an overview of this methodology.

3.1 Tactical phase: Construction of the optimal personnel roster with reserve duty assignments

The tactical roster is constructed using a branch-and-price algorithm. We test instances with a planning period of at least 7 and at most 28 days. The number of employees varies but is limited to 20.
The set of constraints that we consider can be classified into coverage constraints and time-related constraints [4]. Both these constraints include constraints for working duties and reserve duties.
The objectives used are related to the minimisation of assignment costs, preference violations, shortages and surpluses in working and reserve duties.

3.2 Operational phase: Simulation and adjustment of the personnel roster

In the operational phase, we use a 24-hour rolling horizon framework [1] that consists of two iterative steps: simulation and adjustment. During the simulation step, the uncertainty of demand and capacity are simulated, potentially causing understaffing and overstaffing. In the adjustment step, we evaluate if changes are necessary and possible through reserve duty conversion. The decisions for the next 8 hours are executed and we proceed 8 hours to perform the same steps for the following 24 hours. This process is repeated until the last 8 hours of the tactical roster are simulated and adjusted.


Feedback
Tactical phase: Construction of the optimal personnel roster with reserve duty assignments
Operational phase (1 - 2 - 3 - ... - m):
- Simulation: uncertainty of demand and capacity
-Adjustment of the personnel roster
Evaluation
Measures
1
2
3
m

	Day 1						Day 2							Day n					
	S1		S2		S3		S1		S2		S3			S1		S2		S3	
	W	R	W	R	W	R	W	R	W	R	W	R		W	R	W	R	W	R
E1	1	0	0	0	0	0	1	0	0	0	0	0		0	1	0	0	0	0
E2	0	0	1	0	0	0	0	0	1	0	0	0		0	0	0	0	0	1
E3	0	0	0	1	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0
E4	0	0	0	0	0	0	0	0	1	0	0	0		0	0	0	0	1	0
E5	0	0	0	0	1	0	0	0	0	0	0	0		0	0	1	0	0	0
Σ	1	0	1	1	1	0	1	0	2	0	0	0		1	1	1	0	1	1
R	1	0	1	1	1	0	1	0	2	0	0	0		1	1	1	0	1	1

	Day 1						Day 2							Day n					
	S1		S2		S3		S1		S2		S3			S1		S2		S3	
	W	R	W	R	W	R	W	R	W	R	W	R		W	R	W	R	W	R
E1	1	0	0	0	0	0	1	0	0	0	0	0		0	1	0	0	0	0
E2	0	0	1	0	0	0	0	0	1	0	0	0		0	0	0	0	0	1
E3	0	0	0	1	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0
E4	0	0	0	0	0	0	0	0	1	0	0	0		0	0	0	0	1	0
E5	0	0	0	0	1	0	0	0	0	0	0	0		0	0	1	0	0	0
Σ	1	0	1	1	1	0	1	0	2	0	0	0		1	1	1	0	1	1
R	1	0	1	1	1	0	1	0	2	0	0	0		1	1	1	0	1	1

$$\frac{\text{Actual costs + preference violations}}{\text{Planned costs + preference violations}}$$

$$\frac{\text{Total shortage in the final roster}}{\text{Total shortage in the original roster}}$$

$$\frac{\text{Total number of converted reserve duties}}{\text{Total number of available reserve duties}}$$

Fig. 1 Methodology

3.3 Evaluation and feedback

A first important measure for robustness is based on the planned/actual costs and preference violations.

$$RM_1 = \frac{Actual\,costs\ +\ preference\,violations}{Planned\,costs\ +\ preference\,violations} \tag{1}$$

A second important measure is the service level the organisation can offer to its customers and is based on the solution quality evaluation in [7]. Therefore, we compare the shortages in the original roster to the shortages in the final roster.

$$RM_2 = \frac{Total\ shortage\ in\ the\ final\ roster}{Total\ shortage\ in\ the\ original\ roster} \tag{2}$$

A third measure is related to the conversion of reserve duties into working duties at the time of execution.

$$RM_3 = \frac{Total\ number\ of\ converted\ reserve\ duties}{Total\ number\ of\ available\ reserve\ duties} \tag{3}$$

These measures indicate the robustness of our original roster and we can determine them for each day or even each shift. Furthermore, it is possible to split the third measure up per employee. This indicates to what extent an employee is efficiently used. This information is then used to change the objective function coefficients and the constraint parameter values of the related reserve variables and reserve constraints in the tactical phase, after which the process is repeated.

References

1. Bard, J., Purnomo, H.: Hospital-wide reactive scheduling of nurses with preference considerations. IIE Transactions **37**, 589–608 (2005)
2. Bard, J., Purnomo, H.: Short-term nurse scheduling in response to daily fluctuations in supply and demand. Health Care Management Science **8**, 315–324 (2005)
3. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**, 367–385 (2013)
4. Burke, E., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The state of the art of nurse rostering. Journal of Scheduling **7**, 441–499 (2004)
5. De Causmaecker, P., Vanden Berghe, G.: A categorisation of nurse rostering problems. Journal of Scheduling **14**, 3–16 (2011)
6. Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research **153**, 3–27 (2004)
7. Koutsopoulos, H.N., Wilson, N.H.: Operator workforce planning in the transit industry. Transportation Research Part A: General **21**(2), 127–138 (1987)
8. Maenhout, B., Vanhoucke, M.: Reconstructing nurse schedules: Computational insights in the problem size parameters. Omega - International Journal of Management Science **41**, 903–918 (2013)

9. Maier-Rothe, C., Wolfe, H.: Cyclical scheduling and allocation of nursing staff. Socio-Economic Planning Sciences **7**, 471–487 (1973)
10. Shebalov, J., Klabjan, D.: Robust Airline Crew Pairing: Move-up Crews. Transportation Science **40**, 300–312 (2006)
11. Topaloglu, S., Selim, H.: Nurse scheduling using fuzzy modelling approach. Fuzzy Sets and Systems **161**, 1543–1563 (2010)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Lessons from Building an Automated Pre-Departure Sequencer for Airports

**Daniel Karapetyan · Andrew J. Parkes ·
Jason A.D. Atkin · Juan Castro-Gutierrez**

Abstract Commercial airports are under increasing pressure to comply with the Eurocontrol Collaborative Decision Making (CDM) initiative, to enable overall airspace improvements. An important element of a CDM system is the provision of automated decision support to aid the controllers to schedule the take-off times and the associated times at which aircraft should push back from the stands. The CDM system then aids effective operations by communicating these scheduling decisions to other relevant parties within the airport and the airspace. One of the major CDM components is aimed at predicting the target take-off times; for medium-sized airports, a common choice for this is a "Pre-Departure Sequencer" (PDS). Here we describe the design and requirements challenges which arose during our development of a PDS system. Firstly, the scheduling problem is highly dynamic and event driven. For example aircraft can be delayed or runway capacity can change, and this requires a careful separation of data ownership responsibility between the system components and special attention to integrity constraints. Secondly, it is important to end-users that the system be predictable and, as far as possible, and transparent in its operation, with decisions that can be explained. These human factors, which influenced the choice of methods for solving the problem, are also explained in this abstract, along with the consequent decisions which were made.

Keywords Airport Operations · On-line Scheduling · Algorithm Design

1 Introduction

Each departing flight in an commercial airport typically follows the following steps:

This work was supported in part by an EPSRC 'Research Development Fund' and also by EPSRC grant EP/F033613/1

School of Computer Science, University of Nottingham
E-mail: {dxk, ajp, jaa, jpc}@cs.nott.ac.uk

1. The aircraft crew and other airport services report the time at which the flight will be ready to depart, called the TOBT.
2. The airport controllers schedule the 'off-block' / 'push back' times based upon these TOBTs and other information. The procedure is usually for a tug to push the aircraft back from the stand, since most of the aircraft are not capable of moving in the reverse direction without risking damage to the stands.
3. The aircraft is pushed back and its engines are turned on.
4. The aircraft taxies towards the end of the runway where it may join a queue of departing aircraft.
5. The aircraft taxies onto the runway, lines up and takes off.

Operations need to be coordinated, so these stages should be scheduled in advance and the relevant timings for the key stages distributed to those people and systems that need to know. To aid this, organisations such as Eurocontrol[1], who are responsible for the management of the airspace over Europe, have introduced and promoted "Collaborative Decision Making" (CDM) systems [2]. In particular, from a Eurocontrol brochure about "Airport CDM" (A-CDM) [3], some key aims are:

> *"Information sharing is the first and most essential element of A-CDM as it creates the foundation by creating a common situational awareness. In addition, it potentially brings predictability and resource efficiency benefits. ... With the pre-departure sequencing function the target start-up approval time (TSAT) can be calculated, providing an off-block sequence."*

Commercial airports are under increasing pressure to ensure that they have a minimum of a "Pre-Departure Sequencer" (PDS) system deployed within the Air Traffic Control (ATC) systems of the airport. The primary responsibility of the PDS is to predict the basic information as to the time at which aircraft plan take off. This time is called the "Target Take-Off Time" (TTOT), and is usually linked to the time when the aircraft plan to start engines to move towards the runway ("Target Start-up Approval Time", TSAT). A key input to the PDS is the "Target Off-Block Time" (TOBT) which is the time specified by the airline operator at which they plan to be ready to leave. This is initially taken from published flight schedules, but can be modified due to operational reasons. The PDS system, optionally with the manual intervention of the ATC, is then responsible for taking the TOBTs together with information about current airport conditions and capacities and producing the TSATs and TTOTs.

Another responsibility of PDS is to comply with the Eurocontrol instructions. To manage bottlenecks in the airspace, Eurocontrol may declare a flight *regulated* and issue a so-called "Calculated Take-Off Time" (CTOT), which defines a time window for the flight departure. A CTOT window is a hard constraint; if the flight is not ready to depart before the end of the CTOT window,

[1] `https://www.eurocontrol.int/`

a new CTOT usually has to be requested, which may cause additional delays and associated costs. In other words, any plan which would violate a CTOT is strongly undesirable for the airport.

For the largest airports, a more complex "Departure Management" (DMAN) system becomes a standard choice. The fuller DMAN system being needed when the schedules need to be optimised with consideration of additional requirements such as wake-vortex separation rules, when runways cannot be used in mixed mode but capacity must be maximised, see [1]. However, for medium-sized airports a PDS system is a simpler, and so better, choice, especially when it is to be the first CDM system at the airport.

In this abstract, we discuss some lessons learned from building a pre-departure sequencer intended for a moderate size airport, that can automate most of the airport controller operations related to pre-departure sequencing, and can take and respect user modifications from the ATC. We believe that some of the lessons are also relevant to other scheduling and timetabling problems. In Section 2, we briefly describe the problem, and outline our approach. Finally, in Section 3, we report our conclusions.

2 Designing an Automated Pre-Departure Sequencer

A pre-departure sequence needs to provide TSAT values that obey constraints such as:

- $\text{TSAT} \geq \text{TOBT}$ (the aircraft cannot be pushed back before it is ready);
- TSAT cannot be in the past for any flight at the stand;
- $\text{TTOT} = \text{TSAT} + \text{EXOT}$ (where EXOT is the time needed for the aircraft to reach the runway);
- $\text{CTOT} - 5 \leq \text{TTOT} \leq \text{CTOT} + 10$.

Below we describe the main challenges we faced while designing PDS.

Minimal Perturbation. Besides the above constraints, one of the most important aspects of the PDS is the interaction with humans, for example, with decision makers in the ATC and ground operations. This has the immediate consequence that the system decisions should not 'churn': TSAT values should not be changed more than is needed, as constant updates lead to difficult and inefficient operations. This is an important criterion in the algorithm design.

'Predictability' and 'Explainability' of Decisions. It was also important that the human aspects required the PDS decisions to be predictable, repeatable, and potentially explainable to people that are experts in ATC, but not experts in algorithms or search. If the PDS is stochastic, then the exact outcome is unpredictable, which can be very disconcerting for operators and also makes the software testing phase both onerous and complex, or impossible to guarantee. Also, there should be the potential for decisions to be given explanations that make sense to the human experts. These issues limited the choice of algorithms that would be appropriate, for example, a standard stochastic local search would be a last resort — as it tends to be non-repeatable, and also very difficult to explain or justify the final decisions.

Data Ownership Issues. Another important lesson was the need to decide as early as possible upon the division of responsibility over data between the PDS and the main CDM database: what information was sent, which system 'owned' it and which had authority to make changes.

Overall, this led us to an event-driven rule-based approach; though with multiple passes through carefully designed sets of rules, and various triggers corresponding to circumstances such as the runway capacity changing. We do not present the algorithm here, but it is based on splitting the runway resource into time slots of the same lengths, which are computed from maximum number of take-offs per hour as provided by the ATC. The sequence is updated in reaction to events such as 'a new flight is declared', 'EXOT is changed' or 'controller reallocated a flight'. Such a system avoids unnecessary alteration of flights and has an easy to understand behaviour. With a flexible system of flight locks, we guarantee a certain level of predictability and transparency.

3 Conclusions

We have introduced the main issues that influenced the design of a decision support system for automated on-line pre-departure sequencing. Such a system can significantly improve many aspects of airport operations. Apart from obeying the basic constraints, the system keeps the number of changes in the pre-departure sequence to the minimum and has easily predictable and explainable behaviour. The traditional focus of OR optimisation projects is on the problem alone; however, one of our main lessons was that the "meta-problem" of the human context, with the need for development of high trust levels in the autonomous operations, had an important influence on the user acceptability of different algorithms.

Acknowledgements We would like to thank Ultra Electronics Limited for their collaboration with us in building a prototype of the automated pre-departure sequencer.

References

1. Jason A D Atkin, Geert De Maere, Edmund K Burke, and John S Greenwood. Addressing the pushback time allocation problem at heathrow airport. *Transportation Science*, 47(4):584–602, 2013.
2. EUROCONTROL. European Airport CDM: Airport Collaborative Decision Making. Website: `http://www.euro-cdm.org/` (last accessed Feb 2014).
3. EUROCONTROL and ACIEurope. A-CDM Airport Collaborative Decision Making, 2010. Brochure available at: `http://www.euro-cdm.org/library/cdm_brochure.pdf`.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Integrated Student Sectioning

Jeffrey H. Kingston

Abstract The assignment of students to sections has always seemed different from the rest of timetabling. Courses demand times, teachers, and rooms, but not students; instead, students demand courses. This paper shows how to remove this difference and integrate student sectioning with the rest.

Keywords Timetabling · Student sectioning

1 Introduction

When demand for some course is greater than the maximum class size, the course has to be broken into *sections*: copies of the course. Each student who enrols in the course must be assigned to a section, and this is the *student sectioning problem*. It arises both in universities [1,7] and high schools [2,5].

Student sectioning has always seemed different from the rest of timetabling. Courses demand times and resources (teachers, rooms, etc.), but not students; instead, students demand courses. This point is clarified below (Sect. 2).

One negative effect of this perceived difference is that good ideas developed on one side of the barrier may seem to be inapplicable on the other. For example, the author works with a data structure he calls the *global tixel matching* [4]. It keeps track of whether an instance's resources are sufficient to satisfy the demand for resources, both overall and at specific times. is useful when assigning teachers and rooms in high school timetabling [4], so it might be useful for students too—but not if student assignment is genuinely different.

Another negative effect is that solvers assign students in one way, and teachers and rooms in another, encouraging these tasks to occur in separate phases of the solver. The literature is divided on whether to assign students to sections first [1], or to assign times first and then timetable each student.

J. Kingston
School of Information Technologies, The University of Sydney, Australia
E-mail: jeff@it.usyd.edu.au

This paper breaks down the barrier by transforming student sectioning into conventional timetabling. The author knows of no previous publication of this transformation, but it is not likely to surprise researchers familiar with student sectioning. This paper's novelty lies more in its uncovering of the *utility* of the transformation: it allows good ideas from conventional timetabling to be applied to student sectioning, and it permits the whole timetable to be built in an integrated way, bypassing the problems caused by separate phases.

2 The transformation

This section presents the transformation of student sectioning into ordinary timetabling via an example. It will be clear that it works in general. A few simple extensions are also presented. The transformation would be carried out by software when preparing for solving, not by the end user.

A university has two Chemistry laboratories (rooms) for senior students. A laboratory event occupies four hours, and may start at 9am or 2pm each week day. This makes a total of 20 events per week (10 per week in each of the two laboratories). Each laboratory can hold up to 20 students, giving capacity for 400 students. In fact, the university has decided to have only 17 events, the minimum required to accommodate the 325 currently enrolled students.

A conventional representation consists entirely of 17 events of the form

$$C_j = \langle 4\ T92, 1\ SenChemLab, 1\ SenChemTeacher, \leq 20\ SenChemStudent\rangle$$

The first item of the tuple is a demand for 4 consecutive hours beginning at one of the elements of set $T92$, defined elsewhere to be the 9am and 2pm times; the second is a demand for one laboratory from the set $SenChemLab$ of senior Chemistry laboratories (this set has two elements); and the third demands a senior Chemistry teacher in the same way. The solver is required to assign an appropriate starting time, lab, and teacher to this event. Various constraints must be obeyed: the resources must have no clashes, be employed only when available, and so on. These other constraints are not our focus here.

The fourth item allows up to 20 students from the set $SenChemStudent$ to attend. The '$\leq$' sign marks the difference from the other requests: it is a defect if any of those are not met, but 20 students is just the maximum. Furthermore, the same room or teacher may attend several of these events at different times; but each student should attend exactly one. Clearly, the student assignments are anomalous, and this is what motivates this paper.

The idea of the transformation is to create an event for each student s_i in $SenChemStudent$ in which s_i meets with a *seat* in a laboratory. A seat, quite concretely, is a piece of furniture capable of holding only one student at any one time. Each laboratory has 20 seats, making 40 seats altogether, and these are the elements of set $SenChemSeats$. The event for student s_i is

$$S_i = \langle 4\ T92, 1\ SenChemSeat, s_i\rangle$$

where s_i indicates a preassignment of s_i to this event. Preassignments are standard in timetabling. If times and seats are assigned to these 325 events, and no seat has a timetable clash, then no laboratory will be overfull.

The original events C_j are retained in modified form:

$$E_j = \langle 4\ T92, 1\ SenChemLab, 1\ SenChemTeacher \rangle$$

Assignments of students and teachers might occur in all 20 possible events, whereas the university has decided to have only 17. So three events have to be excluded, and this is done by adding three events of the form

$$R_k = \langle 4\ T92, 1\ SenChemLab, 20\ SenChemSeat \rangle$$

At the times occupied by these events, one fewer laboratory and 20 fewer seats will be available. There is nothing to force the selected seats to come from the selected laboratory, but if they don't, a simple reassignment after solving ends fixes that problem. (A specific laboratory and its 20 seats could be preassigned to one R_k, but it is not clear whether to preassign them to several R_k.)

Several extensions can be added. If students work in preassigned pairs, then each student event could request two seats and contain two students. The seats may not be adjacent in the same laboratory, but that is easily fixed during the seat reassignment after solving ends. At the author's university, senior Chemistry students attend two laboratory events each week. This can be handled by creating two S_i events per student.

There is a problem when there is no natural low limit to the number of events that may occur simultaneously, as is imposed by having only two laboratories in the example. Take a junior Mathematics course broken into 20 sections of 30 students each, held in ordinary rooms which are in plentiful supply. A pure transformation would require 600 seats, the great majority of which are unavailable. In practice, however, it would be safe (and perhaps desirable) to arbitrarily limit the number of events that occur simultaneously, coming close to the Chemistry example. It may be important to do this, to ensure that the transformed instance is not too much larger than the original.

Additional constraints, such as that each event have at least some number of students, or that students be distributed among the events as evenly as possible, must be imposed separately, in both the original instance and the transformed one. Representing them in the transformed instance is a potential problem, because the transformation fragments one event into many.

3 Solving

Conventional solvers first construct a solution and then repair it. There may be a first phase which assigns times, and a second which assigns resources. What would such a solver make of a transformed student sectioning instance?

If, after every event is assigned a time, the global tixel matching reports that sufficient resources are available at all times, then resource assignment for the transformed events is trivial and must succeed (except for unpreassigned teachers, who may have constraints which make assignment difficult, with or without the transformation). So the student sectioning part of the problem is basically about constructing and repairing a time assignment which minimizes the 'insufficient resources' defects reported by the global tixel matching.

If it is considered expedient to group students into initial sections before solving [1], then this is *hierarchical timetable construction* [3]: placing the S_i into groups and constraining the elements of each group to be simultaneous.

Initial construction of a time assignment, with the global tixel matching as a guide, should find a timetable which makes a reasonable starting point for repair. Swapping a C_j with an R_k moves a section to a different time; moving or swapping an S_i moves a student to a different section. Both seem useful. There are also some natural larger neighbourhoods, the kind used by VLSN search [6]. An example is unassigning one student's events, then reassigning them using a tree search with intelligent backtracking. This has been done at the author's university for many years, but never published.

All this could go on at the same time as repairs are tried which improve other aspects of the timetable, producing an integrated solve rather than a series of separate phases. Doing it without the transformation is possible but much clumsier, since special arrangements would then have to be made to compute the equivalent of the current availability of seats at each time.

References

1. Michael W. Carter, A comprehensive course timetabling and student scheduling system at the University of Waterloo, Practice and Theory of Automated Timetabling III (Third International Conference, PATAT2000, Konstanz, Germany, August 2000, Selected Papers), Springer Lecture Notes in Computer Science 2079, 64–81 (2001)
2. Peter de Haan, Ronald Landman, Gerhard Post, and Henri Ruizenaar, A case study for timetabling in a Dutch secondary school, Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Czech Republic, August 2006, Selected Papers), Springer Lecture Notes in Computer Science 3867, 267–279 (2007)
3. Jeffrey H. Kingston, 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 (2007)
4. Jeffrey H. Kingston, Resource assignment in high school timetabling, Annals of Operations Research, 194, 241–254 (2012)
5. Simon Kristiansen and Thomas R. Stidsen, Adaptive large neighborhood search for student sectioning at Danish high schools, PATAT 2012: Ninth international conference on the Practice and Theory of Automated Timetabling, Son, Norway (2012)
6. Carol Meyers and James B. Orlin, Very large-scale neighbourhood search techniques in timetabling problems, 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, 24–39 (2007)
7. Keith Murray, Tomáš Müller, and Hana Rudová, Modeling and solution of a complex university course timetabling problem, 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, 189–209 (2007)

Large-Scale Rostering in the Airport Industry

Andreas Klinkert

We present a major research and business project aimed at developing efficient and flexible software for automated airport staff rostering. Industrial partner is Swissport International, one of the largest ground handling companies worldwide, and pilot site for the project is Zurich Airport in Switzerland. Swissport provides services for 224 million passengers and 4 million tons of cargo a year, with a workforce of 55,000 personnel at 255 airports. Airport ground handling involves a broad range of different tasks, including passenger services like check-in, gate handling and transfer services, and ramp services like baggage management and aircraft handling, servicing and cleaning.

The diversity of the ground handling functions at Zurich Airport, the large number of operational duties, and the around-the-clock business hours result in hundreds of different types of shifts to be planned every month, and an employee base consisting of several thousand persons with numerous different skills. Further challenges come from a dynamic, demand-driven planning policy which does not rely on repetitive shift patterns rolled out over a long-term horizon, and from a so-called shift-bidding approach which attributes high importance to employee preferences regarding the individual work plans.

We start with an introduction to the business environment of the project, and show its actual planning context which comprises other software tools and human planning activities related to the workforce scheduling process. We discuss the various project requirements and the challenges and goals that shaped the project and the methods used.

Employee scheduling typically involves a number of subproblems including demand modeling, shift design, days-off scheduling, and shift assignment. The rostering process considered here focuses on the days-off planning and shift assignment phase.

Zurich University of Applied Sciences (ZHAW)
Institute of Data Analysis and Process Design (IDP)
Rosenstrasse 3, P.O. Box Winterthur CH-8401 Switzerland
E-mail: andreas.klinkert@zhaw.ch

The methodology used for solving the associated complex large-scale optimization problems comprises a broad range of optimization techniques including preprocessing, decomposition and relaxation approaches, large-scale integer programming models and various heuristic procedures.

We provide insight into several aspects of the solution process, with special focus on the analysis and preprocessing phase which turned out to be crucial for the entire planning system. An important purpose of this phase is to deal with feasibility issues related to incorrect or inconsistent input data. In fact, experience shows that most of the operational instances submitted to the planning tool are infeasible, and detecting and patching the infeasibility is typically very difficult. Without specific hints from the software it is virtually impossible for the human planners to discover the causes of infeasibility, and to adjust the input data accordingly. The tools developed for this planning phase range from simple but thorough data checking and analysis modules to sophisticated mathematical models for bottleneck analysis, identification of minimal infeasible constraint systems, and rapid presolving techniques.

Finally, we present computational experience with real world instances and discuss operational impacts of the developed planning tool. The operational deployment started in 2011 in Zurich Airport and has continually been extended since then. Bottom line benefits include faster and more robust planning processes, improved roster quality, better fairness, reduced planning capacity requirements, and as a result, substantial financial savings.

Graphics Processing Unit acceleration of a memetic algorithm for the Examination Timetabling Problem

Vasileios Kolonias · George Goulas · Panayiotis Alefragis · Christos Gogos · Efthymios Housos

Keywords GPU programming · Memetic algorithms · Examination Timetabling

In this paper, we present experimental results from the use of Graphics Processing Units (GPUs) as an accelerator for the solution of the uncapacitated examination timetabling problem (uETP). To test the implementation of the proposed algorithm we use the Toronto datasets as the benchmark set. Details about the uETP problem, the datasets and solution methods can be found in the survey by Qu et al. [1]. GPU implementations of genetic algorithms have performed significant speedups in other application domains [2],[3].

In this work, we use a simple memetic algorithm as the goal of our study was to demonstrate the acceleration possibilities of the GPUs in conjunction with simple algorithmic implementation. We use an array of integers for the direct representation of chromosomes where each cell represents an examination and the corresponding value represents the time slot the examination should be scheduled. For the local search optimization of individual candidate solutions we selected the steepest descend algorithm although it is relative simple to extend the implementation to use more advanced meta-heuristics.

Vasileios Kolonias · George Goulas · Efthymios Housos
University of Patras, Dept. of Electrical and Computer Engineering, Greece
Tel.: +30-2610962436
E-mail: bakoloni@ece.upatras.gr

George Goulas · Panayiotis Alefragis
Technological Educational Institute of Western Greece. Dept. of Computers & Informatics Engineering, Greece

Christos Gogos
Technological Educational Institute of Epirus. Dept. of Accounting and Finance, Greece

As the data transfers to and from the GPU are time consuming, we executed a performance analysis on an initial sequential implementation to determine the most time consuming computational kernels. According to this analysis, the most consuming kernel was the local search improvement of the incumbent solutions. What is interesting is that the kernel execution for each solution is independent and all of them can be performed in parallel by GPU cores. To simplify the implementation, we decided to only delegate the execution of the local search improvement kernel to the GPU and perform the rest of the steps (crossover, mutation and next generation decisions) of the memetic algorithm on the CPU.

The cost function is a weighted sum of the absolute distance in time slots between all examination pairs that have common students. The original weight vector $w = \{-, 16, 8, 4, 2, 1\}$ states that if there is a conflict, i.e. two examinations that have common students are assigned to the same time slot, the solution is illegal and for distances of more than 6 time slots the weight is zero. To achieve an efficient GPU implementation of the cost evaluation we extended the weight vector to $w = \{40000, 16, 8, 4, 2, 1, 0, \ldots, 0\}$, size been the number of the problem's available time slots and dropped conditional statements. This has as a consequence that every absolute distance has a weight so all GPU cores can perform the same operation. This is very important for GPUs as the existence of an "if" statement leads to vast degradation in GPU performance.

Another important pre-processing step was the calculation of the diagonal conflict matrix, where for every examination i, each row R_i contains only conflicts with examinations with indexes $j \geq i+1$. This helped reduce the memory requirements and improved the access complexity during the evaluation phase. The characteristics of the memetic algorithm used are:

- Tournament selection with size 6% of the population size.
- Uniform crossover operator.
- In the mutation operator two random examinations swap their time slots .
- The steepest descent algorithm for local improvement of the incumbent solution. All chromosome genes are traversed in parallel to find the best time slot for every examination. The process of finding the best time slot for each examination is not independent because the cost of each examination should be calculated with the time slots of all the other examinations fixed. Each thread in a block calculates the cost of an examination for each slot and the exam is moved to the time slot with the smallest cost. This process is repeated for all examinations.

Table 1 presents the achieved speedup for different population sizes as well as the quality results of this work and the results of another evolutionary technique used to solve the same problem, the informed genetic algorithm (IGA) [4]. As the average performed speedup is between 17 and 40 times faster compared to the CPU, there is great potential in using GPUs to solve the ETP using memetic algorithms. Our future plans include the selection of more elaborate methods for local improvement of the solutions and the implementation

Dataset	CPU(s)	GPU(s)	S	CPU(s)	GPU(s)	S	This work	IGA
car-f-92	0,497	0,021	23.67	63,14	1,287	49.06	4.74	**4.2**
car-s-91	0,786	0,034	23.12	100,69	2,234	45.07	5.36	**4.9**
ear-f-83	0,090	0,004	22.50	11,504	0,241	47.73	37.04	**35.9**
hec-s-92	0,020	0,001	20.00	2,544	0,059	43.12	**10.83**	11.5
kfu-s-93	0,097	0,006	16.17	12,372	0,323	38.30	**14.09**	14.4
lse-f-91	0,068	0,005	13.60	8,693	0,238	36.53	11.18	**10.9**
rye-s-93	0,162	0,009	18.00	20,791	0,525	39.60	**8.88**	9.3
sta-f-83	0,015	0,001	15.00	1,933	0,064	30.20	**157.06**	157.8
tre-s-92	0,111	0,006	18.50	14,159	0,340	41.64	8.69	**8.4**
uta-s-92	0,644	0,028	23.00	82,283	1,867	44.07	3.93	**3.4**
ute-s-92	0,013	0,001	13.00	1,667	0,064	26.05	**25.05**	27.2
yor-f-83	0,077	0,004	19.25	9,881	0,226	43.72	**38.2**	39.3
Average Speedup			18,82			40.43		

Table 1 Achieved speedup (S) for a single generation of the steepest descend algorithm and quality results

of all the memetic algorithm phases in the GPU. Furthermore, we would like to solve the datasets of the International Timetabling Competition (ITC) with the proposed method and the use of GPUs.

References

1. Qu, R. and Burke, E.K. and McCollum, B. and Merlot, L.T.G. and Lee, S.Y., A survey of search methodologies and automated system development for examination timetabling, Journal of Scheduling, 12, 55-89 (2009)
2. Pospichal, P. and Jaros, J. and Schwarz, J., Parallel genetic algorithm on the CUDA architecture, Proceedings of the 2010 international conference on Applications of Evolutionary Computation - Volume Part I, 442-451. Springer-Verlag, Istanbul, Turkey (2010)
3. Arora, R. and Tulshyan, R. and Deb, K., Parallelization of binary and real-coded genetic algorithms on GPU using CUDA, 2010 IEEE Congress on Evolutionary Computation (CEC), 1-8. Barcelona (2010)
4. Pillay, N.; Banzhaf, W., An informed genetic algorithm for the examination timetabling problem. Appl. Soft Comput. 2010, 10, 457-467

Integer Programming for the Generalized (High) School Timetabling Problem

Simon Kristiansen · Matias Sørensen · Thomas R. Stidsen

1 Introduction

The *High School Timetabling* (HSTT) is an important problem encountered at the high school in many countries. High School Timetabling is the problem of scheduling lectures to time slots and/or resources at high schools, and a large amount of different solution approaches have been proposed, see surveys Schaerf (1999); Kristiansen and Stidsen (2013)

It is well recognized that the specifications of the HST problem varies significantly depending on the country of which the problem originates, and that the problem in general is hard to solve.

Due to lack of exchangeable benchmark in a uniform format of the HSTT, Post et al (2012) introduced the XHSTT format, consisting of a large number of instances from various origins in standardized form. The format is based on the *Extensible Markup Language* (XML) standard, and all instances are available online (Post, 2013). One purpose of the format is to serve as a common test-bed for high school timetabling, in an attempt to promote research within this area.

In this paper, we will describe the first mixed *Mixed Integer linear Programming* (MIP) method capable of handling an arbitrary instance of the XHSTT format. The solution method is a two steps approach using a commercial general-purpose MIP solver. Computational results are performed for all the real-life instances currently available. By applying this MIP method, we are able to find previously unknown optimal solution, and prove optimality of already known solutions.

To the best of our knowledge, all previous methods applied for the XHSTT format have been heuristic in nature. Hence, no proof of optimality has been made for any instances, except for those instances where a solution with objective value 0 is known, as 0 is a trivial lower bound. Moreover, the quality of a given solution is rather hard to measure as only the trivial lower bound exist. An obvious advantage of *Integer Programming* (IP) is the capability to issue certificates of optimality. Therefore it is remarked that a big advance within general-purpose MIP solvers has happened in recent years, see e.g. Bixby (2012). Even though the created MIP is inevitable complex in nature, it will be shown that it can be used to find optimal solutions for several instances of the XHSTT archive `ALL_INSTANCES`. For those instances where an optimal solution cannot be found, we are able to show a non-trivial lower bound on optimum in the majority of cases. These are significant results for high school timetabling in general.

2 Mixed Integer Programming Formulation

In this section a brief description of the different terms of the MIP model is given.

S. Kristiansen (E-mail: sikr@dtu.dk) · M. Sørensen
MaCom A/S, Vesterbrogade 48, 1., DK-1620 Copenhagen V, Denmark

T.R. Stidsen
Section of Operations Research, Department of Management Engineering, Technical University of Denmark
Building 426, Produktionstorvet, DK-2800 Kgs. Lyngby, Denmark

An instance of XHSTT consists of *times* (denoted $\mathcal{T}$ in the following), *time groups* (denoted $\mathcal{TG}$), *resources* (denoted $\mathcal{R}$), *events* (denoted $\mathcal{E}$), *event groups* (denoted $\mathcal{EG}$) and *constraints* (denoted $\mathcal{C}$). An event $e \in \mathcal{E}$ has a duration $D_e \in \mathbb{N}$, and a number of *event resources* which we each denote $er \in e$. An event resource defines the requirement of the assignment of a resource to the event, and this resource can be specified to be preassigned. If the resource is not preassigned, a resource of proper *type* must be assigned. Furthermore an event resource er can undertake a specific role_{er}, which is used to link the event resource to certain constraints.

It is the job of any solver for XHSTT to decide how each event should be split into *sub-events*. A sub-event se is defined as a fragment of a specific event $e \in \mathcal{E}$, has a duration $D_{se} \leq D_e$, and inherits the requirement of resources defined by the event, such that each sub-event has the exact same resource requirements as the event. Let $\mathcal{SE}$ denote the entire set of sub-events, and let $se \in e$ specify that sub-event se is part of event e. The total duration of all sub-events for event $e \in \mathcal{E}$ in a solution cannot exceed D_e. In our model formulation we create the 'full set' of sub-events with different lengths, i.e. all possible combinations of sub-events for a given event can be handled. E.g. if an event has duration 4, the set of sub-events for this event has the respective lengths $1, 1, 1, 1, 2, 2, 3$ and 4. As a constraint it is then specified that the summed duration of the *active* sub-events in a solution must equals 4. A sub-event is active if it is assigned a starting time or a non-preassigned resource. An active sub-event is analogous to the concept of *solution events* defined in the XHSTT documentation.

The times $\mathcal{T}$ are ordered in chronological order, and we let $\rho(t)$ denote the index number of time t in $\mathcal{T}$. A time group $\mathcal{TG}$ defines a set of times, and we let $t \in tg$ denote that time t is part of time group tg.

Each constraint $c \in \mathcal{C}$ is of a specific type, and the set $\mathcal{C}$ can contain several constraints of the same type. Each constraint applies to certain events, event groups or resources, and penalizes certain characteristics of the timetable for these entities.

More details on the MIP model will be given in a full paper on the research.

3 Results

We have two primary intentions with our computational results:

- How does the MIP compete with the heuristics of the ITC2011 round 2? Thereby the potential of this MIP approach can be evaluated on fair terms with well-performing heuristics.
- Are we able to improve the best-known solutions for some instances, or even solve them to optimality?

All tests were run on a machine with an Intel i7 CPU clocked at 2.80 GHz and 12GB of RAM, running Windows 8 64 bit. In all cases the commercial state-of-art MIP solver Gurobi 5.5.0 was used. Two distinct sets of XHSTT instances have been used, both obtained from the XHSTT website (Post, 2013). All obtained solutions have been verified as being valid using the evaluator *HSEval* (Kingston, 2013).

An XHSTT objective consists of both a hard cost, and a soft cost, denoted (hard cost, soft cost). In case a solution has a hard cost of value 0, the objective is simply written as the soft cost, as is usually done in context of the XHSTT format. By this definition, hard constraints always take priority over soft constraint. Hence, we use a *lexicographic multi-objective optimization* techinique, where we first solve problem containing only the hard constraints. In case of optimality the soft constraints are added, and a constraint is added to ensure the hard cost optimality.

Table 1 shows the obtained results of the comparison with the competitors of ITC2011 The value of "Avg. Ranking" was calculated as follows. Each solution method was ranked 1 to 5 on each instance, 1 being the best, and the average of these ranks was taken. According to this measure, the exact method of this paper is competitive with the methods used at ITC2011. On two instances the exact method gave the best results.

In attempt to produce new (optimal) solutions, the XHSTT archive `ALL_INSTANCES`[1] was used, which contains 38 non-artificial instances. According to the website, this archive "contains all latest versions of the contributed instances". The instances with a solution cost of 0 are skipped. Gurobu

[1] The archive and the results compared with are dated to 5th September 2013

was allowed to use all CPU cores (8 in our case) and a time-limit of 24 hours. Table 2 shows the obtained results. We were able to solve 4 instances was solved to optimality, proving optimality of 3 previously known solutions and finding 1 new optimal solution. Furthermore, 11 new non-trivial lower bounds and 7 new best solutions have been established for the instances which were not solved to optimality.

Table 1: *Performance of the MIP using same running time as specified in ITC2011. For each instance is listed the average solution found from each of the competitors of ITC2011, and the solution obtained by the MIP formulations. The best solutions are marked in* ***bold.***

Instance	GOAL	HySST	Lectio	HFT	Exact method
BrazilInstance2	(1, 62)	(1, 77)	**38**	(6, 190)	46
BrazilInstance3	124	118	152	(30, 283)	**39**
BrazilInstance4	(17, 98)	(4, 231)	**(2, 199)**	(67, 237)	(5, 286)
BrazilInstance6	(4, 227)	(3, 269)	**230**	(23, 390)	682
ElementarySchool	4	(1, 4)	**3**	(30, 73)	**3**
SecondarySchool2	**1**	23	34	(31, 1628)	(1604, 3878)
Aigio	**13**	(2, 470)	1062	(50, 3165)	(1074, 3573)
Italy_Instance4	**454**	6926	651	(263, 6379)	17842
KosovaInstance1	**(59, 9864)**	(1103, 14890)	(275, 7141)	(989, 39670)	(3626, 2620)
Kottenpark2003	**90928**	(1, 56462)	(50, 69773)	(209, 84115)	(8491, 6920)
Kottenpark2005A	**(31, 32108)**	(32, 30445)	(350, 91566)	(403, 46373)	(2567, 53)
Kottenpark2008	**(13, 33111)**	(141, 89350)	(209, 98663)	-	(14727, 5492)
Kottenpark2009	**(28, 12032)**	(38, 93269)	(128, 93634)	(345, 99999)	(17512, 140)
Woodlands2009	(2, 14)	(2, 70)	**(1, 107)**	(62, 338)	(1801, 705)
Spanish school	**894**	1668	2720	(65, 13653)	(1454, 11020)
WesternGreece3	**6**	11	(30, 2)	(15, 190)	25
WesternGreece4	**7**	21	(36, 95)	(237, 281)	81
WesternGreece5	**0**	4	(4, 19)	(11, 158)	15
Avg. Ranking	1.72	2.67	2.50	4.44	3.61

4 Conclusion

Establishing optimal solutions and lower bounds is indeed a step forward for research within high school timetabling, and for the XHSTT format in particular. This gives researchers a possibility to compare their obtained solutions with an (optimal) lower bound, which is valuable for evaluating the quality of solutions.

References

Bixby RE (2012) Optimization Stories, 21st International Symposium on Mathematical Programming Berlin, vol Extra, Journal der Deutschen Mathematiker-Vereinigung, chap A Brief History of Linear and Mixed-Integer Programming Computation, pp 107–121

Kingston JH (2013) The hseval high school timetable evaluator. http://sydney.edu.au/engineering/it/ jeff/hseval.cgi [Retrieved 12/8-2013]

Kristiansen S, Stidsen TR (2013) A comprehensive study of educational timetabling - a survey. Tech. Rep. 8, 2013, DTU Management Engineering, Technical University of Denmark

Post G (2013) Benchmarking project for (high) school timetabling. http://www.utwente.nl/ctit/hstt/ [Retrieved 12/8-2013]

Post G, Ahmadi S, Daskalaki S, Kingston J, Kyngas J, Nurmi C, Ranson D (2012) An xml format for benchmarks in high school timetabling. Annals of Operations Research 194:385–397

Schaerf A (1999) A survey of automated timetabling. Artificial Intelligence Review 13:87–127

Table 2: *Performance of the MIP on* ***ALL_INSTANCES****. For each instance is listed the best previously known solution* "Best"*, and for the solution found by our approach is listed the time used to solve Step 1* "Time_1"*, the time used to solve Step 2* "Time_2"*.* "Time" *indicates the total solving time. All times have seconds as unit. Furthermore the objective* "Obj" *and the lower bound* "LB" *is listed. The percentage gap between the objective and the lower bound is divided into the gap for the hard constraints* "Gap_1" *and the gap for the soft constraints,* "Gap_2"*. Objectives in* ***bold*** *denote new best solution while optimal solutions are marked with* **.*

			MIP solution method						
	Instance	Best	Time_1	Time_2	Time	Obj	LB	Gap_1	Gap_2
AU	BGHS98	(3, 494)	>86400	-	>86400	(3, 494)	(-,-)	-	-
AU	SAHS96	(8, 52)	>86400	-	>86400	(8, 52)	(-,-)	-	-
AU	TES99	(1, 140)	>86400	-	>86400	(1, 140)	(0,-)	100.0	-
BR	Instance1	42	0	>86400	>86400	**40**	28	0.0	30.0
BR	Instance2	5	1	>86399	>86400	5	1	0.0	80.0
BR	Instance3	47	1	>86399	>86400	**26**	19	0.0	26.9
BR	Instance4	78	1	>86399	>86400	**61**	42	0.0	31.2
BR	Instance5	43	1	>86399	>86400	**30**	10	0.0	66.7
BR	Instance6	60	1	>86399	>86400	60	14	0.0	76.7
BR	Instance7	122	1	>86399	>86400	122	22	0.0	82.0
DK	Falkoner2012	(2, 23705)	>86400	-	>86400	(2, 23705)	(0,-)	100.0	-
DK	Hasseris2012	(293, 32111)	>86400	-	>86400	(293, 32111)	(-,-)	-	-
DK	Vejen2009	(20, 18966)	>86400	-	>86400	(20, 18966)	(2,-)	90.0	-
UK	StPoul	136	52	>86348	>86400	136	0	0.0	100.0
FI	ElementarySchool	3	2	785	787	*3	3	0.0	0.0
FI	HighSchool	1	1	>86399	>86400	1	0	0.0	100.0
FI	SecondarySchool	88	1	>86399	>86400	88	77	0.0	12.5
GR	UniInstance3	5	0	3	3	*5	5	0.0	0.0
GR	UniInstance4	8	1	>86399	>86400	8	0	0.0	100.0
IT	Instance1	12	1	4561	4562	*12	12	0.0	0.0
IT	Instance4	78	12	>86389	>86400	**62**	27	0.0	56.5
XK	Instance1	3	31	>86369	>86400	3	0	0.0	100.0
NL	GEPRO	(1, 566)	>86400	-	>86400	(1, 566)	(0,-)	100.0	-
NL	Kottenpark2003	1410	57	>86343	>86400	1410	(0,-)	0.0	-
NL	Kottenpark2005	1078	88	>86312	>86400	1078	9	0.0	99.2
NL	Kottenpark2009	9250	92	>86308	>86400	**9035**	160	0.0	98.2
ZA	Woodlands2009	2	22	77878	77900	***0**	0	0.0	0.0
ES	School	(3, 5966)	6525	>79875	>86400	**357**	322	0.0	9.8

The impact of cyclic versus non-cyclic scheduling on the project staffing cost

Broos Maenhout · Mario Vanhoucke

1 Introduction

In project scheduling assumptions are made with respect to the availability of resources. In case resources are scarce and/or costly, the scheduling of resources becomes increasingly important. Despite the fact that projects are typically very labour intensive, little attention in the literature is given to the underlying personnel scheduling problem. In the project staffing problem under study we integrate the personnel scheduling problem and the project planning problem. This encompasses that we have to decide on the project schedule that leads to the optimal staffing plan by simultaneously determining the starting times of the activities, the project duration and the best mix of resources in terms of cost. In order to come up with a staffing plan, a baseline schedule is composed for the regular and temporary crew members that takes into account all the scheduling policies and practices.
In this paper we give insight in the parameters of the integrated staffing problem with personnel calendar constraints. More precisely, we investigate the conditions (i.e. problem characteristics and parameter settings) under which a personnel schedule can be constructed in a cyclical or non-cyclical way. In this way, the project planner will learn about the impact of specific problem characteristics and policies on the integrated project staffing outcome.

Broos Maenhout
Faculty of Economics and Business Administration, Ghent University, Belgium
E-mail: Broos.Maenhout@Ugent.be

Mario Vanhoucke
Faculty of Economics and Business Administration, Ghent University, Belgium
Vlerick Business School, Belgium
University College London, United Kingdom
E-mail: Mario.Vanhoucke@Ugent.be

2 Problem description

Project planning and personnel scheduling are two different problems that are separately studied in the literature in various guises and formulations ([6]; [10]. The relevant literature on the integrated project staffing and personnel scheduling problem is on the other hand limited. The problem was introduced by [1] and [2]. In that papers the authors proposed a mathematical problem formulation for composing a cyclic personnel schedule. The objective is to minimize the project makespan and the personnel costs. Maenhout and Vanhoucke [7] extended their problem definition by incorporating temporary personnel members and allowing the regular workers to work overtime. In that paper, a dedicated solution procedure is used to come up with non-cyclical personnel schedules. Other papers on project staffing do not construct a personnel schedule explicitly. [3] discuss an exact optimization method for a multi-skill project scheduling problem. The problem is then to schedule the project activities given the personnel schedule such that the scheduled personnel is able to carry out the different project activities. [5] presents a model for the simultaneous scheduling and staffing of multiple projects with multiple resources. Several other related papers involve the audit scheduling problem where tasks have to be assigned to a set of workers that have an overall capacity for the entire planning horizon (cfr. [4] for an overview).

In this paper, we study a strategic budgeting problem that decides on the project staffing plan and corresponding personnel schedule for a single project. The activity and project scheduling characteristics are defined as follows. There is a project network that consists of a set of activities and a set of direct precedence relations. The pre-emption of activities is not allowed. Each activity has a deterministic duration and requires a number of resource units per time unit. We assume that there is a prescribed maximum project duration. A project schedule is said to be feasible if it is non-preemptive and if the project duration, precedence and resource constraints are satisfied. The activities are executed by different renewable personnel resource types, i.e. regular personnel time units, overtime units and temporal personnel time units. The availability of these resources, and hence, the resource constraints for each time unit of the project horizon are determined in the personnel scheduling step. The personnel scheduling problem is a manpower day-off scheduling problem where the total number of regular crew members, the budget for overtime and temporary help are determined. All personnel members are indistinguishable, as they all possess the same skills to carry out the activities. The schedule of the regular crew members can be conceived *cyclically*, where the same periodic pattern of days on and days off is repeated for each crew member (e.g. a (5,2) pattern that stipulates that the personnel has to work five days followed by two days off)or *non-cyclically*, where an 'ad hoc' schedule is constructed for each crew member and each period. The objective of this strategic budgeting problem is to minimize the overall staffing cost and activity execution cost.

3 Computational experiments

In this research we investigate the effect of the problem characteristics on the staffing budget and the computational performance. The computational experiments are performed using the branch-and-price procedure of [7]. We did our experiments on a randomly generated dataset under a controlled design consisting up to 30 activities. Based on these project instances, we determined the optimal staffing budget for each project duration between the critical path length and the project deadline. The critical path length averages 30 days and in the optimal crew rosters there are 16 regular workers on the average.

We explore the impact of different project and personnel staffing characteristics on the staffing budget and on the possibility to compose a cyclical and/or non-cyclical personnel schedule. More precisely, for the activity and project characteristics we analyse the impact of the serial and parallel complexity indicator and the activity distribution indicator of [8, 9], the impact of release and due times and the impact of the resource profile of the activities. Last, we examine the non-linear relationship of the staffing cost as a function of the project duration. The number of regular personnel members shows an inverse relationship with the planning period, i.e. as the planning period is increased, the number of regular personnel members decreases. On the other hand, the required CPU time shows an increasing exponential behaviour when the planning period is extended and the degree of freedom for scheduling the activities increases.

References

1. Alfares, H. and Bailey, J., Integrated project task and manpower scheduling, IIE Transactions, 29, 711-717 (1997)
2. Alfares, H., Bailey, J. and Lin, W., Integrated project operations and personnel scheduling with multiple labour classes, Production Planning and Control, 10, 570-578 (1999)
3. Bellenguez-Morineau, O., Methods to solve multi-skill project scheduling problem, 4OR, 6, 85-88 (2008)
4. Dodin, B. and Elimann, A., Audit scheduling with overlapping activities and sequence-dependent setup times, European Journal of Operational Research, 97, 22-33 (1997)
5. Heimerl, C. and Kolisch, R., Scheduling and staffing multiple projects with a multi-skilled workforce, OR Spectrum, 32, 343-368 (2010)
6. Herroelen, W., Demeulemeester, E., and De Reyck, B.,. Project scheduling - recent models, algorithms and applications, chapter A classification scheme for project scheduling problem, pages 1-26. Kluwer Academic Publishers 1, Boston (1999)
7. Maenhout, B., and Vanhoucke, M., An integrated approach to strategic personnel budgeting for a single project with a homogeneous workforce, Technical report, Ghent University (2013)
8. Vanhoucke, M., and Vandevoorde, S., A simulation and evaluation of earned value metrics to forecast the project duration, Journal of the Operational Research Society, 58, 1361-1374 (2007)
9. Vanhoucke, M., Measuring Time - Improving Project Performance using Earned Value Management, International Series in Operations Research and Management Science, volume 136, Springer (2010)
10. Van den Bergh, J., Belien, J., Be Bruecker, P., Demeulemeester, E. and De Boeck, L., Personnel scheduling: A literature review, European Journal of Operational Research, 226, 367-385 (2013)

Decomposition and Recomposition Strategies to Solve Timetabling Problems

**Dulce J. Magaña-Lozano · Ender Özcan ·
Santiago E. Conant-Pablos**

1 Introduction

Timetabling is a crucial and often extremely time consuming task in many educational institutions. This task is generally performed periodically (each year, semester, quarter) for fulfilling the requirements imposed by the institution and people involved, such as students and teachers/lecturers making efficient and effective use of the available resources. The educational timetabling problem has been widely studied and different classifications have been proposed (see [1], [5]). The course timetabling problem (CTTP) is a combinatorial optimisation problem which involves assignment of a given set of meetings along with available resources to appropriate time slots subject to a set of constraints. In general, two types of constraints can be identified: *hard* and *soft*. The hard constraints are those that *must be* satisfied under any circumstances. Timetables that do not violate hard constraints are called *feasible*. On the other hand, soft constraints are those that *need to be* respected as many as possible, but can still be violated if necessary, i.e. they are desirable but not essential. These constraints are frequently used to evaluate how good the solutions (timetables) are.

Two classes of well known solution methods in timetabling are construction and decomposition methods (e.g. [2], [3], [6]). In this study, we investigate an approach which decomposes a given problem into smaller subsets and then sequentially constructs a partial solution using the subsets recomposing them into a complete solution. The proposed approach is tested on the Post Enrolment based Course Timetabling problem instances of the Track 2 from the second International Timetabling Competition (ITC2007) for solving the hard constraints. The

Dulce J. Magaña-Lozano · Ender Özcan
ASAP Research Group, University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{ psxdjm, ender.ozcan }@nottingham.ac.uk

Santiago E. Conant-Pablos
Tecnológico de Monterrey
Av. E. Garza Sada 2501, Monterrey, NL, 64849, Mexico
sconant@itesm.mx

main characteristic of this track is that the timetable is produced after student enrolment on courses has taken place.

2 Experiments and Results

In this study, we formulate the post enrollment based course timetabling problem as a Constraint Satisfaction Problem (CSP). Firstly, all meetings are ordered using a heuristic. In our experiments, we used *Largest Degree* heuristic, which sorts the meetings in decreasing order by the number of conflicts with other meetings. After decomposing a given set of meetings into fixed size subsets, we construct a partial and independent solution using each subset. At this stage, each subset is also in order. The meetings in the first subset will be the events with the largest degree, and in the last subset, the meetings with the smallest degree. Then the subsets are recomposed sequentially towards a complete solution with a certain strategy embedding a conflict resolution heuristic, since merging partial solutions into a larger partial solution could cause hard constraint violations.

Each subset represents a subproblem which are solved by a backtracking algorithm with forward checking. This algorithm dynamically selects the next variable to assign using the *minimum remaining values* heuristic which uses the variable with the smallest domain. To break ties, the *saturation degree* heuristic is used, which chooses the variable with the maximum number of constraints over the unassigned variables. Moreover, the *least constraining value* heuristic is employed to assign a value, which reduces the size of the domain of unassigned variables at the least, to the chosen variable. Ties are broken randomly.

An *incremental* recomposition strategy, similar to the one proposed in [3], is utilised. Without changing the initial ordering of subsets and using the first subset as the initial partial solution, the next subset is incrementally added into the partial solution in hand until all subsets are covered. Each time a subset is included in the growing partial solution, conflict resolution algorithm is invoked. An important feature of the recomposition strategy is that it should be able to reduce the conflicts between the variables that belong to different subsets. Hence, it embeds the *Min-Conflict* local search algorithm as a conflict resolution method while integrating the subsets. This algorithm produces a list of variables in conflict and randomly chooses one of them to assign a different value which generates the minimum number of conflicts with the other variables, namely, the one that minimizes the number of unsatisfied hard constraints. These steps for conflict resolution are repeated for a fixed number of times (attempts) in order to gradually reduce the number of conflicts between the variables, and eventually find a feasible solution.

It is possible that all the events can not be scheduled in the given time without breaking some hard constraints, thus some events in the timetable will not be placed in order to ensure that no hard constraints are being violated. If there are unplaced events, the *Distance to Feasibility* (DtF) measure is calculated [4] as proposed in ITC2007. This measure represents the total number of students that attend to each of the unplaced events, 0 means that all events were scheduled without violations to the hard constraints. The DtF is used to measure the performance of the algorithms. The Equation 1 shows how to calculate the DtF measure where e_i represents the event i, s_{e_i} the number of students that attends to the event i and, n the number of events.

$$DtF = \sum_{i=0}^{n} s_{e_i} [e_i \text{ is unplaced}] \tag{1}$$

We have fixed the number of subsets for a given problem instance. An initial set of exhaustive parameter tuning experiments were conducted to determine the ideal number of subsets. The setting of $\sqrt{Number\_Events}$ for the number of subsets yielded the best performance over multiple runs across the instances with respect to DtF. We repeated each experiment fixing the number of subsets to this value thirty times for each instance. Table 1 shows the best and the average DtF over all runs, as well as, the associated standard deviation for each instance using the Largest Degree decomposition heuristic and the Incremental recomposition heuristic. In all problem instances, except five of them, namely; comp-2, comp-9, comp-10, comp-21 and comp-22, the proposed approach is capable of obtaining feasible solutions. The approach always achieves a feasible solution across all runs for the following five instances: comp-4, comp-8, comp-11, comp-15 and comp-17.

Table 1 Results of one of the proposed algorithm (Largest Degree heuristic with Incremental Recomposition), where *best*, *avr.* and *st.d.* denotes the best, average and standard deviation of the DtF. For each instance, the bold entry marks the best performing approach; the comparison criteria is the average number of DtF.

Instance	best	avr.	st.d.	Instance	best	avr.	st.d.
comp-1	0.0	701.6	447.1	comp-13	0.0	66.0	69.7
comp-2	467.0	1384.7	401.8	comp-14	0.0	114.4	68.3
comp-3	0.0	5.0	27.6	comp-15	0.0	0.0	0.0
comp-4	0.0	0.0	0.0	comp-16	0.0	2.3	12.6
comp-5	0.0	46.2	37.0	comp-17	0.0	0.0	0.0
comp-6	0.0	10.9	20.2	comp-18	0.0	3.5	19.0
comp-7	0.0	4.3	13.0	comp-19	0.0	799.9	479.8
comp-8	0.0	0.0	0.0	comp-20	0.0	4.9	18.5
comp-9	152.0	1152.3	482.7	comp-21	29.0	448.9	266.0
comp-10	816.0	1473.7	376.1	comp-22	3249.0	4759.6	491.6
comp-11	0.0	0.0	0.0	comp-23	0.0	1117.3	638.2
comp-12	0.0	14.6	44.5	comp-24	0.0	396.4	354.4

References

1. Victor A. Bardadym. Computer-aided school and university timetabling: The new wave. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 22–45. Springer Berlin Heidelberg, 1996.
2. M. W. Carter. A decomposition algorithm for practical timetabling problems. Technical paper, Department of Industrial Engineering, University of Toronto, 1983.
3. Ender Özcan, Andrew J. Parkes, and A. Alkan. The interleaved constructive memetic algorithm and its application to timetabling. *Computers and Operations Research*, 39(10):2310–2322, 2012.
4. Ben Paechter. International timetabling competition. `http://www.cs.qub.ac.uk/eventmap/postenrolcourse/course_post_index_files/evaluation.htm`, 2007.
5. A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.
6. Petr Šlechta. Decomposition and parallelization of multi-resource timetabling problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3616 LNCS:177 – 189, 2005.

An Investigation Into the Use of Haskell for Dynamic Programming

David McGillicuddy · Andrew J. Parkes · Henrik Nilsson

Abstract This paper investigates the potential benefits offered by adopting a declarative approach, as embodied by modern functional languages with mature implementations, for prototyping algorithms for solving combinatorial optimisation problems. For this class of problems there are usually many different options for the core algorithms, supporting data structures and other implementation aspects. Thus tools that allow different alternatives to be tried out quickly, focusing on the essence of the problem, and as unencumbered as possible by implementation detail, would be very useful. As a case study, we consider dynamic programming algorithms. These have many uses in scheduling and timetabling: either directly, or as a component within other methods such as column generation. Our findings suggest that off-the-shelf, leading functional languages can indeed offer a range of compelling advantages in this particular problem domain, while yielding a performance that is adequate for verifying and evaluating the implemented algorithms as such.

Keywords Haskell · C · Java · Functional Programming · Dynamic Programming · Language Comparison

1 Introduction

Over the last few decades, the speed of computers has increased by orders of magnitude, but the productivity of programmers has not kept pace. It is often far more important to quickly produce correct and robust code than to optimise code for performance. As computers continue to become more powerful this is ultimately going to become the norm. Prototyping new heuristics and algorithms for combinatorial optimisation is arguably one area where speed of development of correct code is already more important than absolute performance.

D. McGillicuddy, A. J. Parkes and H. Nilsson
School of Computer Science
University Of Nottingham
E-mail: {dxm, ajp, nhn}@cs.nott.ac.uk

We undertook a small case study as a preliminary investigation into whether a declarative approach, specifically functional programming, is feasible for this domain and whether it can help speed up prototyping. Our basic observation is that algorithms and heuristics for combinatorial optimisation at their core have clear mathematical specifications. Implementation, however, is often hampered by the need to spell out a plethora of operational details. This is time-consuming, error prone, and ultimately obscures the essence of the code. Thus if combinatorial optimisation algorithms could be prototyped by, for the most part, transliterating the mathematical specifications, and if the resulting performance were adequate for evaluation purposes, much would be gained already. Additionally because the elementary, "schoolbook" reasoning principle of substituting equals for equals is valid in declaratively formulated code, applying property-based testing (where test cases are derived automatically from stated correctness properties [1]), more easily exploiting multi-core architectures, and formally proving correctness, are potentially facilitated.

For our case study, we have opted to look at a few standard dynamic programming algorithms, specifically *unbounded knapsack* and *longest common substring* (LCS). These have many uses and, for our purposes, are representative of a larger class of algorithms in the domain of combinatorial optimisation. We have opted to use the pure, lazy, functional language Haskell as our declarative implementation framework [4]. Using a pure language increases the contrast to the imperative languages commonly used to implement this class of algorithms, making for a more interesting comparison, while also allowing the specific advantages of working declaratively to be fully realised. Further, Haskell is supported by mature, industrial-strength implementations, resulting in a fairer performance comparison [6].

We would like to emphasise that our aim is not to advocate any particular functional language for prototyping combinatorial optimisation algorithms. Rather, we are interested in exploring what advantages functional notation (supported by mature implementations) can bring today. However, it is worth noting that if these advantages are judged to be compelling enough, functional language implementations can, with relative ease, be leveraged for implementing domain-specific languages (DSL, sometimes referred to as 'executable specifications'). These allow domain-experts interested in working declaratively to reap the benefits of the approach without having to become seasoned functional programmers themselves [3]. One example of such a DSL, used to define and manipulate financial contracts, was produced by Simon Peyton Jones et al. and a derivative of it is used in industry by companies such as Bloomberg and HSBC Private Bank [5].

We carry out the study by implementing each of the chosen algorithms (unbounded knapsack and LCS) in Haskell, Java, and C. The implementations are then compared along a number of dimensions, including conciseness, modularity and performance, as well as ease of debugging, reasoning and parallelising. To make the comparisons meaningful we retain the structure the of the implementations across languages, except where we take advantage of specific language features (such as pointers, objects, or laziness). The implementations are further idiomatic and representative of what a "typical" programmer might write, without non-portable micro-optimisations. In particular, standard libraries are used throughout for data structures and mathematical computations, with as little as possible implemented from scratch.

2 An Illustrative Example

In a recent high profile case [2], a spreadsheet bug caused erroneous results from an economical analysis to be published, possibly influencing European Union policy[1]. The error was caused in part by an indexing mistake that accidentally excluded several countries from the analysis, a clear example of operational details causing problems. As an analogy, consider summing a collection of numbers. In a declarative setting the numbers (whether in the form of an array, list, stream, or otherwise) are simply passed to a generic *sum* function. Indexing and element-wise operations take place behind the scenes, completely eliminating these as possible sources of programmer error. By contrast, in most spreadsheets, the range of cells to be summed are manually selected (e.g., "C3:C100") which can be error-prone. Let us consider how similar ideas might improve a combinatorial optimisation algorithm. Lack of space precludes describing the full results of Knapsack and LCS, however, solving the unbounded knapsack problem involves finding the Greatest Common Divisor (gcd) of the initial capacity and an array $\mathbf{W}$ of n weights. The function *gcd*[2] is associative. Thus to get the gcd of the $n+1$ numbers, first the gcd of the capacity and W_0 is calculated, then the gcd of that number and W_1, and so on for each W_i, reducing to a single integer after n calls. Figure 1 shows the algorithm implemented in Java 7. Iteration over the elements has been abstracted into a *for-each* loop. The accumulator variable *gcd_all* is initialised to *capacity* and then *gcd*'d with each weight, updating the accumulator variable with the result of *gcd* for each $\mathbf{W}_i$. The C version of the algorithm is almost identical, except that the indices and loop ranges have to be written explicitly, adding further operational details. The Haskell version of *gcd* is shown in figure 2. Here the idiom of reducing a list by a binary function and accumulator is captured by the function foldr1 , so called because it folds, associating to the right, over a list with at least one element. There is thus no need for the user to specify how and when the accumulator should be updated. Furthermore, since the definition of *gcd* contains the rule 'gcd 1 _ = 1', which states that $\forall x.gcd(1,x) = 1$, it can be said to be *short-circuiting*; i.e., if the first argument is equal to 1 then, due to lazy evaluation, the second argument is not inspected and is ignored. Therefore *gcds* will automatically stop once a 1 is encountered without any change to the loop itself. Achieving the same optimisation in Java (or C) would require changing the code for the loop itself by fusing it with part of the code *gcd*. This would break modularity, hamper reuse, and possibly render the code less readable. In this very small example, adding a check to see if *gcd_all* is equal to 1 at each iteration and halting the loop if so is a trivial change. However, had the loop or the called function been more involved, the modification would have been correspondingly harder because the code that governs the loop might be quite divorced from the code that updates the accumulating variable.

[1] `www.bbc.co.uk/news/magazine-22223190`

[2] Which takes two strictly positive integers and returns the largest integer that divides them both.

```
public int gcds (int capacity, int[] weights) {
        int gcd_all = capacity;
        for (int weight : weights) {
                gcd_all = gcd (gcd_all, weight);
        }
        return gcd_all;
}
```

Fig. 1: Java 7

```
gcds :: Int -> [Int] -> Int
gcds capacity weights = foldr1 gcd (capacity:weights)
```

Fig. 2: Haskell

3 Results and Conclusions

Our findings so far, to be detailed in the full version of the paper, suggest that functional languages supported by mature implementations can indeed speed up development by allowing implementations to stay close to specifications, taking advantage of specific language features such as laziness, and eliminating certain classes of errors. Furthermore, they can achieve this without incurring a performance penalty that is unacceptable for prototypes. Our benchmark results for unbounded knapsack suggest that the C code is not more than about five times faster than the Haskell version. There are a wide range of languages that provide a transition path to more functional code; first-class functions, folds and pattern-matching have been added to object-oriented languages such as Java, C#, Scala and C++. F# and Clojure can interface seamlessly with C# and Java respectively, and both Haskell and Rust can easily interoperate with C. As such the authors recommend that readers familiarise themselves with these idioms and consider using them in their OR prototypes and implementations.

Acknowledgements This work was funded in part by EPSRC grant EP/F033613/1.

References

1. Claessen, K., Hughes, J.: QuickCheck: A lightweight tool for random testing of Haskell programs. SIGPLAN Not. **46**(4), 53–64 (2011). DOI 10.1145/1988042.1988046. URL `http://doi.acm.org/10.1145/1988042.1988046`
2. Herndon, T., Ash, M., Pollin, R.: Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. Cambridge Journal of Economics (2013). DOI 10.1093/cje/bet075. URL `http://cje.oxfordjournals.org/content/early/2013/12/17/cje.bet075.abstract`
3. Hudak, P.: Modular domain specific languages and tools. In: Proceedings of Fifth International Conference on Software Reuse, pp. 134–142 (1998)
4. Jones, S.P. (ed.): Haskell 98 Language and Libraries – The Revised Report. Cambridge University Press, Cambridge, England (2003)
5. Jones, S.P., Eber, J.M., Seward, J.: Composing contracts: an adventure in financial engineering (functional pearl). ACM SIGPLAN NOTICES **35**(9), 280–292 (2000)
6. Terei, D.A., Chakravarty, M.M.: An llvm backend for ghc. In: ACM Sigplan Notices, vol. 45, pp. 109–120. ACM (2010)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Fairness in Examination Timetabling: Student Preferences and Extended Formulations

Ahmad Muklason · Andrew J. Parkes · Barry McCollum · Ender Özcan

Abstract Although they have been investigated for more than two decades, university examination timetabling problems are still considered challenging and interesting problems. In our study, we are investigating student preferences for the control of the time gaps between examinations; specifically, what students consider to be best for them and also fair between students. To support this, we conducted a survey of student views and there were two main findings. Firstly, students do have concerns about "fairness within a course", that is, fairness between students within their own course as opposed to only between students in the entire university. Secondly, they do consider some examinations harder than others and would prefer a larger time gap before such hard examinations. To account for these student preferences, we intend to extend the formulation of examination timetabling problems by modifying the objective functions, and this paper briefly describes some options. Ultimately, the aim is to automatically produce fairer examination timetables, and to increase student satisfaction.

Keywords Optimisation · Examination Timetabling Problem · Fairness

1 Introduction

Examination timetabling is a well-known and challenging optimisation problem. In addition to requiring feasibility, the quality of an examination timetable

This work was funded in part by EPSRC grant EP/F033613/1.

Ahmad Muklason, Andrew J. Parkes, Ender Özcan
ASAP Group, School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: {abm, ajp, exo}@cs.nott.ac.uk

Barry McCollum
School of Computer Science, Queen's University, Belfast, BT7 1NN, UK.
E-mail: b.mccollum@qub.ac.uk

is measured by the extent of the soft constraint violations. Standard formulations [2, 5] have penalties for violations of various soft constraints so as to spread out the examinations as evenly as possible in the overall time period, and so give students more time for preparation. However, the standard examination timetabling formulations only minimise the average penalty per student, and this can lead to unfairness in that some students receive much higher penalties than others. Noting that poor timetables may adversely affect academic achievement, we believed that overall student satisfaction could be improved by discouraging unfair solutions. In our prior work [7], we extended the formulation in order to encourage fairness among the entire student body. (Also, for a study of fairness in course timetabling see [6]). However, the notion of "fairness" may be quite complex; hence, to determine student preferences we conducted a survey. This paper briefly reports the main results of the survey and also gives some progress towards associated extensions to the models used for optimisation.

2 Students Perspectives on Fairness

Surveys of preferences in examination timetabling have been conducted before. In [1] the survey was conducted through University registrars. A later survey [3] was directed at students and invigilators; as might be expected, it was found that "Students felt that the most important consideration while preparing the timetable is to have a uniform distribution of exams over the examination period". However, in practice, some students will have poorer distributions than others, and previous surveys had not covered their preferences on how such potential unfairness should be managed. Hence, we conducted a survey to give a deeper understanding of their preferences on the fairness and nature of the distribution of exams. Questions included to what extent issues of fairness matter to them, and the kinds of fairness they prefer. The results showed that the majority of students agreed that fairness should be taken into account. A specific question was whether the timetable should also be fair between students in the same course as opposed to only considering between students in the entire University, and a significant number of students agreed with this. This is, 'fairness within a course' should be considered as well as fairness among the entire student body. This is natural as the students on the same course are their 'competitors' and also colleagues, and dissatisfaction may well arise when a fellow student has much more time for revision before an exam. Note that the notion of 'within a course' may be extended to 'within a cohort' with various different choices for cohorts. For example, a 'cohort' could refer to 'year of study', and justified on the grounds that fairness between final year students is more important than for first years (as the exams typically contribute more to the final degree).

The survey also asked whether they find some exams harder than others, and (unsurprisingly) students agreed with this. They also generally agreed that they need more time for preparation before harder examinations. Presumably a problem with accounting for this is the need to determine perceptions of the hardness of examinations, but, maybe it could be collected from students opinion after taking the examinations, or by simply asking students in advance to nominate which examinations needed more preparation time.

3 Towards an Extended Formulation

A commonly used fairness measure is the 'Jain's Fairness Index' (JFI) [4]. Suppose a set C of students, has associated penalties $P(C) = \{p_i\}$, with mean value, $\bar{P}$, and variance σ_P^2. Then a reasonable measure of the width, and so fairness, is the standard 'Relative Standard Deviation' (RSD) defined by $RSD^2 = \sigma_P^2/\bar{P}^2$. The JFI is then a convenient non-linear function of the RSD:

$$J(C) = \left(1 + RSD^2\right)^{-1} = \frac{\left(\sum_{i \in C} p_i\right)^2}{|C| \sum_{i \in C} p_i^2} \tag{1}$$

and it is (arguably) 'intuitive' as it lies in the range $(0, 1]$ and a totally fair solution (all penalties equal) has JFI=1. For a course (or cohort), C_k, the 'fairness within a course' $J(C_k)$ can be defined by simply limiting to the penalties for the students within C_k. A candidate objective function to enhance fairness within cohorts is then simply the sum of JFI values per cohort:

$$\text{(maximise)} \quad \sum_k J(C_k) \tag{2}$$

As an illustration, consider the case of 2 cohorts with 2 (groups of) students each, and with $P1$ and $P2$ giving the set of penalties for cohorts 1 and 2. Suppose there are two candidate solutions $S1$ and $S2$ with values:

Soln	**P1**	**P2**	**avg(P)**	**J(all)**	**J1**	**J2**	**avg(J1,J2)**
S1	{4,4}	{2,2}	**3**	0.9	1.0	1.0	**1.0**
S2	{4,2}	{4,2}	**3**	0.9	0.9	0.9	0.9

where J(all) is the JFI over all the students and J1 and J2 are the JFI values for the two cohorts. The two solutions have the same overall average penalty, avg(P), and overall JFI. However, we believe that students would prefer solution S1 as it is fairer within each cohort, and this is captured by the higher value of J1+J2. Of course, the situation will not always be so simple. Consider, a second example but with 3 students per cohort, and 3 solutions as follows:

Soln	**P1**	**P2**	**avg(P)**	**J(all)**	**J1**	**J2**	**avg(J1,J2)**
S1	{8,8,9}	{2,2,2}	5.2	0.725	0.997	1.0	**0.998**
S2	{8,8,2}	{8,2,2}	**5.0**	0.735	0.818	0.667	0.742
S3	{7,7,9}	{4,3,3}	5.5	**0.852**	0.985	0.980	0.983

S2 is the lowest overall penalty and would be the standard choice, but is not the fairest both overall and within the cohorts. Potentially, S1 might be preferred because it is most fair within the cohorts, or maybe S3 because it is most fair between all the students. It suggests that there should be a trade-off between overall total penalty, overall fairness, and fairness within cohort. Note that alternatives to the objective function in (3) should also be considered; e.g. for some suitable value of p, to simply minimise the sum of p'th powers of RSDs:

$$\text{(minimise)} \quad \sum_k RSD^p(C_k) \tag{3}$$

or maybe even use an extended version of the JFI with $JFI_p = (1 + RSD^p)^{-1}$.

Details of how best to modify the formulation and solver to account for this multi-objective problem is ongoing work. Finally, for the 'hardness', of exams, we plan to simply give a difficulty index for each exam and use this in modified definitions of penalties, e.g. so that having an exam the day before a hard exam is more penalised that if it were before an easy exam.

4 Conclusion

It is intended that this work will contribute to the generation of examination timetables that match student preferences and enhance their satisfaction. The main contribution is to also account for 'fairness within a cohort of students', rather than only between the entire student body. Ongoing work is investigating how to modify the solvers so as to account for the extended objective functions. Future work will then also study which solutions of the multi-objective problem best match the student preferences, as well as the balance with requirements of the other stakeholders such as teachers and invigilators.

References

1. Burke, E., Elliman, D., Ford, P., Weare, R.: Examination timetabling in British universities: A survey. In: Practice and Theory of Automated Timetabling, *Lecture Notes in Computer Science*, vol. 1153, pp. 76–90. Springer Berlin Heidelberg (1996)
2. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society **47**(3), 373–383 (1996)
3. Cowling, P., Kendall, G., Hussin, N.M.: A survey and case study of practical examination timetabling problems. In: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling PATAT02, pp. 258–261 (2002)
4. Jain, R.K., Chiu, D.M.W., Hawe, W.R.: A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Tech. Rep. TR-301, Digital Equipment Corporation (DEC) (1984)
5. McCollum, B., McMullan, P., Parkes, A.J., Burke, E.K., Qu, R.: A new model for automated examination timetabling. Annals of Operations Research **194**(1), 291–315 (2012)
6. Mühlenthaler, M., Wanka, R.: Fairness in academic course timetabling. Annals of Operations Research (2014). URL http://dx.doi.org/10.1007/s10479-014-1553-2
7. Muklason, A., Parkes, A.J., McCollum, B., Özcan, E.: Initial results on fairness in examination timetabling. In: Proc. of the 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA) (2013)

HyperILS: An Effective Iterated Local Search Hyper-heuristic for Combinatorial Optimisation

Gabriela Ochoa · Edmund K. Burke

Keywords Hyper-heuristics · Iterated local search · Timetabling · Combinatorial optimisation

1 Introduction

Two powerful ideas from search methodologies, iterated local search and hyper-heuristics, are combined into a simple and effective framework to solve combinatorial optimisation problems (HyperILS). Iterated local search is a simple but successful algorithm. It operates by iteratively alternating between applying a move operator to the incumbent solution and restarting local search from the perturbed solution. This search principle has been rediscovered multiple times, within different research communities and with different names [2, 12]. The term *iterated local search* (ILS) was proposed in [11]. Hyper-heuristics [4, 6, 7] are a recent trend in search methodologies motivated (at least in part) by the goal of automating the design of heuristic methods to solve computational search problems. The aim is to develop more generally applicable methodologies. Metaheuristics are often used as the search methodology in a hyper-heuristic approach (i.e. a metaheuristic is used to search a space of heuristics). Machine learning approaches can and have also been used as the high-level strategy in hyper-heuristics such as reinforcement learning, case based reasoning, and learning classifier systems [4]. The ILS hyper-heuristic discussed here uses a form of reinforcement learning to adaptively select the best operator/heuristic to apply at each iteration (in either or both the perturbation and improvement stages) from an available pool of operators with different features. It differs from a standard ILS implementation which uses a

G. Ochoa and E. K. Burke
Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland
E-mail: {gabriela.ochoa, e.k.burke}@cs.stir.ac.uk

single variation operator for each stage. The proposed approach has similarities with variable neighborhood search [13], and adaptive large neighborhood search [17]. Approaches combining local search and perturbation heuristics can be found in both memetic algorithms [10,15] and hyper-heuristics [3,5,16,22]. A previous approach specifically incorporating adaptive operator selection to iterated local search can be found in [21].

The next section gives more details of the proposed algorithmic framework. Section 3 overviews recent successful applications of HyperILS, while Section 4 present some concluding remarks and suggestions for future work.

2 HyperILS

Algorithm 1 shows the high-level pseudo-code of the proposed iterated local search hyper-heuristic (HyperILS). It differs from traditional ILS implementations in the design of the perturbation and improvement stages. Indeed, Algorithm 1 is a template or framework rather than a specific algorithm as there are alternative ways of designing the stages that we have renamed *Hyper-Improvement* and *HyperPerturbation*. These alternatives, however, share two key components. First, multiple heuristics or neighbourhood structures are considered, instead of a single standard one, in either or both stages. These operators need to be of different types and if possible incorporate some problem domain information in the form of ruin-recreate or large neighbourhood heuristics [17,18]. Second, these multiple heuristics are not selected uniformly at random or in a pre-determined order. Instead, they incorporate state-of-the-art adaptive operator selection and reinforcement learning mechanisms. The next section describes in more detail the adaptive mechanisms that have been used within this algorithmic framework.

Algorithm 1 *HyperILS: Iterated Local Search Hyper-heuristic.*

```
$s_0$ = GenerateInitialSolution
$s^*$ = HyperImprovement($s_0$)
repeat
    $s'$ = HyperPerturbation ($s^*$)
    $s^{*'}$ = HyperImprovement($s'$)
    if $f(s^{*'}) \leq f(s*)$ then
        $s^* = s^{*'}$
    end if
until time limit is reached
```

3 Case studies

The first implementation of HyperILS was presented in [3] using the HyFlex framework as a benchmark for cross domain heuristic search [14]. In this implementation, a perturbation operator is selected uniformly at random (from the available pool of mutation and ruin-recreate heuristics) and applied to the incumbent solution; followed by a greedy improvement stage (using all the local search heuristics). The approach is extended in [5] by substituting the uniform random selection of neighbourhoods in the perturbation stage, by online learning strategies. Two strategies were implemented: *choice function* (from the hyper-heuristics literature), and *extreme value based adaptive operator selection* [9] (from the evolutionary computation literature), with the latter producing better overall results. A subsequent implementation, successfully tested on the vehicle routing problem, incorporated a mechanism for adaptively reordering the improvement heuristics [22]. Finally, HyperILS has been recently applied to solve real-world instances of the Course Timetabling [19]. The approach was found to be both general (solving different types of instances), and effective (producing and even improving some state-of-the-art results).

We describe below in more detail the learning mechanisms used in the previous mentioned HyperILS implementations. Adaptive operator techniques comprise two main stages, *credit assignment* and *operator selection.* Credit assignment involves assigning credit or reward to an operator, based upon potentially a number of factors, including their current performance, past performance and the amount of time since it has last been called. There are a variety of credit assignment methods, all attempting to ensure that the strongest performing operator will have the largest amount of credit apportioned to them. Once the operator merits are estimated an operator selection mechanism is used to define the next operator which will be applied. This cycle of credit estimation and operator selection is repeated through the search process.

Our current implementations of HyperILS [5, 19, 22] use *extreme* values for assigning credits [8], which is based on the principle that large (but possibly infrequent) credit improvements are more effective than small frequent improvements. It rewards operators which have had a recent large positive impact, while consistent operators yielding only small improvements receive less reward. Rewards are updated as follows, when an operator *op* is selected, it is applied to the current solution. The quality value of this new solution is computed and the change in quality is added to a list of size W. Thereafter, the operator reward is updated to the maximal value in the list.

Operator selection probabilities are calculated from their quality estimates following a selection rule. These rules maintain a probability vector $(p_i, t)_{i=1,...,K}$ (where K denotes the number of operators), and use the operator's raw credit estimate to calculate probabilities. Our studies use two recent and well performing selection rules, namely, *Adaptive Pursuit* and *Dynamic Multi-Armed Bandit.* Adaptive pursuit was originally proposed for learning automata and was adapted to operator selection in [20]. It follows a winner-takes-all strategy, selecting at each step the operator with maximal reward, increasing its selection probability, while all other operators get their probability reduced. This method has two parameters: p_{min} that indicates the minimal probability of selection for each operator, and β, the *learning rate* taken from $(0, 1]$. The multi-armed bandit framework is commonly used in game theory for studying the *exploration vs. exploitation* dilemma. It involves N arms and a decision making-algorithm for selecting one arm at each time step with the goal of maximising the cumulative reward gathered along time. The exploration vs. exploitation balance is also relevant for heuristic search. Indeed, adaptive operator selection can be formulated using multi-armed bandits with arms corresponding to search operators [8]. Specifically, the *upper confidence bound* multi-armed bandit [1] was used as it provides optimal maximisation of cumulative rewards. Two considerations were required to use this framework for adaptive operator selection. First, a scaling factor C is needed, in order to properly balance the tradeoff between exploration and exploitation. Second, the original setting is static, while adaptive operator selection is dynamic, i.e., the quality of the operators is likely to change along the different stages of the search. The multi-armed bandit framework is thus combined with the *Page-Hinkley* statistical test for detecting changes in the reward distribution, and, upon such a detection, restarting the process [8]

4 Conclusions

HyperILS is a simple yet effective framework combining iterated local search and selective hyper-heuristics. It allows the incorporation of state-of-the-art ideas from reinforcement learning and adaptive operator selection. HyperILS has been successfully applied to both cross-domain search, and solving complex optimisation problems such as Vehicle Routing and Course Timetabling. Applications to other complex optimisation problems will be the subject of future research. For the sake of simplicity, the current framework considers a simple acceptance criterion (accepting all non-worsening solutions). Future work will consider a third adaptive stage corresponding to the acceptance mechanism.

Acknowledgements This research is partly funded by the *Engineering and Physical Sciences Research Council* (EPSRC, grant number EP/J017515).

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2-3), 235–256 (2002)
2. Baxter, J.: Local optima avoidance in depot location. Journal of the Operational Research Society **32**, 815–819 (1981)
3. Burke, E.K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.A., Gendreau, M.: Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In: IEEE Congress on Evolutionary Computation (CEC 2010), pp. 3073–3080. Barcelona, Spain (2010)
4. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society (JORS) **64**(12), 1695–1724 (2013)
5. Burke, E.K., Gendreau, M., Ochoa, G., Walker, J.D.: Adaptive iterated local search for cross-domain optimisation. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11, pp. 1987–1994. ACM, New York, NY, USA (2011)

6. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: F. Glover, G. Kochenberger (eds.) Handbook of Metaheuristics, pp. 457–474. Kluwer (2003)
7. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.: Handbook of Metaheuristics, *International Series in Operations Research & Management Science*, vol. 146, chap. A Classification of Hyper-heuristic Approaches, pp. 449–468. Springer (2010). Chapter 15
8. Fialho, A., Costa, L., Schoenauer, M., Sebag, M.: Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In: Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 5851, pp. 176–190. Springer Berlin Heidelberg (2009)
9. Fialho, A., Costa, L.D., Schoenauer, M., Sebag, M.: Extreme value based adaptive operator selection. In: Parallel Problem Solving from Nature PPSN X, *LNCS*, vol. 5199, pp. 175–184. Springe (2008)
10. Krasnogor, N., Smith, J.E.: A memetic algorithm with self-adaptive local search: TSP as a case study. In: Genetic and Evolutionary Computation Conference (GECCO 2000). Morgan Kaufmann (2000)
11. Lourenco, H.R., Martin, O., Stutzle, T.: Iterated Local Search, pp. 321–353. Kluwer Academic Publishers,, Norwell, MA (2002)
12. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the TSP incorporating local search heuristics. Operations Research Letters **11**(4), 219–224 (1992)
13. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research **24**(11), 1097–1100 (1997)
14. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., Parkes, A.J., Petrovic, S., Burke, E.K.: Hyflex: a benchmark framework for cross-domain heuristic search. In: Proceedings of the 12th European conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP'12, *Lecture Notes in Computer Science*, vol. 7245, pp. 136–147. Springer-Verlag (2012)
15. Ong, Y.S., Lim, M.H., Zhu, N., Wong, K.W.: Classification of adaptive memetic algorithms: a comparative study. IEEE Transactions on Systems, Man, and Cybernetics, Part B **36**(1), 141–152 (2006)
16. Özcan, E., Bilgin, B., Korkmaz, E.E.: Hill climbers and mutational heuristics in hyperheuristics. In: Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006), *Lecture Notes in Computer Science*, vol. 4193, pp. 202–211. Reykjavik, Iceland (2006)
17. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Computers and Operations Research **34**, 2403– 2435 (2007)
18. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics **159**(2), 139 – 171 (2000)
19. Soria-Alcaraz, J.A., Ochoa, G., Swan, J., Carpio, M., Puga, H., Burke, E.K.: Effective learning hyper-heuristics for the course timetabling problem. European Journal of Operational Research **238**(1), 77 – 86 (2014)
20. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, GECCO '05, pp. 1539–1546. ACM, New York, NY, USA (2005)
21. Thierens, D.: Adaptive operator selection for iterated local search. In: Second International Workshop on Engineering Stochastic Local Search Algorithms (SLS 2009), *Lecture Notes in Computer Science*, vol. 5752, pp. 140–144. Springer (2009)
22. Walker, J., Ochoa, G., Gendreau, M., Burke, E.: Vehicle routing and adaptive iterated local search within the HyFlex hyper-heuristic framework. In: Y. Hamadi, M. Schoenauer (eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp. 265–276. Springer Berlin Heidelberg (2012)

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Planning the Amusing Hengelo Festival

Gerhard Post · Martin Schoenmaker

1 Introduction

The Amusing Hengelo festival [1] is an annual musical event in which around 4000 singers participate. These singers belong to one or more of the approximate 100 choirs which usually give two performances: one on a stage outside and one on a stage inside. This leads to the first part of the planning problem: assign the choirs to times and stages such that the requirements are met and the preferences are respected as good as possible. The second part of the planning problem is scheduling the volunteers that supervise the stages. This supervision is needed to look after the equipment on the stages and to assist when problems occur.

The planning problem gets complicated because of the interrelations between the main objects, choirs and volunteers, that we try to plan. We give a more detailed description in the next sections.

2 Planning the choirs

2.1 Stages and Choirs

The Amusing Hengelo festival takes place on the first or second Saturday in June. The festival on June 6, 2015 will be the tenth festival. The public performances of the choirs between 11 AM and 5 PM form the central part of

Gerhard Post
University of Twente
Department of Applied Mathematics
The Netherlands
E-mail: g.f.post@utwente.nl

Martin Schoenmaker
Amusing Hengelo
The Netherlands

the festival. Each performance takes 30 minutes allowing 12 performances on a stage. The stages and choirs have certain properties that have their influence on the planning.

More specific, a stage has the following properties:

- The *public opening time* and *closing time*. These are the times between which performances on the stage can take place.
- A *location* either *indoor* or *outdoor*. Most choirs give exactly one indoor and one outdoor performance.
- A *capacity*, the maximum number of singers that fit on the stage.
- An *equipment level*, ranging from A (no equipment) to E (full electronic equipment with piano). There are rules which higher equipped stages can replace the lower equipped stages.
- The *allowed styles*. Some stages are dedicated to choirs of certain styles, for example in the main church only religious and gospel choirs are allowed.
- A *quality*. Some stages are less attractive than others; it should be avoided the choirs are planned twice on a less attractive stage.

For the planning problem a choir is described by

- The *arrival time* and *departure time*. The performances of the choir should lie between these two times.
- The number of requested *indoor* and *outdoor* performances. The sum is at most 2.
- A *size*, the registered number of singers.
- A *required equipment level*, ranging from A to E. The performances should take place on stages providing the required equipment level.
- The *style*, like 'pop', 'religious', 'barber shop'.

The properties 'capacity', 'equipment level' and 'allowed styles' of the stage versus the properties 'size', 'required equipment level' and 'style' lead to a compatibility matrix for the stage-choir pairs. It is possible to adjust this compatibility matrix by special stage-choir relations, for example to allow a choir of size 41 on a stage with capacity 40.

2.2 Additional requirements

The problem above gets complicated because of choir members and directors (together called 'musicians') that are member of more than one choir; in the datasets there are around 40 of these musicians. Clearly the planning should be such that these persons are not planned double at any time. Moreover it is required that two performances of a musician at consecutive times should be on the same stage ('travel time constraint').

For the planning of the choirs we have the following requirements (H - hard, must be respected) and requests (S - soft, respect as much as possible).

- (H) The performances can only be scheduled on stages that the compatibility matrix allows.

– (H) The time of a performance of a choir on a stage must respect the time windows of the choir and the stage.
– (H) The performances of a choir can not be at consecutive times, neither at times more than three hours apart.
– (S) The number of requested indoor and outdoor performances should be respected.
– (S) The time between two performances should be as close as possible to the preferred time (two hours) between performances.
– (S) A stage of lower quality should be assigned to a choir at most once.
– (S) To retain the audience, stages should be planned without idle times: once a stage has a performance, it should have performances at all the time till the last performance.

Our main objective is to plan the number of requested performances for each choir. Secondary we try to reach good quality choir schedules, as specified by the soft contraints above. At the same time we try to avoid idle times for stages.

3 Planning the volunteers

Most stages cannot be left unattended. This means that we need someone to look after the stage, even if there is no performance. The festival has a list of around 50 volunteers that are willing to do this. The planning of volunteers is aligned with the performances, i.e. in blocks of 30 minutes. All blocks together assigned to a volunteer we call a *shift*.

Unfortunately the planning of the volunteers is not independent of the planning of choirs, because the volunteers are in around 20 % of the cases member of a choir. Like the musicians in Section 2.2 we must avoid double planning of volunteers at any time, and moreover the travel time constraint (planned at consecutive times implies same stage) should hold. A volunteer has the following properties that are important for the planning.

– (H) The *experience*. The first volunteer at a stage should be an experienced volunteer.
– (H) The *arrival time* and *departure time*. The shift of a volunteer should lie between these two times.
– (H) The *maximum shift length*. The shift assigned should not be longer than this maximum.
– (H) The *break rule*. If a shift is longer than 3 hours, there should be a break of one hour after at most 2.5 hours.
– (H) After the break the volunteer must be planned to a different stage.
– (S) If a volunteer is planned to two stages or more, preferable one of the stages is an inside stage, and another one is an outside stage.
– (H/S) Volunteers can have preferences around stages and choirs.

The main objective here is to plan at all times the required number of volunteers to the stages. Secondary we try to reach good quality volunteer schedules, as specified by the soft contraints above.

4 Typology

Although the problem has some interaction it seems convenient to solve it in two phases: planning the choirs and planning the volunteers once the choirs are fixed. Clearly both problems belong to the area of timetabling, [5], and we can try to categorize these two separate problems. The planning of choirs resembles school timetabling problems, see [3]. In fact the problem could be modeled in the XHSTT framework for high school timetabling problems, see [4]. The requested performance are the events to schedule, and the stages are the rooms to be used. The resources attached to an event (the choir and its musicians) should be scheduled without clashes, and we have some preferences on the rooms to use, and the time between the events of the same choir.

Planning the volunteers is slightly different in nature, and resemble call center rostering, see for example [2]. The demands are volunteer requirements for the stages, and we try to fulfill these requirements by assigning shifts to the volunteers. What is very specific here are the extra limitations on the availabilities of the volunteers.

5 Methods and results

For both planning problems we developed algorithms that we describe here. In view of the interrelations between the two parts, first the choirs are planned, and then the volunteers.

5.1 A capacity check

A check is implemented to get an a priori estimate whether the planning of choirs is possible or not, and, if not, to give an indication why not. We construct two networks in which a bipartite graph with arcs reflecting the (choir, stage)-compatibility matrix is the core. We consider two networks, as we can separate the inside and outside performances. To the bipartite graph we add a source node and a sink node. From the source to each choir-node we add an arc with capacity the number of requested inside, respectively, outside performances. Similarly we add an arc from each stage-node to the sink node, with capacity the number times the stage is open for performances. Essentially this network forgets about the timing of the performances; a maximum flow from source to sink is an upperbound for the maximum number of performances that can be planned inside, respectively, outside. The upperbound found here usually is reached by the algorithms described below.

5.2 Algorithms for planning the choirs

In the construction phase we first plan the musicians that belong to four or more choirs. These musicians are performing 8 of the 12 times, which in combination with travel time constraint deserves special attention. In fact the most suitable inside stage and outside stage are selected, and the choirs of the musician are planned in two sequences (with a break) on these stages.

Once the most difficult musicians are planned (and fixed) we try to plan the remaining performances. Since the planning of inside performances is more tight, we first plan those, and after that all outside performances. The choirs are sorted by difficulty and planned to the most suitable stage one by one. We add some randomization in the order of choirs and stages, to be able to repeat this process several times.

In the practical cases the solution is not satisfactory even after several attempts of the construction above. For this reason we implemented several improving algorithms. The first algorithm selects the worst planned choir that is not tabu and finds the best time-stage combination for one of the requested performances of the choir. Then the choir is made tabu, indicating that it will not be considered for some time, and the search restarts.

Usually the tabu-search above will not assign all requested performances. Therefore in the next phase, we take a complementary point of view and consider the stages one by one, trying to improve the selected stage's planning; accordingly we consider the performances on the other stages fixed. For the selected stage we try to increase the total number of planned performances while reducing the idle times. For this we solve a matching problem in a bipartite graph. One side of the bipartite graph consists for the choirs that currently are planned on the stage together with the choirs that still can be planned on this stage, because the requested number of performances is not reached yet. On the other side of the bipartite graph are the times that this stage is available for performances. We add an edge from the choir-node to time-node, if the choir can be planned on the stage at this time, and add costs to the edges reflecting choir planning costs, and our wish to reduce the idle times of the stage. We solve the matching problem and keep on iterating over the stages as long as improvements are found.

5.3 Algorithm for planning the volunteers

Planning the volunteers turned out to be harder than planning the choirs. We developed several local search procedures, that unfortunately were not able to plan all required blocks. For this reason we decided to work in two phases: in the first phase we do local search, yielding a solution with a few percent of unplanned blocks. In the second phase we fix the most restricted volunteers to the shift as planned during the local search, and continue to formulate and solve an ILP model for the remaining volunteers. A direct model (without partial fixing) is also possible, but the time to obtain a solution with all blocks planned takes over 10 hours of computation time, even with a commercial solver. Fixing the easy volunteers reduces this to 10 or 15 minutes.

Year	Choirs	Stages	Volunteers
2012	114	20	51
2013	105	19	26
2014	93	18	46

Table 1 Main characterisitics of the data of 2012, 2013, and 2014

5.4 Results

We developed algorithms for both planning problems and tested them on the planning data of 2012, 2013, and 2014. The main characteristics of the data can be found in Table 1. It gives the number of registered choirs, number of stages, and number of known volunteers at the time of planning; the volunteer data in 2013 is not complete, since the planning was done by hand.

The algorithms for choir planning have been used in all these years, while the approach for volunteers planning was tested this year. The version of 2014 revealed that the choir planning is solved very satisfactory, but for volunteer planning more modeling is needed to describe the acceptable rosters for volunteers. The generated roster served as starting point for manual improvements, and thus already saved hours of work.

References

[1] Amusing Hengelo, http://amusing-hengelo.nl/?l=en.
[2] Ernst A, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1): pp. 3–27.
[3] Pillay N (2013) A survey of school timetabling research. Annals of Operations Research, doi: 10.1007/s10479-013-1321-8.
[4] Post G, Ahmadi S, Daskalaki S, Kingston J, Kyngas J, Nurmi C, Ranson D (2012) An XML format for benchmarks in high school timetabling. Annals of Operations Research 194(1): pp. 385–397.
[5] Schaerf A (1999) A survey of automated timetabling. Artificial Intelligence Review 13(2): pp. 87–127.

Investigation into an Evolutionary Algorithm Hyper-Heuristic for the Nurse Rostering Problem

Christopher Rae
Nelishia Pillay
School of Mathematics, Statistics and Computer Science
+27 33 2605644
stingrae789@gmail.com; pillayn32@ukzn.ac.za

Abstract: Nurse rostering is a well researched domain with various methods successfully applied to solving this problem including constraint programming, integer programming, simulated annealing, tabu search and genetic algorithms. The effectiveness of hyper-heuristics in solving combinatorial optimization problems of this nature has been illustrated in its application to educational timetabling and packing problems amongst others. However, there has not been much research into the use of hyper-heuristics in solving the nurse rostering problem. In particular, multi-point search methods, namely population based hyper-heuristics such as evolutionary algorithm hyper-heuristics, have not been investigated for this domain. The research presented in this paper forms part of a larger initiative aimed at researching the use of evolutionary algorithm perturbative hyper-heuristics in solving the nurse rostering problem. The paper reports on the application and evaluation of an evolutionary algorithm selection perturbative hyper-heuristic (EA-SPHH) in solving the benchmark data set used for the first international nurse rostering competition.

Keywords: hyper-heuristics, nurse rostering, evolutionary algorithm hyper-heuristic

1. Introduction

The nurse rostering problem (NRP) is a two-dimensional timetabling problem that deals with the assignment of nursing staff to shifts across a scheduling period subject to certain constraints (Burke et al. 2004). The constraints are divided into hard constraints, constraints which must be fulfilled, and soft constraints, i.e. those that are desirable. These constraints are set by the hospital's contracts for the staff and their own personal requests. This problem has been proven to be NP-hard.

Hyper-heuristics is a problem solving approach, that aims to have a higher level of generality than traditional optimization methods by operating on a heuristic space rather than a solution space (Burke et al. 2013). Hyper-heuristics select or generate low-level heuristics. The low-level heuristics can be constructive or perturbative. Constructive heuristics are used to construct a solution while perturbative heuristics improve a candidate solution. This study focuses on selection perturbative hyper-heuristics. These hyper-heuristics are categorized as single-point or multi-point.

Previous research on hyper-heuristics for the domain of nurse rostering have essentially been single point selection perturbative hyper-heuristics. Cowling et al. (2002) used a choice function selection perturbative hyper-heuristic to solve 52 instances of the nurse rostering problem for a UK hospital. Burke et al. (2003) employed a selection perturbative hyper-heuristic with a tabu list memory to the same problem set with an improvement in performance. Bai et al. (2010) apply a selection perturbative hyper-heuristic hybridized with a genetic algorithm to solve the same set of problems which produced further improvement in results. Heuristics are chosen based on dynamic performance and acceptance ratios and simulated annealing is used for move acceptance. The hyper-heuristics implemented by Bilgin et al. (2009) to solve the Belgian nurse rostering problem used random heuristic selection and simulated annealing for move acceptance. In later work Bilgin et al. (2010) extend this work by hybridizing a selection perturbative hyper-heuristic and a greedy shuffle approach to solve problems from the first international nurse rostering competition.

Previous research has examined the use of single point selection perturbative hyper-heuristics in solving the nurse rostering problem. Given the success of multi-point hyper-heuristics in other domains such as educational timetable (Burke et al. 2013), this study investigates the use of a multi-point selection perturbative hyper-heuristic, namely an evolutionary algorithm hyper-heuristic, in solving this problem. This study extends the work presented in Rae et al. (2012).

2. Evolutionary Algorithm Selection Perturbative Hyper-Heuristic (EA-SPHH)

The EA-SPHH employs the generational evolutionary algorithm depicted in Figure 1.

```
Create initial population
Repeat
    Evaluate individuals in the population by applying each heuristic in the individual to the
    current solution
    Set best individual's solution as the current roster
    Select parents using tournament selection
    Apply genetic operators to selected parents  and evaluate individuals in population
Until a maximum number of generations has been reached or the solution has converged
```

Figure 1. Generational evolutionary algorithm hyper-heuristic

An initial population is created, this population is then refined iteratively following a process of evaluation, selection of parents and use of genetic operators to create offspring which form the next generation's population. An individual in the population is represented by a string with each character representing a low-level perturbative heuristic. A number of heuristics were tested and developed based on literature and it was decided to use a set of 13 swap operators. In addition to this a "blank" move is included as a means of introducing entropy into the system. Each element of the string is randomly chosen until a string within a maximum and minimum length is created.

The fitness measure of each element of the population is determined by using the individual to improve a nurse roster created by random allocation of shifts to nurses. The fitness of the individual is the sum of the hard and soft constraints violated by the constructed roster. For each generation the best individual's roster replaces the initial roster. This is a form of shared memory as we are giving information to all the individuals in the population. The system and chosen heuristics avoid incurring hard constraint penalties. This is done by only having as many shifts as defined by the cover requirement and only allowing nurses to have single shift days in the roster representation. Tournament selection is used to choose parents for regeneration which involves application of mutation and crossover operators. This selection method returns the fittest individual of a randomly selected number of individuals.

In each generation the population is created by applying the mutation and crossover operators on the selected parents. The mutation operator selects a random position in a parent string and changes the heuristic at that point to a random heuristic from the given set to produce an offspring. The crossover operator randomly selects two points in each of the parents. The substrings at these points are swapped to create two offspring. The fitter offspring is returned as the result of the operation.

Multithreading was introduced into the algorithm in order to improve runtimes and simulations were run on a multicore machine using 8 processors of the available 128 cores per run. Table 1 lists the parameters values used by the evolutionary algorithm which were determined empirically by performing trial runs.

Table 1. Parameter values

Parameter	***Value***
Population Size	100
Initial Chromosome Length	10-25×Cover Size
Tournament size	5
Crossover percentage	70%
Mutation percentage	30%
No. of Runs	10
Maximum no. of Generations	20

4. Results and Discussion

This section discusses the performance of the of the EA-SPHH in solving the benchmark problems of the first international timetabling competition. This included 30 sprint instances (10 early, 10 late and 10 hidden) 15 medium instances (5 early, 5 late and 5 hidden) and 15 long instances (5 early, 5 late and 5 hidden. EA-SPHH was able to produce optimal results for 20 of the sprint instances. For the remaining sprint problem instances the EA-SPHH produced solutions that differed by 1 for 4 instances, 2 for 2 instances, 3 for 1 instance, 5 for 2 instances and 6 for 1 instance. The performance of the EA-SPHH was not as good on the medium and long instance, producing optimal solutions for 7 of the medium instances and 4 of the long instances. The difference from the optimal ranged from a minimum of 1 to a maximum of 36. Forty six of 60 instances were either solved to optimality or within 5 soft constraint violations from the best known solution. The performance of the hyper-heuristic does not compare with the state of the art approaches, however this was not expected as this is a first attempt at employing a multi-point search in solving this problem. The performance of the EA-SPHH is better than the selection perturbative hyper-heuristic hybridized with a greedy shuffle implemented by Bilgin et al. (2010) on the sprint instances and is comparative on the medium and some of the long instances, with the hybrid implemented by Bilgin et al. performing better on the late variant of the long instances. The EA-SPHH was also found to perform much better than the harmony search employed by Awadallah et al. (2011) in solving these problems. EA-SPHH was able to match the solution found by Bilgin et al. (2010) on long_hidden02, the solution found by Burke and Curtois (2010) for the sprint_late04 instance and the solution found by Nonobe (2010) for the sprint_hidden01 instance. These results were previously the best known results. Future research will investigate means of improving the performance of the EA-SPHH. It is anticipated that one of the reasons for the poor performance on medium and late instances is the set of low-level heuristics used. The EA-SPHH essentially uses swap heuristics. The heuristic set does not include ruin and recreate or mutational and crossover heuristics as used in some of the previous work on hyper-heuristics. Future research will investigate methods for investigating the most effective set of heuristics to be used by the EA-SPHH. Furthermore, too big a set of low-level heuristics will lead to a greater number of combinations resulting in a larger search space to explore and possibly poor success rates. In this study a total of 14 low-level heuristics were used.

5. Conclusion and Future Work

The paper is an initial attempt at employing a multi-point selection perturbative hyper-heuristic to solve the nurse rostering problem. An evolutionary algorithm selection perturbative hyper-heuristic was implemented and evaluated on the benchmark set of problems used for the first international nurse rostering competition. While the hyper-heuristic produced good results for the sprint set of problems it did not perform as well on the medium and long instances, particularly on the late problem type. It is hypothesized that this could possibly be attributed to the set of low-level heuristics used and future work will investigate methods for determining the set of low-level heuristics to use. Future work will also investigate the use of a generative perturbative hyper-heuristic to evolve low-level perturbative heuristics and a hybrid hyper-heuristic combining both selection and generation perturbative hyper-heuristics to solve this problem.

Acknowledgements. This work is based on the research supported in part by the National Research Foundation of South Africa for the Grant CSUR13091742778. Any opinion, finding and conclusion or recommendation expressed in this material is that of the author(s) and the NRF does not accept any liability in this regard.

7. References

Awadallah, M. A., Khader, A. T., Al-Betar, M. A., & Bolaji, A. L. (2011). Nurse Scheduling Using Harmony Search. In proceedings of the Sixth International Conference on Bio-Inspired Computing: Theories and Applications, 58–63.

Bai, R., Burke, E. K., Kendall, G., Li, J. & McCollum, B. (2010) A Hybrid Evolutionary Approach to the Nurse Rostering Problem. *IEEE Transactions on Evolutionary Computation*, 14(4), 580–590.

Burke, E. K., & Curtois, T. (2010) An Ejection Chain Method and a Branch and Price Algorithm Applied To The Instances Of The First International Nurse Rostering Competition. Technical Report, School of Computer Science, University of Nottingham.

Bilgin, B., De Causmaecker, P. & Vanden Berghe, G. (2009) A Hyper-Heuristic Approach to Belgian Nurse Rostering. In proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009), pp. 683-689.

Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, W., Vanden Berghe, G. & Wauters, T. (2010) A Hyper-Heuristic Combined with a Greedy Shuffle Approach to the Nurse Rostering Problem. Online. https://www.kuleuven-kulak.be/~u0041139/nrpcompetition/abstracts/ l3.pdf.

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.

Burke, E. & Soubeiga, E. (2003) Scheduling Nurses Using a Tabu-Search Hyper-Heuristic. In Proceedings of First Multidisciplinary International Scheduling Conference: Theory and Applications, pp. 197-218.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. & Qu, R. (2013) Hyper-Heuristics: A Survey of the State of the Art. *Journal of the Operational Research Society*, 1-30.

Cowling, P., Kendall, G., Soubeiga, E. (2002) Hyperheuristics : A Robust Optimisation Method Applied to Nurse Scheduling. *Parallel Problem Solving from Nature — PPSN VII Lecture Notes in Computer Science*, 2439, 851–860.

Haspelagh, S., De Causmaecker, P., Stolevik, M., Schaerf, A. (2012) The First International Competition on Nurse Rostering 2010. *Annals of Operations Research*. DOI: 10.1007/s10479-012-1062-0.

Nonobe, K. (2010) INRC2010 : An Approach Using a General Constraint Optimization Solver. https://www.kuleuven-kulak.be/~u0041139/nrpcompetition/abstracts/m3.pdf.

Rae, C., Pillay, N. (2012) A Preliminary Study into the Use of an Evolutionary Algorithm Hyper-Heuristic to Solve the Nurse Rostering Problem, in Proceedings of the 4th World Congress on Nature and Biologically Inspired Computing (NaBIC 2012), pp. 156-161.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Models for the Shift Design Problem

Troels Martin Range · Richard Martin Lusby · Jesper Larsen

Abstract Rostering and staff scheduling problems often have a predefined set of shifts to which staff members can be allocated. These shifts are typically based on a small set of shift types, where each shift type is characterized by the period of the day the shift covers. The number of people assigned to a shift type reflects to some extent the demand for staff during the period it covers. We address the issue of whether or not we have the correct set of shift types given that we know the required staffing levels over a single day or a set of days.

The Shift Design Problem (SDP) is the problem of identifying the set of shift types prior to the solution of rostering or staff scheduling problems. This should be done in such a way that the demand for staff in each period of the day is matched as closely as possible, but with restrictions on the number of the staff used each day as well as the number of shift types used.

The SDP is a variant of the shift scheduling problem where – among other constraints – the number of shift types used is upper bounded. Shift scheduling has been considered by Dantzig (1954), where undercovering is prohibited and overcovering is free. Aykin (1996) extends this with breaks on the shifts and Mehrotra et al. (2000) develops a Branch-and-Price method for shift scheduling. The SDP is solved heuristically by Di Gaspero et al. (2013).

The motivation for the problem is based on the study by Lusby et al. (2012), who discuss a rostering problem for ground crew at airports. At an airport the required staffing level over a day is correlated with the arrival times of incoming flights and departure times of outgoing flights as well as the number of passengers on the flights. This staffing level demand can be forecast, yielding a demand

Troels Martin Range
Department of Business and Economics, COHERE, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark, Tel.: +45 6550 3685, E-mail: tra@sam.sdu.dk

Richard Martin Lusby
Department of Management Engineering, Technical University of Denmark, Produktionstorvet, building 426, 2800 Kgs. Lyngby, Denmark, Tel.: +45 4525 3084, E-mail: rmlu@dtu.dk

Jesper Larsen
Department of Management Engineering, Technical University of Denmark, Produktionstorvet, building 426, 2800 Kgs. Lyngby, Denmark, Tel.: +45 4525 3385, E-mail: jesla@dtu.dk

curve for each day of the week. With the exception of holiday seasons, it can be observed that this demand is cyclic with a periodicity of one week. Hence we limit our attention to at most seven different demand curves. Based on these demand curves Lusby et al. (2012) show that it is better to let the rostering determine the number of persons used on each shift type than fixing this number prior to solving the rostering problem. However, the authors only consider a small number of predetermined shift types; here we are interested in whether or not this combination of shift types is the best combination.

Let $\mathcal{D}$ be the set of days. Suppose that a day is discretized into periods $\mathcal{T} = \{1, \ldots, T\}$, each covering an interval of time, I_t, such that the day is partitioned by these intervals. Associated with each day $d \in \mathcal{D}$ and each period $t \in \mathcal{T}$ is a known required staffing level $\delta_{dt} \geq 0$. Additionally, a set of possible shifts $\mathcal{S}$ is given, and each shift $s \in \mathcal{S}$ covers a subset of the periods $\mathcal{P}_s \subseteq \mathcal{T}$. The staff demand for any period $t \in \mathcal{T}$ on a given day $d \in \mathcal{D}$ does not need to be strictly satisfied; however, for each unit of under cover (over cover) a penalty cost u_{dt} (o_{dt}) is incurred.

A solution to the SDP is hence a selection of a subset of shift types $\overline{\mathcal{S}} \subseteq \mathcal{S}$ and for each $s \in \overline{\mathcal{S}}$ an assignment of staff $n_{ds} \geq 0$ to each day $d \in \mathcal{D}$. This selection has to be done such that $|\overline{\mathcal{S}}| \leq K$ and $\sum_{s \in \overline{\mathcal{S}}} n_{ds} \leq N$ for each day $d \in \mathcal{D}$. That is, the number of shifts selected is no larger than a pre-specified number K, and no more than N staff members are assigned each day. From such a solution the amount of under cover and over cover in each period of each day can easily be determined. The constraint on the number of shift types used makes the problem difficult as a selection of shifts which enables a close match of the demand for one day may not be able to match the demand of another day well.

We suggest a mixed integer linear programming (MILP) model for the problem. The model uses two types of variables; one indicating whether or not a shift type is used and one counting the number of persons used on a given shift. These are coupled by big-M constraints – one for each shift type – making the resulting LP-relaxation weak. For $|\mathcal{D}| = 1$ the model can be solved to optimality using a commercial solver, while it becomes harder as $|\mathcal{D}|$ increases.

We discuss two approaches to solve the problem. The first approach is a column generation approach in which each column corresponds to a selection of shift types (together with their respective staffing levels) for each day. The master problem then makes a selection of the columns to match the demand on each day. The pricing problem disregards the demand covering constraints and thereby becomes a multidimensional cardinality constrained knapsack problem.

The MILP model exhibits a clear block angular structure where the only common constraint is the number of shift types used and each block corresponds to assigning a staffing level to shifts on a particular day. This lends itself to Benders decomposition where the master problem determines the set $\overline{\mathcal{S}}$, and the subproblems determine n_{ds} for each day $d \in \mathcal{D}$ and $s \in \overline{\mathcal{S}}$. The subproblems generate optimality cuts for the master problem.

We test and discuss our approach in relation to two airport cases. In this discussion we will point out the limitations as well as possible extensions of our models.

References

Aykin, T. (1996). Optimal shift scheduling with multiple break windows. *Management Science*, 42(4):591–602.

Dantzig, G. B. (1954). A comment on edie's "traffic delays at toll booths". *Operations Research*, 2(3):339–341.

Di Gaspero, L., Grtner, J., Musliu, N., Schaerf, A., Schafhauser, W., and Slany, W. (2013). Automated shift design and break scheduling. In Uyar, A. S., Ozcan, E., and Urquhart, N., editors, *Automated Scheduling and Planning*, volume 505 of *Studies in Computational Intelligence*, pages 109–127. Springer Berlin Heidelberg.

Lusby, R., Dohn, A., Range, T. M., and Larsen, J. (2012). A column generation-based heuristic for rostering with work patterns. *Journal of the Operational Research Society*, 63:261–277.

Mehrotra, A., Murphy, K. E., and Trick, M. A. (2000). Optimal shift scheduling: A branch-and-price approach. *Naval Research Logistics (NRL)*, 47(3):185–200.

Course Timetabling Using Graph Coloring and A.I. Techniques

Jordan Rickman and Jay Yellen

Department of Mathematics and Computer Science

Rollins College

1000 Holt Ave.

Winter Park, FL, 32789 USA

jrickman@rollins.edu jyellen@rollins.edu

We present a dual-objective course-timetabling system designed to construct course schedules for the Science Division at Rollins College. Our work builds on the system described in [Wehrer and Yellen (2013)], which models the timetabling problem as a vertex-coloring problem in a weighted graph. The weighted graph model allows the system to incorporate both hard and soft constraints by assigning a 2-component weight to each edge that reflects the undesirability of assigning various pairs of timeslots to its endpoints. An earlier version, using single-component penalty weights, was initially developed in [Kiaer and Yellen (1992)]. The two objectives of our system are: (1) minimize the number and severity of conflicts resulting from the schedule; and (2) create compact schedules for students and faculty. Accordingly, the two edge-weight components are: the conflict penalty, incurred when the endpoints are assigned overlapping timeslots (colors); and the proximity penalty, incurred when the endpoints are assigned timeslots with a large gap between them on the same day. The overall objective is to minimize the total penalty of the completed graph coloring.

Wehrer and Yellen used a one-pass algorithm to color the graph. Heuristics adapted from [Carrington, Pham, et al (2007)] were used to select the most “troublesome” uncolored vertex (a troublesome vertex is one that is likely to create problems if its coloring is deferred), and then to select the best color for that vertex. The process repeats until all vertices are colored. Vertex-selection and color-selection both used linear combinations of a few “primitive” heuristics, as introduced in [Burke, Pham, et al (2008)].

Our system uses a new algorithm that incorporates artificial intelligence techniques via a search-tree representation. The root node is the original uncolored weighted graph, and the nodes at level *k* correspond to the (partial) colorings for which *k* of the vertices have been colored. Each child of a node is the result of assigning a color to an uncolored vertex (see Figure 1). The leaf nodes of the search tree represent complete colorings, and our objective is to find a leaf node of minimum total penalty. Our system performs a best-first search by maintaining a priority queue to determine which nodes to expand. Each node generated is placed in the priority queue according to how “promising” it is.

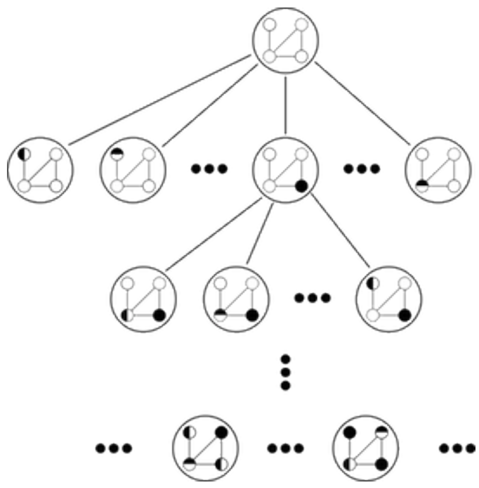


Figure 1: The search tree of 3-colorings of a 4-vertex graph.

Promising nodes are those that appear likely to be the ancestors of low-cost complete colorings. A heuristic evaluation of how promising a node is uses various properties of the partial coloring, including its total penalty and its relation to the rest of the (uncolored) graph. The most promising node is removed from the queue and expanded (i.e., its children are generated) in an iterative process that ends with a complete coloring. The priority queue enables backtracking: if the children of a node have high penalties, they will move towards the back of the queue, and the algorithm will attempt a different branch of the tree. As the number of vertices in the graph increases, the size of the search tree grows exponentially, making a search of the entire solution space intractable. Therefore, we restrict the number of children generated when expanding a node (i.e., the branching factor). We do so by adapting the vertex- and color-selection heuristics to generate a small subset of promising children.

For the purposes of testing, we have developed software that randomly generates course-timetabling problems using a seed problem. The procedure makes random changes to the seed problem, via genetic techniques of mutation and recombination, resulting in a set of problems with similar characteristics. For our seed problem, we use the set of Rollins Science Division courses offered in Fall 2011. We evaluate the effectiveness of various branching and priority-queue strategies by testing them on these randomly generated problems. Currently, we manually adjust various parameters for the branching strategy and the priority queue's heuristic evaluation function, based on their performance on these problems. Our framework will eventually enable us to apply machine learning to adjust these parameters.

Preliminary results on the Fall 2011 problem and 10 randomly generated problems show that even with some simple priority functions and a small branching factor, our search-tree approach produces timetables slightly better to those produced by the old one-pass algorithm. Table 1 compares the performance of the old one-pass algorithm to four different versions of our new algorithm (labeled PQ1, …, PQ4). For the Fall 2011 problem, the table displays the total penalty and its two primary components, conflict penalty and proximity penalty. For the 10 random problems, the values are averaged over the 10 resulting timetables. As we refine our priority function, we expect that our algorithm's performance will continue to improve.

Table 1: Comparison of One-Pass to Priority-Queue Algorithm

Algorithm	Fall 2011			Averages on 10 Random Problems		
	Total Penalty	Conflict Penalty	Proximity Penalty	Total Penalty	Conflict Penalty	Proximity Penalty
One-Pass	7366	83	5291	16364	456	4754
PQ1	7224	100	4724	11601	278	4646
PQ2	7338	85	5213	12594	310	4654
PQ3	7322	99	4847	13974	364	4784
PQ4	7345	86	5195	12149	297	4714

Keywords *Graph Coloring, Heuristics, Timetabling, Weighted Graph, Artificial Intelligence, Genetic Algorithms, Search Algorithms*

References

[Burke, Pham, et al (2008)] Burke, E.K., Pham, N., Qu, R., and Yellen, J., Linear Combinations of Heuristics for Examination Timetabling, Annals of Operations Research, Vol. 194, No. 1 (2012) 89-109.

[Carrington, Pham, et al (2007)] Carrington, J.R., Pham, N., Qu, R., and Yellen, J., An Enhanced Weighted Graph Model for Examination/Course Timetabling, Proceedings of 26th Workshop of the UK Planning and Scheduling (2007) 9-15.

[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.

[Wehrer and Yellen (2013)] Wehrer, A., and Yellen, J., The Design and Implementation of an Interactive Course-Timetabling System, Annals of Operations Research, May 2013.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Set Partitioning Methods for Robust Scheduling: an Application to Operating Theatres Optimisation

Elizabeth Rowse · Paul Harper · Rhyd Lewis · Jonathan Thompson

Robust optimisation techniques and simulation models have been used with column generation methods within the framework of a set partitioning problem for the construction of a weekly operating theatre timetable, the Master Surgery Schedule (MSS). The model takes into account the preferences of surgical specialties for weekly theatre sessions, the stochastic nature of surgical demand and the availability of beds on wards for post-operative recovery.

Operating theatres are very expensive and resource-intensive facilities within modern hospitals; the efficiency of which has a significant impact on patient throughput and the patient experience. Operating theatres experience high demand and can also be seen as a driver of demand on wards and other departments in a hospital as surgical patients will often require other services during post-operative recovery.

Hospitals in the UK are increasingly facing the problem of a large proportion of elective operations being cancelled due to the unavailability of beds on hospital wards for post-operative recovery. The availability of post-operative beds is critical to the scheduling of surgical procedures and the throughput of patients in a hospital. The utilisation of the operating theatre schedule and the impact that it has on the demand for beds on hospital wards is investigated in order to better understand this important relationship. The insights gained from this relationship will aid the construction of a robust operating theatre schedule.

A large teaching hospital in Wales, UK, in which over 25,000 surgical operations are performed annually, is used as a case study. Around 18

A large amount of work has been published in both Operational Research and medical journals on the challenging problem of constructing a weekly MSS timetable which assigns surgical specialties to operating theatre sessions whilst taking account a range of restrictions on resources. These resources can be surgeons, skilled nurses, specialist theatre equipment and the operating

Cardiff University, UK
E-mail: {rowseel, harper, lewisr9, thompsonjm1}@cardiff.ac.uk

theatres themselves. Other aspects to consider while constructing the MSS are the stochastic nature of the demand from hospital waiting lists and that of different surgical procedures; this includes both the occurrence of emergency patients and the duration of the procedures. The process of creating a theatre schedule can therefore be quite arduous and time-consuming, as often there is no systematic approach employed in hospitals that exploits Operational Research techniques.

The work to be presented includes the constraints of post-operative resource requirements, primarily beds on hospital wards, when constructing a practicable and efficient MSS. A set partitioning based optimisation model is used to assign specialties to operating theatres to construct a MSS, and a novel extension to the formulation is used to incorporate constraints on the demand for beds. Simulation of the resulting MSS is then performed in order to measure how robust the MSS is when different aspects of the uncertainty is realised.

A robust optimisation approach to the construction of the MSS is then developed and compared with the results from the nominal formulation in terms of the MSS satisfying the post-operative bed constraints. Initial results are presented to highlight the potential of these approaches to constructing the MSS.

Master State Examination Timetabling

Hana Rudová · Jiří Rousek · Radoslav Štefánik

1 Problem Statement

Master state exams take place at the Faculty of Informatics two times a year. Timetabling of the Master state exams consists of two main tasks: commissions with three examiners are created and each student is assigned to a time slot and to a commission. There is only one student assigned to each commission at any time. Master state exams typically take a week, there are about four commissions a day and about eight students for one commission. The timetable of each commission is defined by a sequence of their students split by a lunch break. Each student has his/her own supervisor and referee who may also serve as examiners. Certainly all teachers (examiners, supervisors and referees) must be available at the scheduled time of the exam. A fair assignment of examiners to commissions with respect to the number of their assigned commissions is necessary.

This base problem is very close to the Bachelor state examination timetabling we have described in [Kochaniková and Rudová(2013)]. The main difference lies in a stronger emphasis on fields of study. Each student has given its field of study and each examiner is associated with an appropriateness factor for each field of study (0–6 corresponding to required, strongly preferred, preferred, neutral, discouraged, strongly discouraged and prohibited). The primary goal of timetabling is the assignment of students to commissions with a good appropriateness factor. Actually we minimize the function

$$\sum_{s \in students} \sum_{e \in commission(s)} w(e,f)^2$$

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

where $w(e, f)$ is a function giving the appropriateness factor of the examiner e for the field of study f. The sum of squares is applied to discourage the use of higher values of appropriateness factor and allow for its fairer distribution.

There are two other optimization criteria related with the supervisors and reviewers. Since some of them may be the members of commissions, it is desirable to minimize the number of students with supervisor or reviewer not being the member of his/her assigned commission. Second, we want to create sequences of students for each supervisor and/or referee such that they may appear at the state exams in the minimal number of these sequences.

2 Solution Approaches

The first author of this paper has been solving this problem manually with the help of the supportive graphical user interface [Petr(2007)] for six semesters. We have been developing two different approaches to solve the described problem. The first approach is based on the CPSolver which was applied in the International Timetabling Competition 2007 (ITC2007) where it was among finalists for all three tracks and it won two of them [Müller(2009)]. Our constraint model is defined with the help of hard constraints which must be satisfied and with the help of soft constraints to handle the optimization criteria described. The current search procedure is similar to the search in the solvers for the competition problems: The iterative forward search is applied to construct the solution where commissions with their students are created one by one, subsequent local search iteratively improves the solution.

The second approach constructively creates commissions based on the number of students and on the properties of particular fields of study and the appropriateness factor of the examiners. In the next step, students are assigned to commissions by another constructive approach with the help of various ordering heuristics based on the appropriateness factor or the number of students for examiners (being their supervisors or referees). Subsequent local search swaps students using simulated annealing. Students with the worst evaluation given by the weighted sum of optimization criteria are exchanged with other students (in the same commission or in a commission where their supervisor or their reviewer is the examiner, etc.). Also commission members can be exchanged, added or removed while keeping their fair allocation. Random local changes removing parts of the solution are also applied to escape from local minimum.

3 Conclusion

Both approaches were applied to solve experimental problems of three semesters. The base problem described here is rather complicated by various exceptions and additional constraints which must be handled in the real life. The second approach based on the local search was applied in practice to solve the

Spring 2014 problem since it was able to handle all features of the real-life problem. Still manual modifications of the generated solution were necessary to remove some troublesome hard constraint violations. Both automated approaches as well as manual solutions violate some of the hard constraints. Further study of problem characteristics and problem definition will be directed to provide a proper constraint model and data definition such that the generated solution could be easily applied in practice. The ultimate goal of this work is an application of one of the solvers to the real-life problem solved each semester.

Acknowledgements This work is supported by the Grant Agency of Czech Republic under the contract P202/12/0306.

References

[Kochaniková and Rudová(2013)] Kochaniková B, Rudová H (2013) Student scheduling for bachelor state examinations. In: Proceedings of the 6th Multidisciplinary International Scheduling Conference – MISTA 2013, pp 762–766

[Müller(2009)] Müller T (2009) ITC2007 solver description: a hybrid approach. Annals of Operations Research 127:429–446

[Petr(2007)] Petr R (2007) Small computer aided system for defences for final thesis and exams (in Czech). Bachelor Thesis, Masaryk University, Faculty of Informatics

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

An Exponential Monte-Carlo Local Search Algorithm for the Berth Allocation Problem

Nasser R. Sabar • Masri Ayob • Graham Kendall

1. Introduction

Over the years, the demand for maritime transportation has rapidly increased [1], [2]. This usually leads to an increasing in competition among different ports in providing efficient services. Port managers face significant challenges in efficiently utilizing the given resources to provide a cost-effective service [1]. Among many optimization and decision making problems in the port management system, the berth allocation problem (BAP) is considered as one of the most challenging and it has a critical role in the ports effectiveness and competitiveness. The BAP seeks an allocation for a given set of vessels berthing positions and berthing times. The main objective of the BAP is to minimize the total waiting time for all vessels in a port [2].

BAP is known to be an NP-hard optimization problem [1], [2], [3]. Thus, due to the exponential growth of the computational time as instance increases, exact methods are usually only applicable for the small-sized instances; despite being able offer optimal solutions if given enough computational resource [3]. Consequently, meta-heuristic algorithms are widely adopted by the researchers to deal with BAP, as they can often return a good quality solution within a reasonable computational time. Examples of meta-heuristic algorithms are: tabu search [2], clustering search [4] and particle swarm optimization [5]. In this work, we propose an Exponential Monte-Carlo with Counter (EMCQ) local search algorithm for the BAP. EMCQ is a variant of simulated annealing, that accepts worse solutions in order to escape from local optima using a non-monotonic acceptance criterion [6], [7]. In addition, to enhance the effectiveness of the proposed EMCQ, we utilize multi-neighborhood operators to effectively explore the search space and also deal with different instance characteristics. The proposed algorithm has been tested on BAP benchmark instances that were used by other researchers and compared with the best known results in the scientific literature [2].

2. Problem description

The BAP has been categorized into two types, based on the berth type and the vessels arrival time [1]. The berth is called a discrete berth if the quay is divided into a set of sections (berths) and a continuous berth if the quay is not divided. The vessel's arrival time is dynamic if the vessels can arrive at any time over the planning horizon, and static if all vessels have arrived at the port before the berth planning starts. In this work, we deal with the BAP that has discrete berths and dynamic arrival times [1]. Given a set of berths and a set of vessels, each vessel is associated with an arrival time, priority and a handling time. Some vessels can be assigned to any berths while other can only be assigned to a subset of berths. The handling time of a vessel is different from one berth to another.

More formally, assign for each vessel a berth and a berthing time on the selected berth while ensuring that each vessel is assigned to exactly one berth and there is no more than one vessel assigned to the same berth at the same time. The overall goal (objective function) is to minimize the total waiting time of all vessels which is calculated as follows [2], [3], [4]:

N. R. Sabar • G. Kendall
The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih Selangor, Malaysia.
E-mail: {Nasser.Sabar, Graham.Kendall}@nottingham.edu.my

M. Ayob
Data Mining and Optimization Research Group (DMO), Centre for Artificial Intelligent Technology (CAIT), Universiti Kebangsaan Malaysia, 43600 UKM Bangi Selangor, Malaysia.
E-mail: masri@ftsm.ukm.my.

$$\min \sum\nolimits_{i\in n} \; \sum\nolimits_{k\in m} v_i \left[T_i^k - a_i + +t_i^k \sum_{j\in n} x_{ij}^k \right] \tag{1}$$

where

- n : number of vessels
- m : number of berths
- v_i : the priority of vessel i
- $T^k{}_i$: the berting time of a vessel i at berth k.
- a_i : the arrival time of vessel i.
- t_i^k : the handling time of vessel i at berth k.
- $x_{ij}{}^k$: descion variable, $x_{ij}{}^k = 1$ if vessel j is serviced by berth k after the vessel i.

3. The proposed algorithm

We propose a local search algorithm for the BAP that follows the general framework of most local search algorithms. That is, generates an initial solution and try to improve it. In the following subsections, we describe the initial solution generation method and the proposed improvement algorithm.

3.1 Initial solution generation method

The initial solution is generated in a random manner. For each vessel, determine the number of the available and feasible berths. Next, randomly assign vessel to a berth from the determined set of berths. If the selected berth for the current vessel is empty, the berthing time of this vessel is the same as the vessel arrival time. If selected berth has some vessels, add the current vessel, sort the assigned vessels based on their arrival time and assign to each one a berthing time based on the current vessels order. This process is repeated until all vessels have been allocated. Calculate the quality (objective function) of the generated initial solution using Equation (1).

3.2 The improvement algorithm

In this work, we utilize the Exponential Monte-Carlo with counter (EMCQ) local search algorithm to further improves the generated initial solution [6], [8]. The EMCQ search strategy is similar to simulated annealing [6], which also accepts worse solutions in order to escape from a local optima but utilizes a different mechanism. EMCQ starts with an initial solution and iteratively modifies it, seeking for a better solution, for a certain number of iterations. The initial solution is modified to generate a neighborhood solution using a neighborhood operator. Then, the quality of the neighborhood solution is calculated using Equation (1) and compared with the initial one. If the quality of the neighborhood solution is better than the initial solution, it will replace the initial solution. Otherwise, the solution might be accepted based on EMCQ acceptance criterion. In EMCQ, the probability of accepting worse solution is calculated as follows: $p= e^{-\Theta/\lambda}$ where $\Theta=\delta{*}t$, $\lambda=q$, δ is the difference between the objective values of the initial and neighborhood solutions, t is an iteration counter, and q is a control parameter that represents consecutive non-improving iterations. The q parameter controls the acceptance of worse solutions which controls the diversification and intensification process. In this work, the initial value of q is set to 1 (q=1) and it will be increased by one (q=q+1) after a certain number of a consecutive non-improving iterations [9], [10]. Once a worse solution is accepted, q will be reset to 1. In EMCQ, the probability of accepting a worse solution decreases as the number of iterations t, increases. However, if there is no improvement for a certain number of consecutive iterations, then the probability of accepting a worse solution will increase according to the quality of the generated neighborhood solution, *q* and *t*. The pseudo-code of the EMCQ is presented in Figure 1 [6], [11].

To improve the effectiveness of the EMCQ in dealing with various instances and also to cope with search landscape changes, we utilize three different neighbourhood operators; where at each iteration of EMCQ a random neighbourhood operator is selected. The utilized neighbourhood operators for the BAP are:

- Nop_1: select one vessel at random and move it to the least cost berth (least handling time).
- Nop_2: randomly select two vessels and swap their berths if feasible.
- Nop_3: select the highest cost vessel and move it to another feasible berth.

```
Generate initial solution, Sol;
Calculate the objective function for Sol, f(Sol);
Set best solution, $Sol_{best} \leftarrow Sol$; $f(Sol_{best}) \leftarrow f(Sol)$;
Set maximum consecutive number of non-improvement, Max_no-improvement;
Set no-improvement counter ← 0;
Set $q = 1$; current iteration counter, $t = 0$, Max_no_iteration;
Do while ($t <$ Max_no_iteration)
    Generate a neighborhood solution, $Sol^*$
    Calculate the objective function of neighborhood solution, $f(Sol^*)$;
    if ($f(Sol^*) < f(Sol)$)   // better solution in term of the objective value
        $Sol \leftarrow Sol^*$;
        $f(Sol) \leftarrow f(Sol^*)$;
        $q = 1$;
        no-improvement = 0;
        if ($f(Sol^*) < f(Sol_{best})$) // update the best solution
            $Sol_{best} \leftarrow Sol^*$;
            $f(Sol_{best}) \leftarrow f(Sol^*)$;
    end if
    else  // accept worse solutions based on the acceptance probability
        Calculate $\delta = f(Sol^*) - f(Sol)$;
        Generate a random number RandNum in [0,1];
        if ($RandNum \leq e^{-\delta * t/q}$)
            $Sol \leftarrow Sol^*$;
            $f(Sol) \leftarrow f(Sol^*)$;
            $q = 1$;
            no-improvement = 0;
        else
            no-improvement++;
            if (no-improvement = Max_no-improvement)
                q++;
                no-improvement = 0;
    end else
    t++;
end while;
Return the best solution, $Sol_{best}$ and $f(Sol_{best})$
```

Figure 1. The pseudo-code of the EMCQ

3.3 Experiments and results

EMCQ was tested on BAP benchmark instances that have been introduced in [2] and widely used by other researchers. The benchmark has 30 different instances (denoted as *i*1 to *i*30); each instance contains 30 vessels and 13 berths. We run EMCQ 31 times for each instance. The EMCQ parameters were set based on a preliminary test as follows: t=1, q=1, *Max_no_iteration* =1,000,000 and *Max_no-improvement* =1,000. The results obtained by

EMCQ (out of 31 runs) and the current state of the art algorithms reported in the literature are presented in Table 1. For each algorithm, we report, for each instance, the best obtained results (best objective value) and the computational time (seconds). In the table, the third column (Opt.) represents the optimal value for each instance [3] and last row represents the average (Avg.). The best results obtained are shown in bold. In this work, we compare the effectiveness of EMCQ with the following algorithms that have obtained the best known results as represented in the scientific literature:

- Generalized set partition programming (GSPP) [3].
- Tabu search (TS) algorithm [2].
- Column generation (CG) algorithm [12].
- Clustering search (CS) [4].
- Particle swarm optimization (PSO) [5].

The results in Table 1 illustrates that, EMCQ obtained the optimal values for all tested instances, i.e., the best results of the EMCQ are the same as those produced by the GSPP as well as CS and PSO on all tested instances. EMCQ outperforms TS on 18 and CG on 4 instances, while producing the same results as TS and CG on 18 and 24 out of 30 instances, respectively. Considering the computational time, EMCQ outperforms other algorithms on all tested instances (see Table 1) and the average computational time of EMCQ is relatively small (3.82, see last row in Table 1). Overall, the EMCQ has a fewer parameters that need to be tuned in advance compared to other algorithms which indicates that EMCQ is an effective and efficient algorithm for the BAP.

Table 1 The results of EMCQ compared to the state of the art methods

Inst.	EMCQ		GSPP		TS	CG		CS		PSO	
	Best	Time	Opt.	Time	Best	Best	Time	Best	Time	Best	Time
i01	**1409**	**6.11**	**1409**	17.92	1415	**1409**	74.61	**1409**	12.47	**1409**	11.11
i02	**1261**	**5.2**	**1261**	15.77	1263	**1261**	60.75	**1261**	12.59	**1261**	7.89
i03	**1129**	**4.3**	**1129**	13.54	1139	**1129**	135.45	**1129**	12.64	**1129**	7.48
i04	**1302**	**6.03**	**1302**	14.48	1303	**1302**	110.17	**1302**	12.59	**1302**	6.03
i05	**1207**	**3.11**	**1207**	17.21	1208	**1207**	124.7	**1207**	12.68	**1207**	5.84
i06	**1261**	**4.32**	**1261**	13.85	1262	**1261**	78.34	**1261**	12.56	**1261**	7.67
i07	**1279**	**3.07**	**1279**	14.6	**1279**	**1279**	114.2	**1279**	12.63	**1279**	7.5
i08	**1299**	**4.65**	**1299**	14.21	**1299**	**1299**	57.06	**1299**	12.57	**1299**	9.94
i09	**1444**	**2.72**	**1444**	16.51	**1444**	**1444**	96.47	**1444**	12.58	**1444**	4.25
i10	**1213**	**2.01**	**1213**	14.16	**1213**	**1213**	99.41	**1213**	12.61	**1213**	5.2
i11	**1368**	**4.11**	**1368**	14.13	1378	1369	99.34	**1368**	12.58	**1368**	10.52
i12	**1325**	**6.52**	**1325**	15.6	**1325**	**1325**	80.69	**1325**	12.56	**1325**	12.92
i13	**1360**	**6.53**	**1360**	13.87	**1360**	**1360**	89.94	**1360**	12.61	**1360**	11.97
i14	**1233**	**3.47**	**1233**	15.6	**1233**	**1233**	73.95	**1233**	12.67	**1233**	7.11
i15	**1295**	**2.96**	**1295**	13.52	**1295**	**1295**	74.19	**1295**	13.8	**1295**	8.3
i16	**1364**	**4.11**	**1364**	13.68	1375	1365	170.36	**1364**	14.46	**1364**	8.48
i17	**1283**	**2.13**	**1283**	13.37	**1283**	**1283**	46.58	**1283**	13.73	**1283**	5.66
i18	**1345**	**3.18**	**1345**	13.51	1346	**1345**	84.02	**1345**	12.72	**1345**	8.02
i19	**1367**	**4.06**	**1367**	14.59	1370	**1367**	123.19	**1367**	13.39	**1367**	11.42
i20	**1328**	**5.13**	**1328**	16.64	**1328**	**1328**	82.3	**1328**	12.82	**1328**	12.28
i21	**1341**	**3.06**	**1341**	13.37	1346	**1341**	108.08	**1341**	12.68	**1341**	7.11
i22	**1326**	**3.82**	**1326**	15.24	1332	**1326**	105.38	**1326**	12.62	**1326**	7.94
i23	**1266**	**3.08**	**1266**	13.65	**1266**	**1266**	43.72	**1266**	12.62	**1266**	7.25
i24	**1260**	**1.98**	**1260**	15.58	1261	**1260**	78.91	**1260**	12.64	**1260**	5.67
i25	**1376**	**3.07**	**1376**	15.8	1379	**1376**	96.58	**1376**	12.62	**1376**	7.13
i26	**1318**	**3.08**	**1318**	15.38	1330	**1318**	101.11	**1318**	12.62	**1318**	7.44
i27	**1261**	**2.06**	**1261**	15.52	**1261**	**1261**	82.86	**1261**	12.64	**1261**	6.16
i28	**1359**	**4.84**	**1359**	16.22	1365	1360	52.91	**1359**	12.71	**1359**	11.52
i29	**1280**	**3.07**	**1280**	15.3	1282	**1280**	203.36	**1280**	12.62	**1280**	8.11
i30	**1344**	**2.86**	**1344**	16.52	1351	**1344**	71.02	**1344**	12.58	**1344**	7.13
Avg	**1306.8**	**3.82**	**1306.8**	14.98	1309.7	1306.9	93.99	**1306.8**	12.79	**1306.8**	8.17

4. Conclusion

In this work, we have proposed an Exponential Monte-Carlo with Counter (EMCQ) local search algorithm for the berth allocation problem. The proposed algorithm starts with an initial solution and iteratively improves it for a certain number of iterations. At each iteration, EMCQ uses a neighborhood operator to generate a neighborhood solution. Improving neighborhood solutions are always accepted, while worse solutions are adaptively accepted based on the quality of the incumbent solution, the search time and the number of consecutive non-improving iterations. To improve the effectiveness of the proposed EMCQ, we utilized three different neighbourhood operators to deal with a different instance characteristics. The proposed algorithm has been tested on berth allocation problem benchmark instances that have been used by other researchers in the literature. Results demonstrated that the proposed algorithm is very promising and can be used to produce good quality solutions compared to state of art methods.

References

[1] C. Bierwirth and F. Meisel, "A survey of berth allocation and quay crane scheduling problems in container terminals," *European Journal of Operational Research,* vol. 202, pp. 615-627, 2010.

[2] J.-F. Cordeau, G. Laporte, P. Legato, and L. Moccia, "Models and tabu search heuristics for the berth-allocation problem," *Transportation science,* vol. 39, pp. 526-538, 2005.

[3] K. Buhrkal, S. Zuglian, S. Ropke, J. Larsen, and R. Lusby, "Models for the discrete berth allocation problem: a computational comparison," *Transportation Research Part E: Logistics and Transportation Review,* vol. 47, pp. 461-473, 2011.

[4] R. M. de Oliveira, G. R. Mauri, and L. A. Nogueira Lorena, "Clustering Search for the Berth Allocation Problem," *Expert Systems with Applications,* vol. 39, pp. 5499-5505, 2012.

[5] C.-J. Ting, K.-C. Wu, and H. Chou, "Particle swarm optimization algorithm for the berth allocation problem," *Expert Systems with Applications,* vol. 41, pp. 1543-1550, 2014.

[6] M. Ayob and G. Kendall, "A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine," in *Proceedings of the international conference on intelligent technologies, InTech*, 2003, pp. 132-141.

[7] N. R. Sabar, M. Ayob, and G. Kendall, "Tabu exponential Monte-Carlo with counter heuristic for examination timetabling," in *Computational Intelligence in Scheduling, 2009. CI-Sched '09. IEEE Symposium on*, 2009, pp. 90-94.

[8] N. R. Sabar and M. Ayob, "Solving Examination Timetabling Problems using Exponential Monte-Carlo with Counter Heuristics," in *Proceeding of the 2008 first conference on Data Mining and Optimization (DMO'08). 3-4 Dec, Universiti Kebangsann Malaysia, pp.16-20.*, 2008.

[9] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A honey-bee mating optimization algorithm for educational timetabling problems," *European Journal of Operational Research,* vol. 216, pp. 533-543, 2/1/ 2012.

[10] N. R. Sabar, M. Ayob, and G. Kendall, "Solving examination timetabling problems using honey-bee mating optimization (ETP-HBMO)," *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), Dublin, Ireland,* pp. 399-408, 2009.

[11] S. Abdullah, N. R. Sabar, M. Z. Ahmad Nazri, and M. Ayob, "An Exponential Monte-Carlo algorithm for feature selection problems," *Computers & Industrial Engineering,* vol. 67, pp. 160-167, 1// 2014.

[12] G. R. Mauri, A. C. Oliveira, and L. A. N. Lorena, "A hybrid column generation approach for the berth allocation problem," in *Evolutionary Computation in Combinatorial Optimization*, ed: Springer, 2008, pp. 110-122.

Timetabling in Higher Education: Considering the Combinations of Classes Taken by Students

Carlos A. Sánchez

Keywords: Higher education, timetabling, association rules, data mining, analytics, market basket analysis, graphs, network analysis

1 Introduction

This work is part of a project that aims to develop an analytics based architecture and methodologies to support the design and implementation of collaborative timetabling systems in higher education. As part of the larger project, this contribution will report on methodologies and algorithms that are being developed to enable the exhaustive identification of combinations of courses that students take from the offered class schedule and combinations of interest that are frequently not possible due to schedule conflicts. This article discusses the architectural components that identify the course offerings that limit the enrollment options for students based on data on schedules and enrollments from recent previous terms.

An aspect that has not been considered in detail in the higher education timetabling literature, relates to the inefficiencies embedded in the constraints that are specified and passed to optimization algorithms. A common approach to timetabling in higher education is to take the requirements and constrains as a given, and then to use optimization algorithms to search for optimal solutions that meet those requirements and constraints. The problem is that optimization algorithms are neither designed nor intended to identify and solve inefficiencies embedded in constraints passed to them.

With the previous ideas in mind, the goal of this work is to help scheduling authorities gain a better understanding of enrollment patterns, identify the schedule offerings that frequently limit the enrollment options for students, and determine the sections that need to be offered that are free of schedule conflicts. The ideas proposed in this work do not require a detailed knowledge about the programs offered at the institution and provide information on courses that are good candidates for section offerings that should be free of schedule conflicts. This information would be intended to help scheduling authorities to produce

School of Information Sciences, University of Pittsburgh
Pittsburgh, PA, 15260, USA
carlosal@pitt.edu

class schedules that better reflect the needs and interest of the students, the institution, and when applicable specify better informed constraints to be passed to timetabling algorithms.

This initiative is in alignment with the identified need to offer students maximum flexibility of choice when selecting courses to take (McCollum, 2007), to improve on measurability and reproducibility of solutions to timetabling problems (Schaerf & Di Gaspero, 2007), to consider the inefficiencies embedded in the input constraints that are provided to the sophisticated optimization algorithms that the research community has developed, and to leverage the existing corpus of knowledge in the field and at the institutional levels (De Causmaecker & Berghe, 2012).

There are initial advances on the referred directions that based on detailed knowledge of an institution's academic programs account for schedule conflicts (Zeising & Jablonski, 2012); that use that knowledge to help transform the curriculum model into the enrollment model (Müller & Rudová, 2012); and that survey faculty on courses that they consider should be offered in non-conflicting schedules (Wehrer & Yellen, 2013).

2 Proposed Approach

Large universities normally have in the order of tens of thousands of students registered in hundreds of programs across schools and departments that offer from hundreds to thousands of sections per term, frequently with multiple sections of the same course offered at different schedules. A common approach to enrollments in U.S. universities is for academic units (i.e. schools and departments) to prepare and offer their class schedules in an independent fashion with little or no coordination across units. Students then enroll in the offered sections across academic units considering the requirements of their programs of study, interests and offerings in the schedule of classes. Students are frequently enrolled in multiple majors and do not advance in curriculum-like synchronization with their peers.

If we are to explicitly consider the enrollment patterns across courses and schools in support of better informed timetabling activities, then it is necessary to: *First*, identify all the unique frequent combination of courses that students take or are able to take per term as well as those combinations that have not been possible due to schedule conflicts but that could be of interest. This is done using enrollment data from recent previous terms. *Second*, identify the course or combinations of courses of interest per term, which are those with section offerings that limit the enrollment options for students. *Third*, enable analyses of data across terms to identify which offerings appear to be limiting the enrollment options over several terms. *Fourth*, develop components to support an interactive interface that enables scheduling authorities to explore and visualize enrollment patterns and if required work on a collaborative fashion to produce better class schedules.

The exhaustive identification of unique course combinations that students take during a term is performed by modeling the problem as an association rules analysis (Agrawal, Imieliński, & Swami, 1993; Agrawal & Srikant, 1994; Krajca, Outrata, & Vychodil, 2011; Srikant, Vu, & Agrawal, 1997). As summary, a transaction is defined as the group of courses that an individual student takes during a term (e.g. MATH 0200, ENGLISH 0220 and ECONOMICS 0100 where the label refers to the academic subject and the number to the catalog/level of the course). Each course is an item in the transaction and the group of courses in the transaction is a course set. Having the data modeled in the described form enables the direct use of association rules algorithms implemented in multiple software packages.

Among others, the output of the association rules analysis indicates how many students enrolled in each course set. As an illustration, one course set could be MATH 0200, ENGLISH 0220 and ECONOMICS 0100 with 100 students. In the next step it is then necessary to identify if 100 students enrolled in that particular course set because no more students were interested (i.e. there is capacity left) or because an enrollment limit was reached due to schedule conflicts or number of seats offered. Those are course sets of interest as they include courses with sections that potentially limit the enrollment options for students. To identify them, it is necessary to analyze the data set at the level of individual sections and schedules.

A backtracking algorithm and methodologies are proposed to identify the course sets of interest considering actual enrollments, enrollment limits and schedules at the level of individual sections on each course set. A de-normalized relational schema is proposed to support the operations of the referred algorithm, longitudinal analyses of historical enrollment data. Other architectural components that will facilitate a collaborative approach to timetabling will be discussed in a separate article.

Prototypes of the referred relational schema and algorithm have been developed and are being tested and refined using actual undergraduate enrollment data from five recent fall terms at the University of Pittsburgh (Pitt) main campus. This campus enrolls approximately 19,000 undergraduate students in 11 of its 17 academic units; they take about 90,000 seats in 3,000 sections each term. Approximately 80% of the students enroll in three or more different subjects across different departments and schools with 56% taking classes in four or five subjects. The processing of enrollment data for five recent fall terms renders data sets with an average of 45,000 closed course sets per term. The reason for the relatively large number of closed course sets is that the association rules algorithms consider the course sets that students take and subsets that are frequent.

As illustration, Table 1 below shows a sample of seven course sets obtained using preliminary results from the components referred above on Pitt's undergraduate enrollments during a recent fall term. The results enable the identification of four course sets of interest in the sample group that appear to be limiting the enrollment options for students. Students enroll in sections of their preference as available in the offered schedule. After the enrollment period is closed, two of the course sets have subjects with sections that reached or exceeded the enrollment capacity offered and two of them have schedule conflicts that limit further enrollments in the course set.

Sample Course Sets	Enrolled in Set	Seats Left in Set	Interest
ARTSC_CHEM_0960, ARTSC_MATH_0220, ENGR_ENGR_0081	243	0	Oversubscribed: ENGR 0081
ARTSC_CHEM_0960, ARTSC_MATH_0220, ENGR_ENGR_0011	241	26	
ARTSC_ECON_0100, ARTSC_ENGCMP_0200, ARTSC_MATH_0120, CBA_BUS_0010	59	8	
ARTSC_ECON_0100, ARTSC_ENGCMP_0200, ARTSC_MUSIC_0711	12	0	Oversubscribed: MUSIC 0711
ARTSC_ITAL_0001, ARTSC_MATH_0120, CBA_BUS_0010	9	8	
ENGR_CHE_0100, ENGR_CHE_0101, ENGR_ENGR_0020	6	0	Conflict: ENGR 0020
CBA_BUSACC_0040, CBA_BUSHRM_1050, CBA_BUSMKT_1441	4	0	Conflict: CBA_BUSACC_0040

Notes:
Course set labels indicate a course School, Subject Code and Catalog Number

ARTSC	The Kenneth Dietrich School of Arts and Sciences		
CBA	College of Business Administration		
ENGR	The Swanson School of Engineering		
BUS…	Business (ACC: Accounting; HRM: Human Resources Management; MKT: Marketing)		
ENGR…	Engineering (CHE: Chemical Eng)		
CHEM	Chemistry	ITAL	Italian
ECON	Economics	MATH	Mathematics
ENGCMP	English Composition	MUSIC	Music

Table 1: Sample of course sets from actual undergraduate enrollments during a recent fall term at the University of Pittsburgh

Table 2 below illustrates the details on schedules and enrollments on all sections for one of the course sets listed in Table 1 (ENGR_CHE_0100, ENGR_CHE_0101, ENGR_ENGR_0020) after the enrollment period has ended. Even though there were seats left in sections of each of the courses, it appears that it was ultimately a schedule conflict that prevented more than six students from enrolling in this course set during the analyzed term.

Course #	School	Subject Code	Catalog #	Class #	Days	Start Time	Stop Time	Enrollment CAP	Enrolled	Seats Left	Seats Left Catalog
1	ENGR	ENGR	0020	14327	M W	16:00	17:15	70	71	-1	10
1	ENGR	ENGR	0020	14443	T H	9:30	10:45	70	70	0	10
1	ENGR	ENGR	0020	14566	T H	9:30	10:45	70	59	11	10
2	ENGR	CHE	0100	14484	M W F	8:00	9:50	65	56	9	18
2	ENGR	CHE	0100	23971	M W F	8:00	9:50	65	56	9	18
3	ENGR	CHE	0101	14485	H	8:00	9:50	65	53	12	25
3	ENGR	CHE	0101	23963	T	8:00	9:50	65	52	13	25

Table 2: Schedules and enrollments on individual sections offered in a selected course set

Currently, work is advancing on: *First*, algorithm improvements to reduce processing time. *Second*, there is ongoing development of analyses and metrics across several terms (i.e. longitudinal analyses) to provide better and more informative identification of course sets of interest. That is, those course sets that appear to limit enrollments across multiple terms. *Third*, components are being extended to enable the identification of negative association rules. This entails the identification of n-tuples of courses that cannot be taken together due to schedule conflicts. Those *n-tuples* do not show up in the association rules analysis as students cannot take those course sets. *Fourth*, development of a graph/network based visualization to facilitate understanding of enrollment patterns and identification of courses that would benefit from collaborative scheduling efforts.

References

Agrawal, R., Imieliński, T., & Swami, A. (1993). *Mining association rules between sets of items in large databases.* Paper presented at the ACM SIGMOD Record.

Agrawal, R., & Srikant, R. (1994). *Fast algorithms for mining association rules.* Paper presented at the Proc. 20th Int. Conf. Very Large Data Bases, VLDB.

De Causmaecker, P., & Berghe, G. V. (2012). Towards a reference model for timetabling and rostering. *Annals of Operations Research, 194*(1), 167-176.

Krajca, P., Outrata, J., & Vychodil, V. (2011). *Using frequent closed itemsets for data dimensionality reduction.* Paper presented at the Data Mining (ICDM), 2011 IEEE 11th International Conference on.

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): Springer.

Müller, T., & Rudová, H. (2012). *Real-life curriculum-based timetabling.* Paper presented at the Dag Kjenstad, Atle Riise, Tomas Eric Nordlander, Barry McCollum and Edmund Burke. Proccedings of the 9th International Conference on the Practice and Theory of Automated Timetabling. Son, Norway: SINTEF.

Schaerf, A., & Di Gaspero, L. (2007). Measurability and reproducibility in university timetabling research: discussion and proposals *Practice and Theory of Automated Timetabling VI* (pp. 40-49): Springer.

Srikant, R., Vu, Q., & Agrawal, R. (1997). *Mining association rules with item constraints.* Paper presented at the KDD.

Wehrer, A., & Yellen, J. (2013). The design and implementation of an interactive course-timetabling system. *Annals of Operations Research*, 1-19.

Zeising, M., & Jablonski, S. (2012). *A Generic Approach to Interactive University Timetabling.* Paper presented at the ACHI 2012, The Fifth International Conference on Advances in Computer-Human Interactions.

The second International Nurse Rostering Competition

Sara Ceschia · Nguyen Thi Thanh Dang · Patrick De Causmaecker · Stefaan Haspeslagh · Andrea Schaerf

Abstract Announcement of the second international nurse rostering competition.

Keywords Nurse Rostering · Competition

1 Introduction

Nurse rostering is a very important problem in healthcare management. Early papers date from the seventies, but especially in the last decade, it has drawn significant attention, see [3, 4] for a review of literature and a classification.

The First International Nurse Rostering Competition (INRC-I) [7] was run in 2010. The competition welcomed 15 submissions in three categories (sprint, medium and long tracks). Since then, several groups also took this formulation and the corresponding instances as a challenge [1, 2, 5, 6, 8, 9] and

Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX. (http://comex.ulb.ac.be)

Sara Ceschia, Andrea Schaerf
Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica
Università di Udine, via delle Scienze 206, 33100, Udine, Italy
Tel: +39 0432 55 8280, fax: +39 0432 55 8251
E-mail: {sara.ceschia,schaerf}@uniud.it

Nguyen Thi Thanh Dang, Patrick De Causmaecker
KU Leuven, Department of Computerscience, CODeS & iMinds-ITEC, KULAK, E. Sabbelaan 53, 8500 Kortrijk, Belgium
Tel.: +32 56 24 60 02, Fax: +32 (0) 56 246052,
E-mail: {nguyenthithanh.dang, patrick.decausmaecker}@kuleuven-kulak.be

Stefaan Haspeslagh
Technologie en informatica - VHTI, Doorniksesteenweg 145, 8500 Kortrijk, Belgium
Tel: 056 26 41 20, fax: 056 21 98 67,
E-mail: stefaan.haspeslagh@vives.be

produced remarkable results. Optimal solutions as well as new best solutions have also been found and reported [2,8,9].

The problem considered for INRC-I was the assignment of nurses to shifts in a fixed planning horizon, subject to a large number of hard and soft constraint types.

For the Second International Nurse Rostering Competition (INRC-II), we propose a smaller set of constraint types, but within a *multi-stage* formulation of the problem. That is, the solvers of the participants are requested to deal with a sequence of cases, referring to consecutive weeks of a longer planning horizon (4 or 8 weeks).

The search method designed by the participants has to be able to solve a single stage of the problem corresponding to one week. Some information, called *history*, is carried out between consecutive weeks, and the one coming from the previous week has to be taken into account by the solver. The history includes *border* data, such as the last worked shift of each nurse, and counters for cumulative data, such as total worked night shifts. Counters' value has to be checked against global thresholds, but only at the end of the planning period.

The organizers provide a simple command-line simulation/validation software to be used to evaluate the quality of the solver. The *simulator* invokes the participant's solver for each stage iteratively, then updating the history after each single execution. The provided *validator* concatenates the solutions for all weeks, and evaluates them all together, along with the cumulative data coming from the final history.

The solver should take into account the following separate input sources:

Scenario: Information that is global to all weeks of the entire planning horizon, such as nurse contracts and shift types.

Week data: Specific data of the single week, like daily coverage requirements and nurse preferences for specific days.

History: Information that must be passed from a week to the other, in order to compute constraint violations properly. It includes border information and global counters.

The solver must deliver an output file, based on which, the simulator computes the new history file, to be passed back to the solver for the solution of the next week.

The rules and early instances of INRC-II will be made available at the day of the presentation. The competition will be run during the fall of 2014 till summer of 2015 and its results will be presented at MISTA 2015 in Prague. Winners will be awarded a financial prize, as well as free conference fees.

References

1. Mohammed A Awadallah, Ahamad Tajudin Khader, Mohammed Azmi Al-Betar, and Asaju La'aro Bolaji. Nurse rostering using modified harmony search algorithm. In *Swarm, Evolutionary, and Memetic Computing*, pages 27–37. Springer, 2011.

2. Edmund K Burke and Tim Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 2014.
3. Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.
4. Patrick De Causmaecker and Greet Vanden Berghe. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1):3–16, 2011.
5. Federico Della Croce and Fabio Salassa. A variable neighborhood search based matheuristic for nurse rostering problems. *Annals of Operations Research*, pages 1–15, 2010.
6. Martin Josef Geiger. Personnel rostering by means of variable neighborhood search. In *Operations Research Proceedings 2010*, pages 219–224. Springer, 2011.
7. Stefaan Haspeslagh, Patrick De Causmaecker, Andrea Schaerf, and Martin Stølevik. The first international nurse rostering competition 2010. *Annals of Operations Research*, pages 1–16, 2012. Online first.
8. HG Santos, TAM Toffolo, S Ribas, and RAM Gomes. Integer programming techniques for the nurse rostering problem. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2012)*, pages 256–283, 2012.
9. Ioannis P Solos, Ioannis X Tassopoulos, and Grigorios N Beligiannis. A generic two-phase stochastic variable neighborhood approach for effectively solving the nurse rostering problem. *Algorithms*, 6(2):278–308, 2013.

Hybridizing Integer Programming and Metaheuristics for Solving High School Timetabling

Matias Sørensen · Thomas R. Stidsen

Keywords High School Timetabling · Integer Programming · XHSTT · Metaheuristics · Matheuristics

1 Introduction

The preferred solution methodology within the area of high school timetabling is *heuristics* and the sub-field of *meta-heuristics*. *Integer Programming* (IP) is generally believed to be less efficient, and many researchers seem to apply heuristics per default when facing a *High School Timetabling Problem* (HSTP). The hybridization of IP and metaheuristics has been studied for many decades, but has recently attracted attention under the alias of *matheuristics* (by the contraction of Mathematical Optimization and Metaheuristics). For instance, Ryan (2012) argues that it is time to *enjoy the best from both worlds*. See also Maniezzo et al (2009a) and Maniezzo et al (2009b) for some recent collections of work within this area.

In this text it will be shown that also for the HSTP it can be beneficial to apply matheuristics. Specifically, the general format XHSTT is considered (see Post et al (2012) for a description of the format). This format can be used for both exchanging models and solutions for the HSTP, and currently there are around 50 instances publicly available (originating from 12 different countries). As a test-bed for computational results, instances from the *International Timetabling Competition 2011* (ITC2011) (Post et al 2013) are used.

2 Hybridizing Integer Programming and Metaheuristics

Caserta and Voss (2010) describe the hybridization of exact methods and metaheuristics in terms of a master-slave structure. That is, either (I) the

Matias Sørensen
MaCom A/S, DK
Tel.: +45 3379 7900
E-mail: ms@macom.dk

Thomas R. Stidsen
Management Science
Department of Management Engineering
Technical University Of Denmark

metaheuristic acts as master at a higher level and controls the exact approach or (II) the exact method acts as the master and controls the metaheuristic. In this text, algorithms of type (I) are considered, and the exact method consists of an IP model solved using a generic IP-solver. Thereby the exact method can be thought of as a *local search* component, and the metaheuristic guides the local search on an overall level.

The IP model for the XHSTT format, which is used as a basis for the matheuristic, was recently developed in-part by the present authors. This IP model is capable of handling any instance of the XHSTT format, and therefore the same applies for the presented matheuristic. The matheuristic is described in the following.

Given is a problem instance P and set of neighborhoods N. The set of decision-variables $\mathcal{X}$ (which are part of the IP model) describes a solution to the problem instance. The basic idea of the algorithm is to modify a subset of variables $V \subseteq \mathcal{X}$ in each iteration of the algorithm (which is done in practice by invoking the IP-solver). Each neighborhood $n \in N$ defines a certain way of selecting V (possibly in a problem-dependent way), incorporating some element of randomness. This element of randomness is incorporated to help the algorithm escape local optima. With each neighborhood $n \in N$ is associated a size-parameter, which determines the amount of variables to select. This size-parameter is adjusted throughout the algorithm, based on how hard the neighborhood is to handle for the IP-solver. The idea of this adaptive layer is to obtain neighborhoods which are efficiently handled by the IP-solver.

The pseudo-code for the matheuristic is shown in Algorithm 1. In Line 3, an initial solution is constructed using a simple heuristic (in this case a greedy algorithm). In Line 5, a neighborhood is chosen. This selection is biased towards neighborhoods which have previously performed well, according to some measure. In lines 6 and 7 the neighborhood is applied, by selecting the set of variables to fix, and thereby the set of variables which are left free for the IP-solver to modify. In Line 8 the IP-solver is invoked. In case the IP-solver is unable to improve the current solution, the current solution is maintained. In Line 9 the adaptive layer of the algorithm adjusts the size of the chosen neighborhood, according to the final IP-gap found by the IP-solver. Line 10 un-fixes variables (in preparation for the next iteration of the algorithm).

Algorithm 1 Matheuristic pseudo-code

```
 1: input: XHSTT problem instance $P$, neighborhoods $N$
 2: output: feasible solution $S$
 3: $S :=$ construct initial solution of $P$
 4: while stopping criteria not met do
 5:     choose neighborhood $n$
 6:     obtain variables $V := n(S)$
 7:     fix variables $\mathcal{X} \setminus V$ to their current value
 8:     invoke IP-solver on $S$ with short timelimit
 9:     adjust size of $n$ subject to the obtained IP-gap
10:     unfix all variables
11: end while
```

Future research will investigate more advanced approaches for adjusting the size of the neighborhoods, and more advanced ways of selecting variables in each neighborhood. Such approaches will hopefully lead to more efficient algorithms.

3 Preliminary Results

For establishing computational results, Gurobi 5.5 is used as IP-solver. This is a commercial product, which is among the best IP solvers currently available. The XHSTT instances of round 2 of ITC2011 are used as test-bed. In this round of the competition, the performance of the finalist algorithms were compared on the same computer using the same time-limit (1000s). The ITC2011 organizers have released a benchmark executable which can be used for determining the equivalent runtime on any given computer. This gives us the opportunity to perform a fair comparison with the ITC2011 round 2 finalists, apart from the fact that the use of commercial software was not allowed in ITC2011. Nevertheless, the presented test-setup provides a good setting for showing the potential of the matheuristic. Table 1 shows the obtained results. These preliminary results show that the matheuristic finds the best solution on 9 of the 18 instances. Furthermore the matheuristic achieves second place when considering average ranks of the algorithms. These are promising results, and show the potential of hybridizing mathematical programming and metaheuristics.

Table 1 Performance of the matheuristic compared to the finalists of round 2 of ITC2011 and the IP solved directly with an IP-solver. For each instance is listed the average solution found from each of the competitors of round 2 of ITC2011 (columns *GOAL*, *HySTT*, *Lectio* and *HFT*), the performance of the IP when solved directly using Gurobi (column *IP*) and the performance of the matheuristic (the average solution obtained over 10 runs). An objective of a solution to a XHSTT instance is denoted (H, S), where H and S denote the cost of violation of the hard-constraints and the soft-constraints, respectively. The best solutions are marked in **bold**. Row *Avg. Ranks* denotes the average ranking of each solution method, 1 being best.

		GOAL	HySST	Lectio	HFT	IP	Matheuristic
BR	Instance2	(1, 62)	(1, 77)	38	(6, 190)	46	**6**
BR	Instance3	124	118	152	(30, 283)	39	**27**
BR	Instance4	(17, 98)	(4, 231)	(2, 199)	(67, 237)	(5, 286)	**58**
BR	Instance6	(4, 227)	(3, 269)	230	(23, 390)	682	**57**
FI	ElementarySchool	4	(1, 4)	**3**	(30, 73)	**3**	**3**
FI	SecondarySchool2	**1**	23	34	(31, 1628)	(1604, 3878)	6
GR	Aigio	**13**	(2, 470)	1062	(50, 3165)	(1074, 3573)	180
IT	Instance4	454	6926	651	(263, 6379)	17842	**48**
XK	Instance1	(59, 9864)	(1103, 14890)	(275, 7141)	(989, 39670)	(3626, 2620)	**(9, 23525)**
NL	Kottenpark2003	**90928**	(1, 56462)	(50, 69773)	(209, 84115)	(8491, 6920)	(238, 43143)
NL	Kottenpark2005A	**(31, 32108)**	(32, 30445)	(350, 91566)	(403, 46373)	(2567, 53)	(566, 19968)
NL	Kottenpark2008	**(13, 33111)**	(141, 89350)	(209, 98663)	-	(14727, 5492)	(6112, 353671)
NL	Kottenpark2009	**(28, 12032)**	(38, 93269)	(128, 93634)	(345, 99999)	(17512, 140)	(9418, 705605)
ZA	Woodlands2009	(2, 14)	(2, 70)	**(1, 107)**	(62, 338)	(1801, 705)	(2, 429)
ES	SpainSchool	894	1668	2720	(65, 13653)	(1454, 11020)	**485**
GR	WesternGreece3	**6**	11	(30, 2)	(15, 190)	25	**6**
GR	WesternGreece4	**7**	21	(36, 95)	(237, 281)	81	12
GR	WesternGreece5	**0**	4	(4, 19)	(11, 158)	15	**0**
Avg. Ranks		2.2	3.4	3.3	5.3	4.6	2.3

References

Caserta M, Voss S (2010) Metaheuristics: Intelligent problem solving. In: Maniezzo V, Stützle T, Voss S (eds) Matheuristics, Annals of Information Systems, vol 10, Springer US, pp 1–38

Maniezzo V, Stützle T, Voß S (2009a) Matheuristics: Hybridizing metaheuristics and mathematical programming. Annals of Information Systems 10

Maniezzo V, Voss S, Hansen P (2009b) Special issue on mathematical contributions to metaheuristics. Journal of Heuristics 15(3)

Post G, Ahmadi S, Daskalaki S, Kingston J, Kyngas J, Nurmi C, Ranson D (2012) An xml format for benchmarks in high school timetabling. Annals of Operations Research 194:385–397

Post G, Gaspero L, Kingston J, McCollum B, Schaerf A (2013) The third international timetabling competition. Annals of Operations Research

Ryan D (2012) It is time to enjoy the best of both worlds. In: The 46th ORSNZ Conference, Victoria University of Wellington, New Zealand

Online Scheduling System for Server Based Personnel Rostering Applications

Přemysl Šůcha · István Módos · Roman Václavík · Jan Smejkal · Zdeněk Hanzálek

Keywords Scheduling · Distributed Computing · Grid Computing · Machine Learning · Personnel Rostering

1 Introduction

In the recent time many software applications have web based character. It means that the computer of the user is used as a terminal while the application is running on a server. Nevertheless, the server may be a bottleneck of the application from the performance point of view. All data manipulations and all computations are processed there and if the server is overloaded it usually causes slow reactions of the application measured by *response time* of the server. Moreover, if the server process non-trivial requests like solving a combinatorial problem, e.g. *personnel rostering* [1], the performance of the server is significantly worse. In this case the incoming requests must be carefully scheduled in order to efficiently exploit available computational resources and to minimize the system response time.

In this paper we consider a server based *personnel rostering application* where many users send their requests at the same time. To solve the requests the server uses a *heterogeneous grid*, i.e. a collection of computer resources such that each one has different speed. A single request of a user is called task which has to be mapped and solved on a computer resource. Tasks are instances of combinatorial optimization like personnel rostering, personnel rerostering etc. The tasks are usually solved by *anytime algorithms*. It means that if the algorithm is interrupted earlier (but not before the minimal execution time) the solution is a valid solution to the problem, however its *quality* in terms of

P. Šůcha (✉) · I. Módos · R. Václavík · J. Smejkal · Z. Hanzálek
Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic
E-mail: suchap@fel.cvut.cz

the objective function value is lower. This is an important aspect which allows deteriorating quality of solutions in a defined range when the server side is overloaded. The aim of a *scheduling system* that schedules the incoming tasks on a heterogeneous grid is to balance average response time of the server and average quality of solutions.

There are already many papers dealing with task mapping on grids [5]. However, to the best of our knowledge there is no work dealing with heterogeneous grids taking into account properties of anytime algorithms. As a related paper can be considered e.g. [2] where the authors describe an algorithm for task mapping on GPU resources while considering the trade-off between the quality of solutions and execution time. However, their approach requires specially designed anytime algorithms where the scheduler is able to control the execution path of the algorithms. It also requires quite frequent communication between the scheduling algorithm and the computer resources. A related domain is scheduling of imprecise computation tasks [4] where a task consists of two subtasks: a mandatory subtask and an optional subtask. Each task must be completed before its deadline while its mandatory subtask must be completed and the optional part may be left unfinished. The objective is to minimize a penalty proportional to the lengths of unfinished parts of all tasks. In contrast to our work, where duration of the tasks for given quality is not known, their penalty function is given a priori.

As stated above, the existing approaches cannot be efficient for scheduling of anytime algorithms that solve combinatorial problems. The problem is that the execution time of a task for the given solution quality is not known a priori. That means the incorrectly estimated execution time may cause deterioration of response time of the system. Therefore we propose a scheduling system for online scheduling of tasks having character of anytime algorithm executed on a heterogeneous grid. Based on the task features the scheduling algorithm is able to estimate the execution time of tasks. Moreover, a controller in the scheduling system is able to balance the average solution quality in order to achieve reasonable trade-off between the average response time and the average quality of solutions.

2 Scheduling System Design

Our design of the scheduling system is outlined in Figure 1. *External applications*, controlled by users, generate input tasks TI_i (rrt_i is requested response time for task TI_i). These tasks are sent into the *task list* through the *estimator* which defines estimated execution time $p_i(q_t) = \max(a_i e^{b_i q_t}, met_i)$ of TI_i where q_t is demanded quality in time t and met_i is the minimal execution time of task TI_i. The *scheduler* then selects tasks from the task list and allocates them to the resources with respect to $p_i(q_t)$ and speed coefficient s_r of each resource r [3]. The actual task allocation is executed through the *resource manager* which passes the task data (TI_i) and its assigned time for execution ($p_i s_r$) to the particular *resource* r. Resources may be arbitrary com-

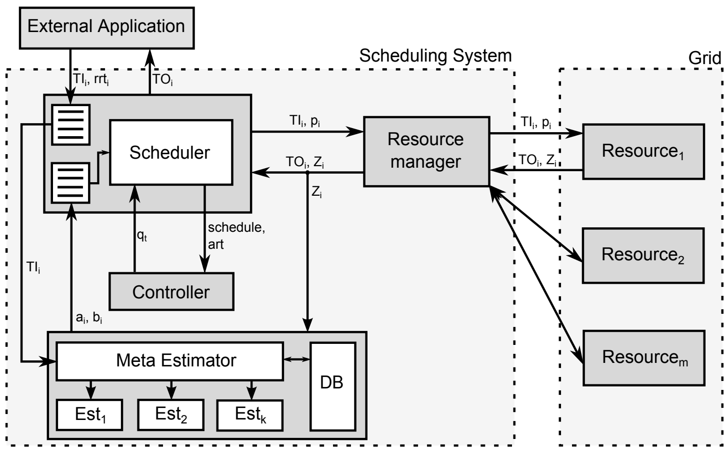


Fig. 1 Scheduling System Design

puters/CPU cores which executes the particular rostering algorithm. When the resource finishes processing the allocated task, it sends the solution (TO_i) together with computing progress (Z_i - progress of the objective function value over time) to the scheduling system. The solution is then sent to the corresponding external application and Z_i is stored in the estimator database for the online re-learning purposes.

The importance of the *controller* is to dynamically control the average response time (art) of the system through the q_t. This quality is dependent on the number of tasks to be processed and the amount of available resources.

3 Preliminary Results

For the preliminary experiments we consider STF+MCT (shortest task first + minimal completion time) scheduler, bisection controller [3] and six computer resources.

The behavior of the scheduling system with bisection controller and without controller on the same data is illustrated in Figure 2 and 3 respectively. The first subgraph in those figures shows the number of waiting tasks over time, the second one displays the average lateness, i.e. how many times the due date of tasks is exceeded, over time and finally the third subgraph indicates the quality set by the controller over time.

A comparison of Figure 2 and 3 shows significant improvement of the average response time of the system with the controller and the estimator. More experimental results and comparisons will be shown at the conference.

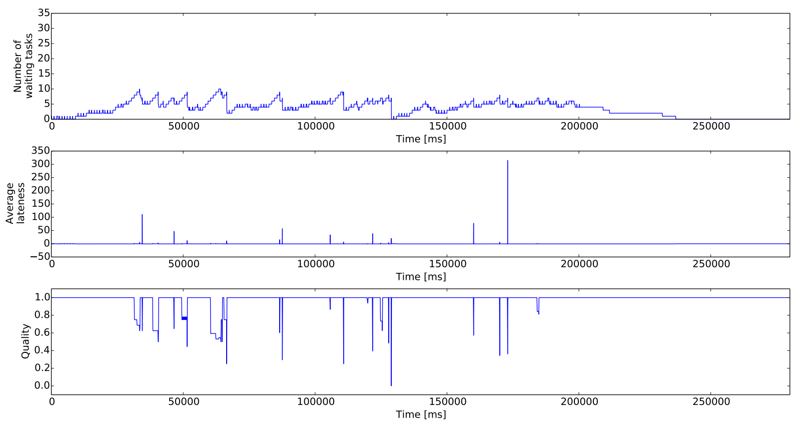


Fig. 2 The behavior of the scheduling system with bisection controller

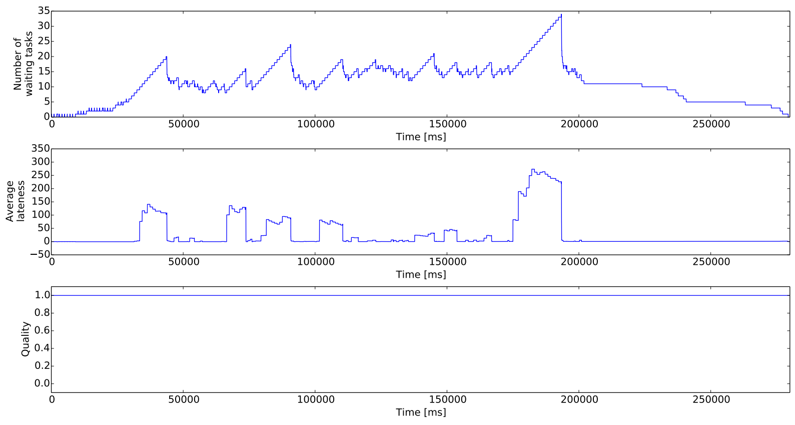


Fig. 3 The behavior of the scheduling system without controller (=constant quality)

References

1. E. K. Burke, P. De Causmaecker, G. Vanden Berghe and H. Van Landeghem, The State of the Art of Nurse Rostering, Journal of Scheduling, 7, 441-499 (2004)
2. R. Mangharam and A. A. Saba, Anytime Algorithms for GPU Architectures, IEEE 32nd Real-Time Systems Symposium, 47-56 (2011)
3. I. Módos, P. Šůcha, R. Václavík, J. Smejkal and Z. Hanzálek, Server Based Rostering Application, Czech Technical University in Prague, Internal technical report (2014)
4. G. Wan and J. Y.-T. Leung and M. L. Pinedo, Scheduling imprecise computation tasks on uniform processors, Information Processing Letters, 104, 45-52 (2007)
5. F. Xhafa and A. Abraham, Computational models and heuristic methods for Grid scheduling problems, Future Generation Computer Systems, 26, 608-621 (2010)

Optimal Duty Rostering for Toll Enforcement Inspectors

Elmar Swarat · **Guillaume Sagnol** ·
Thomas Schlechte

Abstract This abstract presents an Integer Programming based approach on optimal inspector rostering for the toll enforcement on German motorways.

Keywords Duty Rostering · Large Scale Integer Programming · Multi-Commodity Flow Problem

Mathematics Subject Classification (2000) 90B20 · 90C06

1 Abstract

We address the problem of planning optimal toll inspector tours on German motorways. Since 2005 a distance-based toll for all commercial trucks weighting 12 tonnes or above is set up on German motorways. A major part of the enforcement is conducted by tours of inspector teams on the toll network. Important enforcement goals are to take the spatial and temporal traffic distribution into account and to cover the complete network by unpredictable controls. Therefore, one task is to plan daily tours for the inspectors. In additon, a crew must be assigned to each tour, while each duty of a crew has to fit in a monthly (duty) roster. A feasible roster has to comply with a lot of different rules. Hence, a major challenge of our model is to optimize the rosters of the inspectors. This is the main focus of our presentation.

In other applications, e.g., in the railway or airline industry, the rostering is part of a multi-stage planning process. In particular, in duty scheduling tasks, e.g., timetabled trips, are covered by duties or pairings and afterwards rosters

This work was funded by the Federal Office for Goods Transport (BAG).

Elmar Swarat · Guillaume Sagnol · Thomas Schlechte
Zuse Institute Berlin, Takustr. 7, D-14195 Berlin
Tel.: +49-30-84185244
Fax: +49-30-84185269
E-mail: swarat@zib.de

are computed that cover all duties or pairings. In our setting a sequential planning of tours and crews is not appropriate according to the spatial distribution of the inspectors. Therefore, an integrated model of tour planning and duty rostering must be considered. We called this problem Toll Enforcement Problem (TEP). In [1] a case study presents the benefits of using our approach for the toll enforcement. Here, we extend our previous publications by a deeper look on the modeling and solving issues of the rostering part and by computational results that stem from production operation. The method can be transferred to other inspection or monitoring problems, where the availability of crews is an important issue for the planning problem.

Our approach is similar to the model, that Cappanera and Gallo [2] used to solve an airline crew rostering problem. Our model extends their approach in the respect that no activity has to be covered but there are coupling constraints that connect duties with tours. The base of our model is a directed graph $D = (V, A)$. The nodes $v \in V$ correspond to potential duties, that are defined as a triple $v = (d, b, e)$ of a day d, start time b, and end time e. Arcs $a = (v_1, v_2) \in A$ connect two duties $v_1 \in V$ and $v_2 \in V$, if v_2 is a feasible subsequent duty of v_1 according to legal rules. In addition, there are two nodes s und t indicating the beginning and end of the roster. Arcs $a = (s, v_i)$ connect s with all nodes v_i that might be the first duty of a roster. Analogously, arcs (v_i, t) connect potential last duties with t. Hence, a feasible duty roster corresponds to a s-t-path in D. We will discuss in detail, how several requirements can be modeled by this graph construction, like minimum rest times, days of leave or pre-assigned duties. The idea is to model as many constraints as possible as local decisions in our graph model.

The objective for the roster optimization is to minimize some artificial costs associated with unintended sequences of duties. Therefore, for each $a \in A$ artificial costs $c_a \geq 0$ are defined which have a non-zero value, if the current sequence of duties corresponds to *rotations*, i.e., changes in the duty starting time on two subsequent days. It is particularly known for rotations, where the following duty starts earlier, that they alter the human biorythms and affect the sleep. We model this optimization problem by a multi-commodity flow problem in D. For each inspector $m \in M$ we introduce binary flow variables $x_a^m \in \{0, 1\} \, \forall a \in A$. This results in the following Integer Program (IP):

$$\min \sum_{m \in M} \sum_{a \in A} c_a x_a^m \tag{1}$$

$$\sum_{v} x_{s,v}^m = 1, \qquad \forall m \in M, \tag{2}$$

$$\sum_{k} x_{v,k}^m - \sum_{u} x_{u,v}^m = 0, \qquad \forall v \in V, m \in M, \tag{3}$$

$$Rx \leq r, \tag{4}$$

$$x_a^m \in \{0, 1\}, \quad \forall a \in A, m \in M. \tag{5}$$

The objective function (1) minimizes the cost of the rosters. Constraints (2) and (3) represent the flow value and the flow conservation for the inspectors. Constraints (4) model additional requirements for feasible roster paths in D that we describe hereafter. In (5) the integrality constraints of the flow variables are given.

Each path consumes units of limited resources on its sequence arcs. In (4) this is coded by the matrix R and the limitation by the right hand side r. The resources deal with those requirements that can not be modeled locally in the graph. One example are *horizontal rules.* They involve single paths, like the working time consumption or limits on *unsocial working hours.* Other resources belong to *global rules* that involve all rosters. One example are *duty mix constraints* that limit the percentage of duties of a specified type, like weekend-duties. We will discuss some of these resources and their impact on the solvability of the TEP. Another important aspect will be the influence of input parameters on the size of D.

Furthermore, computational results from real-world instances complete our presentation. To solve the IP, we use the CPLEX solver by IBM. On a 8-core Intel Xeon workstation almost all instances occuring in daily operation can be solved either to optimality or with a small gap of at most 5%. One reason for the good performance is the quality of the lower bound produced by the linear relaxation. For most instances the gap between the initial lower bound and the optimal solution is distinctly lower than 10%. Since there are different control areas in Germany with different sizes and settings, some instances could be solved to optimality within 10 minutes while other require even more than two hours to find a feasible solution. We conclude that this approach is a successful example for the use of mathematical optimization techniques in real-world, since our method is implemented at the enforcement agency in Germany and part of the planning process for the enforcement.

References

1. Ralf Borndörfer and Guillaume Sagnol and Elmar Swarat, A Case Study on Optimizing Toll Enforcements on Motorways, 3rd Student Conference on Operational Research, OpenAccess Series in Informatics (OASIcs), 22, 1–10 (2012)
2. P. Cappanera and G. Gallo, A Multicommodity Flow Approach to the Crew Rostering Problem, Operations Research, 52 (4), 583–596 (2004)

Meta-heuristic algorithm for binary dynamic optimisation problems and its relevancy to timetabling

Ayad Mashaan Turky[1,2], Salwani Abdullah[1] and Nasser R. Sabar[3]
[1]Data Mining and Optimisation Research Group,
Centre for Artificial Intelligence Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
[2]Faculty of Information & Communication Technologies,
Swinburne University of Technology, Victoria 3122, Australia.
[3]The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih Selangor, Malaysia.
ayadalrashid@gmail.com, salwani@ukm.edu.my, Nasser.Sabar@nottingham.edu.my

1. Introduction

Many real world problems are dynamic in the sense that changes occur during the optimisation process.These problems are more convincing in real world applications than the static ones. This is due to the fact that most of the real world applications are dynamic as the problems differ in the changes that occur in the optimisation environment or the size of the problem increases from time to time [1, 2]. This phenomenon can be illustrated by the following example of a delivery company having to render a service to a set of customers where usually, the number of customers to be served changes on the service schedule due to the length of the contract period. Furthermore, the service that is demanded from the customer could also vary over time. This sort of situation could be considered to be a dynamic problem because the parameters would only be revealed during the delivery process where the number of customers or the demand of the product may increase or decrease.

Much effort has been made to solve dynamic optimisation problems over the recent decade [3]. In solving this problem, a solution method that is able to keep track of the changes is much needed. In addition the solution method should be adaptable in line with the current changes. In contrast to static optimisation problems (where the aim is to find the global optima), the goal of dynamic optimisation problems is to find not only the global optima but also to keep track of changes that usually occur during the optimisation process.

Generally, one could easily remark that the success of these algorithms is due to the incorporated mechanism that manages to maintain the population diversity when dealing with the changes [4]. Even though at present, there are a number of population-based methods applied on dynamic optimisation problems, there is still plenty of room for further research work, since the nature of this problem usually requires an efficient and effective algorithm that would quickly respond to changes.

Harmony search algorithm has been used to successfully solve a number of static optimization problems [5-8]. In this work, we investigate the applicability of the harmony search algorithm in tackling binary dynamic optimisation problems where four standard binary test functions are used. Based on the performance obtained, the proposed approach will later be employed on dynamic combinatorial optimisation problems such as dynamic vehicle routing problems and dynamic job shop scheduling problems.

2. Solution Approach

In this section, we present our proposed Harmony Search Algorithm (HSA) for solving binary dynamic optimisation problems. In this work, different mechanisms have been used to maintain the population diversity and their hybridisation with the HSA. This is referred to as Hybrid Harmony Search. HSA is one of the recent stochastic population-based meta-heuristic optimisation algorithms proposed by Geem et al. [9]. HSA has five steps as depicted in Fig. 1.

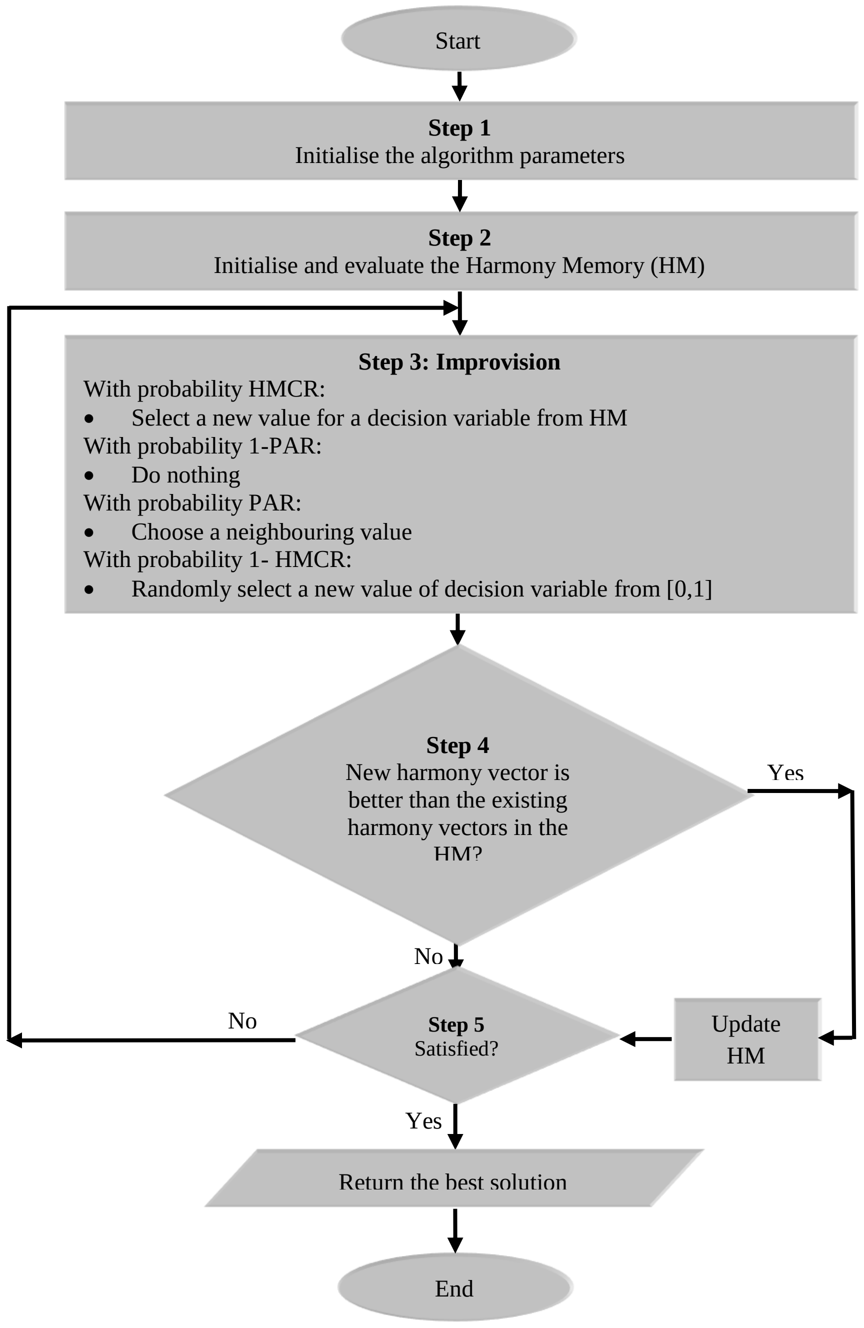

Start
Step 1
Initialise the algorithm parameters
Step 2
Initialise and evaluate the Harmony Memory (HM)
Step 3: Improvision
With probability HMCR:
• Select a new value for a decision variable from HM
With probability 1-PAR:
• Do nothing
With probability PAR:
• Choose a neighbouring value
With probability 1- HMCR:
• Randomly select a new value of decision variable from [0,1]
Step 4
New harmony vector is better than the existing harmony vectors in the HM?
Yes
No
Update HM
No
Step 5
Satisfied?
Yes
Return the best solution
End

Figure1: Steps in HSA

In Step 1, parameters of HSA (as in Table 1) are initialised. Harmony Memory (HM) is initialised and evaluated in Step 2. Step 3 is an improvisation process where Harmony Memory Consideration Rate (HMCR) parameter is used during the improvisation process in order to determine whether the value of a decision variable of a new harmony will be selected from the HM or will it be generated at random from the possible range that takes a value between [0, 1]. The probability of randomly selecting the decision variable value from the possible range is given as 1-HMCR. Pitch Adjusting Rate (PAR) parameter is used to decide either the values of decision variables (that have been selected from the HM) will be modified or maintained. PAR takes a value between [0, 1]. In Step 4, the HM will be updated and if the termination criterion is satisfied then the process will be terminated (Step 5).

In order to cope with the dynamic changes, harmony search algorithm needs to keep track of the changes during the search process. This is needed because the changes in the problem may change the current local optima into global optima and vice versa [1]. In addition, it is also shown in the literature that the developed algorithms for stationary problems cannot be directly used to solve dynamic problems [1, 10].

Therefore, to handle this problem, the HSA has been hybridised with three population diversity mechanisms, (i) HSA with random immigrant, HSA-I, (ii) HSA with memory mechanism, HSA-M, and (iii) HSA with memory based immigrant mechanism, HSA-MI.

- HSA-I: First mechanism where HSA is hybridised with random immigrant. In this approach, at each of the generation a subset of solutions is generated at random and is used to replace the worst solutions in the HM. In this paper, the number of solutions are fixed to be replaced at every iteration as *rs*=HMS*0.2 where *rs* represents the number of replaced solutions.

- **HSA-M:** Second mechanism where HSA is hybridised with a memory mechanism. In this approach, a subset of best solutions is kept and will be re-inserted in the HM once changes are detected.

- **HSA-MI:** In the third mechanism where HSA is hybridised with a random immigrant and a memory based mechanism in order to maintain the diversity of HM.

3. Results and Discussions

The performance of the proposed approaches is verified on four well-known binary dynamic optimisation test functions i.e. OneMax, Plateau, Royal Road, and Deceptive. The parameter values of HSA which is based on our preliminary tests are presented in Table 1.

Table 1 HSA parameter values

Parameters	***Description***	***Tested range***	***Suggested value***
HMS	Harmony memory size HMS= 1 to 100	10-200	100

HMCR	Harmony memory consideration rate... $0 < HMCR < 1$	0.1-0.99	0.6
RCR	Random consideration rate	-	RCR=1-HMCR
PAR	Pitch adjustment rate $0 < PAR < 1$	0.1-0.99	0.3
NI	Number of improvisations or iterations	-	500000 function evaluations

Our hybridisation approaches are compared against the well-known methods in the literature. The algorithms in comparison are presented in Table 2.

Table 2 Acronyms of compared methods

#	*Symbol*	*References*
1	MIGA	[6]
2	MEGA	[7]
3	AHMA	[1]
4	MRIGA	[6]

In order to measure the performance of our proposed algorithm the overall offline performance (the best-of-generation fitness) is calculated over 30 runs (with different initial solutions and seeds) based maximisation of Eq. 1.

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} F_{BOG_{ij}}\right) \quad (1)$$

where G is the total number of generations, N is the total number of runs and F_{BOGij} is the best of generation fitness of generation i of run j. Our results as well as other methods in comparison are presented in Table 3.

Table 3: Comparison of Results

Function name	**HSA-I**	**HSA-M**	**HSA-MI**	**% Deviation**	**MIGA**	**MEGA**	**AHMA**	**MRIGA**
OneMax	91.67	90.42	**96.01**	**	94.0	79.3	95.89	80.8
Plateau	72.21	68.41	**84.91**	**	-	-	62.88	-
Royal Road	64.76	63.96	**66.19**	**	-	-	52.52	-
Deceptive	76.39	73.11	**85.97**	**	71.1	83.1	85.75	68.6

'-': no results are reported. '**': our algorithm is better than others.

As shown the in Table 3, HSA-MI outperforms other methods in all test functions (presented in bold), in which we believe this is due to the idea of of hybridising a random immigrant and a memory based mechanism in order to maintain the harmony memory (population) diversity.

4. Dynamic Optimisation Problems and Its Relevancy to Timetabling Problems

Dynamic optimisation problems however, present a greater challenge to the research community since the problem parameters are either revealed or change during the course of the on-going optimization [4]. These problems are more convincing in real world applications than the static ones. This is due to the fact that most of the real world applications are dynamic, as the problems in the sense that the environment is subjected to changes or the size of the problem increases from time to time [11].

Timetabling problems have been frequently studied because of their wide range of applications such as school timetabling, transport scheduling, job shop scheduling, vehicle routing, and patient admission scheduling problems. In timetabling problems, two main difficulties are encountered: (i) often over-constrained and optimisation criteria are hard to define; (ii) intrinsically dynamic where activities, resources or constraints are sometimes unknown or can often change at the last moment [12].

The relevancy between dynamic optimisation problems with timetabling problems can be expressed through examples on:

- School timetabling: the dynamic part of the schedule is more related to logistic needs and unexpected events such as link with timetabling of other years in terms sharing common resources and, inside and outside teachers availability [12].
- Train scheduling: the information such as train arrival times, train lengths, train speeds are available before solving the problem in the static scheduling environment. However, in dynamic scheduling environment (which mimics the real-world problem), the information of only arrived trains is considered known, then the schedule of the new train and the trains currently in the network should be generated, given no information of later trains [13].
- Job shop scheduling: most manufacturing systems operate in dynamic environment where unexpected disruptions occur during the manufacturing process such as machine breakdowns, material shortage, random job releases, and job cancellations and due date and time processing changes. The disruption will produce uncertainty in the sequence of operation, i.e. the time taken to repair the broken machine [14].
- Dynamic vehicle routing: dynamic scenarios have become more common in vehicle routing. The most common source of dynamism in vehicle routing is the online arrival of customer *requests* (demand for goods and services) during the operation, dynamic travel and service time and vehicle availability, and breakdown of vehicles. These source of dynamism caused schedulers to update the generated timetable [15, 16].
- Patient admission scheduling: it concerns in assigning patients to bed in a hospital. In order to tailor real scenario, several real-world features, such as the presence of emergency patients, uncertainty in the length of stay, and the possibility of delayed admissions are included [17].

The above examples show the important on tackling dynamic optimisation problems since most of the real world problems are dynamic in nature. These disruptions or random occurrences lead timetable officers to develop a new schedule from scratch or reschedule the existing schedule in order to cater the changes.

5. Conclusion and Future Work

The overall goal of the work presented in this paper is to investigate the performance of the hybrid harmony search algorithm in maintaining the population diversity in addressing binary dynamic optimisation problems. In this work, three kinds of population diversity mechanisms are presented i.e. the random immigrant, memory mechanism, and memory based immigrant mechanism. Initial experiments show that the harmony search with memory based immigrant mechanism outperforms two hybrid approaches presented here, and also managed to obtain better offline performance in comparison to other available approaches in the literature. For future work, the proposed method will be investigated on other dynamic combinatorial optimization problems such as dynamic vehicle routing and dynamic job shop scheduling problems.

References

[1] H. Wang, D. Wang, and S. Yang, "A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems," *Soft Computing,* vol. 13, pp. 763-780, 2009.

[2] E.-G. Talbi, "A taxonomy of hybrid metaheuristics," *Journal of heuristics,* vol. 8, pp. 541-564, 2002.

[3] S. Yang, Y. S. Ong, and Y. Jin, *Evolutionary computation in dynamic and uncertain environments* vol. 14: Springer, 2007.

[4] S. Yang, "Non-stationary problem optimization using the primal-dual genetic algorithm," in *The 2003 Congress on Evolutionary Computation. CEC'03.* , 2003, pp. 2246-2253.

[5] M. Hadwan, M. Ayob, N. R. Sabar, and R. Qu, "A harmony search algorithm for nurse rostering problems," *Information Sciences,* vol. 233, pp. 126-140, 2013.

[6] A. M. Turky and S. Abdullah, "A multi-population harmony search algorithm with external archive for dynamic optimization problems," *Information Sciences,* vol. 272, pp. 84-95, 2014.

[7] N. R. Sabar and G. Kendall, "Using Harmony Search with Multiple Pitch Adjustment Operators for the Portfolio Selection Problem.," *2014 IEEE Congress on Evolutionary Computation (IEEE CEC 2014), 6-11 July 2014, Beijing, China.,* 2014.

[8] A. M. Turky, S. Abdullah, and N. R. Sabar, "A Hybrid Harmony Search Algorithm for Solving Dynamic Optimisation Problems.," *The International Conference on Computational Science-ICCS, 10-12 June 2014, Cairns, Australia.,* 2014.

[9] Z. W. Geem, J. H. Kim, and G. Loganathan, "A new heuristic optimization algorithm: harmony search," *Simulation,* vol. 76, pp. 60-68, 2001.

[10] S. Yang, "Genetic algorithms with memory-and elitism-based immigrants in dynamic environments," *Evolutionary Computation,* vol. 16, pp. 385-416, 2008.

[11] J. Branke and H. Schmeck, "Designing evolutionary algorithms for dynamic optimization problems," in *Advances in evolutionary computing,* ed: Springer, 2003, pp. 239-262.

[12] H. Cambazard, F. Demazeau, N. Jussien, and P. David, "Interactively solving school timetabling problems using extensions of constraint programming," in *Practice and Theory of Automated Timetabling V,* ed: Springer, 2005, pp. 190-207.

[13] J. Rodriguez, "A constraint programming model for real-time train scheduling at junctions," *Transportation Research Part B: Methodological,* vol. 41, pp. 231-245, 2007.

[14] G. Chryssolouris and V. Subramaniam, "Dynamic scheduling of manufacturing job shops using genetic algorithms," *Journal of Intelligent Manufacturing,* vol. 12, pp. 281-293, 2001.

[15] A. Haghani and S. Jung, "A dynamic vehicle routing problem with time-dependent travel times," *Computers & operations research,* vol. 32, pp. 2959-2986, 2005.

[16] H. N. Psaraftis, "Dynamic vehicle routing: Status and prospects," *annals of Operations Research,* vol. 61, pp. 143-164, 1995.

[17] S. Ceschia and A. Schaerf, "Modeling and solving the dynamic patient admission scheduling problem under uncertainty," *Artificial intelligence in medicine,* vol. 56, pp. 199-205, 2012.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

From sales data to workforce schedules

Forecasting, workload modeling, shift scheduling and shift rostering

Egbert van der Veen

Abstract We discuss a commercially implemented workforce scheduling approach by going through a real-life application in retail stores. This approach takes as input historical 15-minute sales data and translates this into optimized workforce schedules by going through a number of steps. The approach starts by forecasting future sales from the historical sales. After that, the forecasted sales data is translated into staffing requirements, expressing the number of persons that need to be scheduled in each 15 minute time slot of a specified horizon. This is then input to generate a set of shifts that covers these staffing requirements, and finally a workforce schedule is generated by assigning employees to these shifts.

Keywords Forecasting · Workload modeling · Shift scheduling · Shift rostering

1 Context

In supermarkets, the work that has to be performed is mainly driven by grocery sales. Products can be sold only if the cash desks are staffed, and if shelves are stocked at the right moment and time. To make sure the right person is available at the right time, we have developed a scheduling approach that uses historical sales data as input and converts this to optimized workforce schedules. Our scheduling approach consists of multiple stages, which we describe in the next section.

E. van der Veen
ORTEC, Houtsingel 5, 2719 EA, Zoetermeer, the Netherlands
Tel.: +31886783265
E-mail: Egbert.vanderVeen@ortec.com

2 Approach

Our approach consists of five stages: forecasting, workload modeling, capacity planning, shift scheduling and shift rostering. The capacity planning step is an optional step in this approach and can thus be skipped. Comparable decompositions are found in Burke et al (2004), Ernst et al (2004) and Thompson (1997).

1. Forecasting First, we sum the 15-minute historical sales to daily totals. From the day totals future day total sales are forecasted. To convert the forecasted day totals to 15-minute sales data, we use 'day patterns' that specify how sales are distributed throughout the day. The motivation for this decomposition is that sales patterns throughout the day may differ substantially, whereas day totals tend to be more stable, thereby allowing for more accurate day totals forecasts. Forecasts are calculated using various forecasting algorithms that are combined in a regression model. The forecasting algorithms we use are, among others, Holt-Winters and Exponential Smoothening, see, e.g., Gelper et al (2010).

2. Workload modeling This stage determines the staffing requirements, which express the number of people that should be staffed on a certain activity in a specific 15-minute time slot. Using productivity figures, sales forecasts are translated into staffing requirements. For example, if 500 euros is forecasted for some 15-minute time slot, and the productivity at the cash desk is 250 euros per cashier per 15 minutes, then 2 cashiers have to be staffed.

Depending on the level of detail, various forecasts can be calculated to determine staffing requirements for various activities. For example, to determine the number of cashiers, a forecast on the total sales is applicable, but to determine the number of people stocking the dairy products, a forecast for the dairy sales should be available.

3. Capacity planning From the staffing requirements, a mid-term capacity plan can be made, describing target working hours per employee per week. This is useful, since in practice one week may be busier than the other. Therefore, it makes sense to let people work more hours in one week than in the other. This makes it possible to do more with the same people, since in slow weeks there is less over-capacity and in busy weeks there is less under-capacity, reducing the need to hire (expensive) subcontractors. Using lower and upper bounds on the working hours of individual employees combined with absence information of employees, working hour targets are determined per week per employee, such that the workforce capacity is distributed optimally over the weeks. The modeling details are found in Van der Veen et al (2012). Note that in our approach, the capacity planning step is optional and can be skipped.

4. Shift scheduling After staffing requirements have been determined, shifts are scheduled. Shift scheduling is subject to constraints on, e.g., shift lengths, shift starting times, shift ending times, break rules, and allowed activity combinations. Using staffing requirements and shift generation rules as input, a set of shifts is determined, such that the staffing requirements are covered efficiently, hard constraints are respected and violations of soft constraints are minimized. The set of shifts is optimized by a metaheuristic framework that combines various re-start heuristics with variable neighborhood search.

5. Shift rostering After shift scheduling is complete, employees have to be assigned to the shifts. Shift rostering is subject to labor legislation constraints on, e.g., allowed sequences and combinations of shifts. Moreover, unavailability of employees, employees preferences and working hour targets (as determined in the capacity planning step) are considered in shift rostering. Our shift rostering algorithm aims to assign all shifts to employees, while minimizing violations of soft constraints. Of course, hard constraints, such as labor legislation, may not be violated by the algorithm.

In short, this algorithm employs a hybrid heuristic ordering method to construct multiple initial shift schedules, which are improved by a genetic algorithm. Next, the best schedule found by the genetic algorithm is improved using a variable neighborhood search. Mathematical details of this algorithm are found in Burke et al (2008) and Post and Veltman (2004).

3 Conclusions

We have designed a process that creates demand driven workforce schedules using historical sales data. Hereby, the aim is to have people available if needed and when needed. The next steps in our research and implementation are to analyze the effects on cost reduction, profit increases, and service level improvements. Moreover, there are interesting future research questions such as combining the shift scheduling and rostering steps, and assessing the feasibility of our approach for other industries.

References

Burke EK, de Causmaecker P, vanden Berghe G, van Landeghem H (2004) The state of the art of nurse rostering. Journal of Scheduling 7(6):441–499

Burke EK, Curtois T, Post G, Qu R, Veltman B (2008) A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European Journal of Operational Research 188(2):330–341

Ernst A, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1):3–27

Gelper S, Fried R, Croux C (2010) Robust forecasting with exponential and holt-winters smoothing. Journal of Forecasting 29(3):285–300

Post G, Veltman B (2004) Harmonious personnel scheduling. In: Proceedings of the 5th international conference on the Practice and Theory of Automated Timetabling, pp 557–559

Thompson GM (1997) Labor staffing and scheduling models for controlling service levels. Naval Research Logistics (NRL) 44(8):719–740

Van der Veen E, Hans EW, Veltman B, Berrevoets LM, Berden HJ (2012) Cost-efficient staffing under annualized hours. Tech. Rep. 1995, Enschede, the Netherlands, URL http://doc.utwente.nl/84363/

System Demonstrations

Do it yourself (DIY) optimisation approach to practical timetabling

Yuri Bykov, Sanja Petrovic, Christos Braziotis

Nottingham University Business School

Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK

yuri.bykov, sanja.petrovic, christos.braziotis@nottingham.ac.uk

This abstract describes our work in progress towards facilitating a greater uptake of metaheuristic optimisation algorithms in practice. Many researchers and practitioners have recognised that there is still a considerable gap between theory and practice in metaheuristic optimisation. Although this gap exists in many application areas, the educational timetabling is the field where it was clearly formulated (see McCollum 2007).

One of the causes of this gap is the inflexibility of the existent timetabling applications, i.e. they cannot enfold the high variety of real-world requirements and restrictions necessary for a good timetable. A survey by Burke et al. (1996) revealed that students and administrative preferences vary greatly among universities. This means that a computer-aided timetabling system developed in one educational institution is unsuitable for another one. Therefore, a common solution here is to order made-to-measure timetabling systems separately tailored for each particular university. This becomes very expensive, time-consuming and inflexible approach, especially when some alterations have to be included into existing systems. As an alternative to that, some universities develop in-house timetabling systems. However, this solution requires a capacity for programming skills by timetabling department staff.

In this study we propose a third variant, which constitutes a middle ground between the above-mentioned extremes. It can be viewed as DIY (do it yourself) optimisation approach because it is more flexible than the first approach but requires fewer user's skills than the second one. The main idea of the approach is based on the two following observations.

Our first observation is that almost in every timetabling (as well as scheduling, rostering, etc.) research paper we can find a formal definition of a problem as a set of *formulas*. For example, the well-known Carter's formulation of the Exam Timetabling Problem where objective function refers to the proximity of exams (see Carter et al. 1996) is expressed in a mathematical model as follows (taken from Burke et al. 2004):

$$\text{minimize: } \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij} \times proximity(t_i t_j)$$

$$\text{where: } proximity(t_i t_j) = \begin{cases} 2^{5-|t_i - t_j|} & if\ 1 \le |t_i - t_j| \le 5 \\ 0 & otherwise \end{cases}$$

$$\text{subject to clash-free requirement: } \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij} \times clash(t_i t_j) = 0$$

$$\text{where: } clash(t_i t_j) = \begin{cases} 1 & if\ t_i = t_j \\ 0 & otherwise \end{cases}$$

In these formulas N is the number of exams, c_{ij} are the elements of NxN conflict matrix, which indicate the numbers of students sitting exams i and j together and t_i is the timeslot of ith exam. However, in most cases such formulas are given for reference only and are not used *explicitly* in the supplementary software (experimental or end-user ones). They just formally describe the rules, which are implemented in the form of an *algorithm*, which de-facto represents a more complex procedure than just few formulas. Our second observation is that in other areas there exists software that explicitly operates with formulas for different purposes, such as Matlab for mathematical operations, CPLEX for integer programming or Microsoft Excel for tabbed calculations. By combining these two observations we came up with an idea of the explicit use of mathematical formulas in metaheuristic optimisation as well. It should be noted that the generic idea of embedding a CPLEX-like functionality for increasing the flexibility of metaheuristics is rather straightforward and was circulated in private communications throughout the research community. Therefore, the novelty of our particular contribution to that idea is its effective practical implementation.

To implement this idea in practice a compiler-like intelligent engine was developed, which recognises mathematical notations for the cost function and constraints, verifies them and then prepares them for the use within metaheuristic search algorithm. The formulas can be entered in a quite transparent machine-readable form, which follows common rules used in existing systems (e.g. MS Excel). The detailed description of the rules can be downloaded together with our software. For example, the formulas for the cost function presented above can be entered as follows:

```
cost : sum[i,1,n-1,sum(j,i+1,n,c(i,j)*proximity)]
proximity : when[(1<=absdt) and (absdt<=5),pow(2,5-absdt),0]
absdt : abs[t(i)-t(j)]
```

The corresponding constraints can have the following form:

```
constraint : sum[i,1,n-1,sum(j,i+1,n,c(i,j)*clash)]
clash : when[t(i)=t(j),1,0]
```

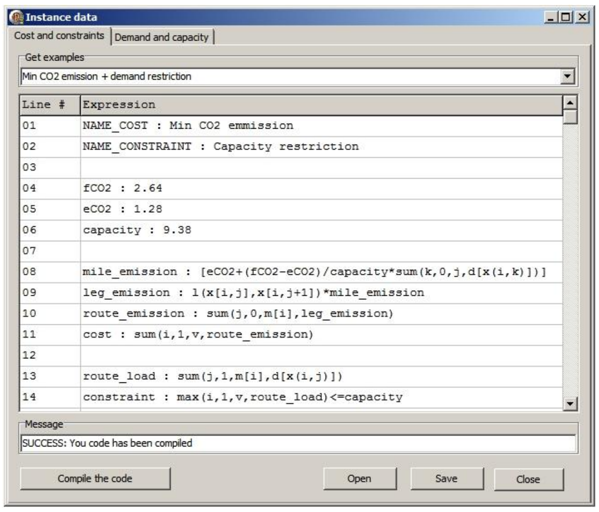
As a result, the user needs just to write five formulas to solve a timetabling problem with Carter’s cost/constraint. Moreover, the user is free to change them or enter completely new ones according to his/her own requirements. If, for example, additionally to the clash-free requirement we need to schedule exam #1 before exam #2 and the number of timeslots should not be more than 15, then the constraint definition could be re-written as:

```
constraint : clash_free and [t(1)<t(2)] and (max[i,1,n,t(i)]<=15)
clash_free : sum[i,1,n-1,sum(j,i+1,n,c(i,j)*clash)]
clash : when[t(i)=t(j),1,0]
```

Our engine recognises virtually any type of cost and constraints, which can be expressed by formulas. This way of cost/constraints specification is much simpler than in-house programming, while at the same time is much more flexible than the ordering of made-to-measure software.

In our particular implementation the engine represents a run-time library, which can be embedded into any optimisation algorithm, not necessary only for timetabling problems. In each case, the engine should be adjusted to the particular problem and solution representations. In the above example (uncapacitated exam timetabling problem), the compiler recognises variables **n** and **c** as the elements of the problem statement and variable **t** as the element of solution. However, if one would like to solve the capacitated problem, the engine should be adjusted in order to recognise the room-related variables.

In order to demonstrate the simplicity and flexibility of the proposed approach and to prove that it is workable we have embedded the prototype cost/constraints compiler into our Vehicle Routing Problem (VRP) solver (the choice of a problem was motivated by a good visual characteristics of VRP solutions, but we anticipate the variant for Exam Timetabling to be ready soon). Figure 1 illustrates a built-in formula editor where the user can enter or change his/her formulas. After pressing the button "Compile the code" the system produces error checks and if no errors are found, incorporates these formulas to the search procedure. In this example, the cost formula represents an amount of CO_2 emissions, but certainly, the user can enter here an unlimited number of possible cost functions.



Get examples: Min CO2 emission + demand restriction

Line #	Expression
01	NAME_COST : Min CO2 emmission
02	NAME_CONSTRAINT : Capacity restriction
03	
04	fCO2 : 2.64
05	eCO2 : 1.28
06	capacity : 9.38
07	
08	mile_emission : [eCO2+(fCO2-eCO2)/capacity*sum(k,0,j,d[x(i,k)])]
09	leg_emission : l(x[i,j],x[i,j+1])*mile_emission
10	route_emission : sum(j,0,m[i],leg_emission)
11	cost : sum(i,1,v,route_emission)
12	
13	route_load : sum(j,1,m[i],d[x(i,j)])
14	constraint : max(i,1,v,route_load)<=capacity

Message: SUCCESS: You code has been compiled

Fig 1. The cost/constraints entered as formulas

We expect that the variation of the cost/constraints definition could affect the performance of a core optimization technique: therefore the solver offers to choose the preferable one among 6 available metaheuristics: Hill-Climbing, Simulated Annealing, Tabu Search, Great Deluge Algorithm, Late Acceptance Hill-Climbing and Step Counting Hill Climbing (the details of the last two ones can be found in (Burke and Bykov 2008) and (Bykov and Petrovic 2013)). Also the user can select appropriate algorithmic parameters, initialization, restarting/reheating strategies as well as the type of moves (neighborhood operators).

The prototype solver can be downloaded from: http://www.yuribykov.com/MHsolver/. It works in MS Windows with a minimum hardware configuration and does not require an installation procedure.

In order to test the practical effectiveness of the proposed approach, we used our solver as a part of the coursework for the undergraduate module Management Science for Business Decisions at the Nottingham University Business School. This served as a pilot study to assess the usability of our approach in solving the given VRP problems and it involved 10 students without programming skills and with different mathematical background. Among other tasks the students were asked to solve the variants with known and unknown formulas for cost/constraints. The results revealed that all students were able to transform the known formulas to the machine-readable form and enter them into the system. However, some students were not able to solve a more complex task where the formulas are not given but cost/constraints are specified in a textual description. Here 6 out of the 10 students were able to derive a correct formula. These preliminary results suggest that a particular level of mathematical skill/experience is required for the successful use of our approach. Thus, we see the development of a proper training methodology as an important direction of increasing the practical value of our approach.

References:

Burke, E.K., D. Elliman, P. Ford, R. Weare. 1996. Examination timetabling in British universities: a survey. Practice and Theory of Automated Timetabling, Springer Lecture Notes in Computer Science 1153, 76-90.

Burke, E.K., Y. Bykov, J. Newall, S. Petrovic. 2004. A time-predefined local search approach to exam timetabling problems. IIE Transactions 36, 509-528.

Burke, E.K., Y. Bykov. 2008. A late acceptance strategy in hill-climbing for exam timetabling problems. Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling PATAT 2008, Montreal, Canada, August 2008.

Bykov, Y., S. Petrovic. 2013. An initial study of a novel Step Counting Hill Climbing heuristic applied to timetabling problems. Proceedings of the 6th Multidisciplinary International Scheduling Conference MISTA 2013, Gent, Belgium, August 2013.

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

McCollum, B. 2007. A perspective on bridging the gap between theory and practice in university timetabling. Practice and Theory of Automated Timetabling VI, Springer Lecture Notes in Computer Science 3867, 3-23.

Meeting Rural Transport Needs through Demand Responsive Transport Scheduling (Bwcabus)

Clark, Owen
University of South Wales
+44 (0)1443 654047
Owen.clark@southwales.ac.uk

Dr Olden, Andrew
University of South Wales
+44 (0)1443 483613
Drew.Olden@southwales.ac.uk

Keywords: Timetabling in Transport, Demand Responsive Transport, Complex Evolving Systems, Heuristics, Artificial Intelligence, Simulated Annealing.

Introduction

The following system demonstration presents an approach to demand responsive transport (DRT) that has been developed and is currently being used in the real world environment of West Wales. The system has been developed by the University of South Wales in conjunction with the local government organisations covering the test area, namely Carmarthenshire and Ceredigion County Councils. Other project partners include the Welsh national public transport information organisation Traveline Cymru and bus operators. The demonstration will introduce Bwcabus as a case study, describe the scheduling system and identify the system challenges associated with managing passenger demand and expectations.

What is Bwcabus?

The traditional models of public transport delivery (based on fixed timetables and routes) can fail to meet the needs of passengers in rural areas because services can be too infrequent and inflexible. It is suggested that Demand Responsive

Transport (DRT) can be used to address social exclusion and rural accessibility by providing a more flexible and customer responsive service.

Bwcabus is a DRT service covering rural Carmarthenshire and Ceredigion. The service commenced in August 2009 funded by the Welsh Government, the European Convergence Fund and Carmarthenshire and Ceredigion County Councils.

DRT are services that provide transport on demand, scheduled to pick up and drop off passengers in accordance with their needs. Bwcabus is therefore a 'hybrid', falling somewhere between a conventional timetabled bus service and a taxi (Gerrard, 1974). A DRT timetable is not fixed and will vary each day. This form of 'dynamic' scheduling allows passengers greater flexibility to book journeys at the times (or close to the times) they require.

Bwcabus is integrated with strategic public transport services, providing connections at designated hubs. Communications technologies are deployed to maximise the efficiency of the service and ensure connections are guaranteed. Therefore Bwcabus facilitates a large number of journey options between the fixed and demand responsive services. A similar scenario is presented in Hall et. al. (2009).

The System

At the heart of the Bwcabus operation is the scheduling system. The complete system includes journey scheduling, booking management and public transport information import and management.

The scheduling system is based upon the selection of either combinetrics or Simulated Annealing (Baugh et. al, 1998; Uchimura et al 2002) depending on the number of unique locations visited on a trip. Where a limited number of locations (less than five) are visited it has been demonstrated the optimal methodology is the use of combinetrics, that is to say the generation of every single journey permutation. At larger numbers of locations Simulated Annealing becomes optimal. The simulated annealing parameters vary in line with the number of locations visited.

The following section presents a high-level overview of the system operation:

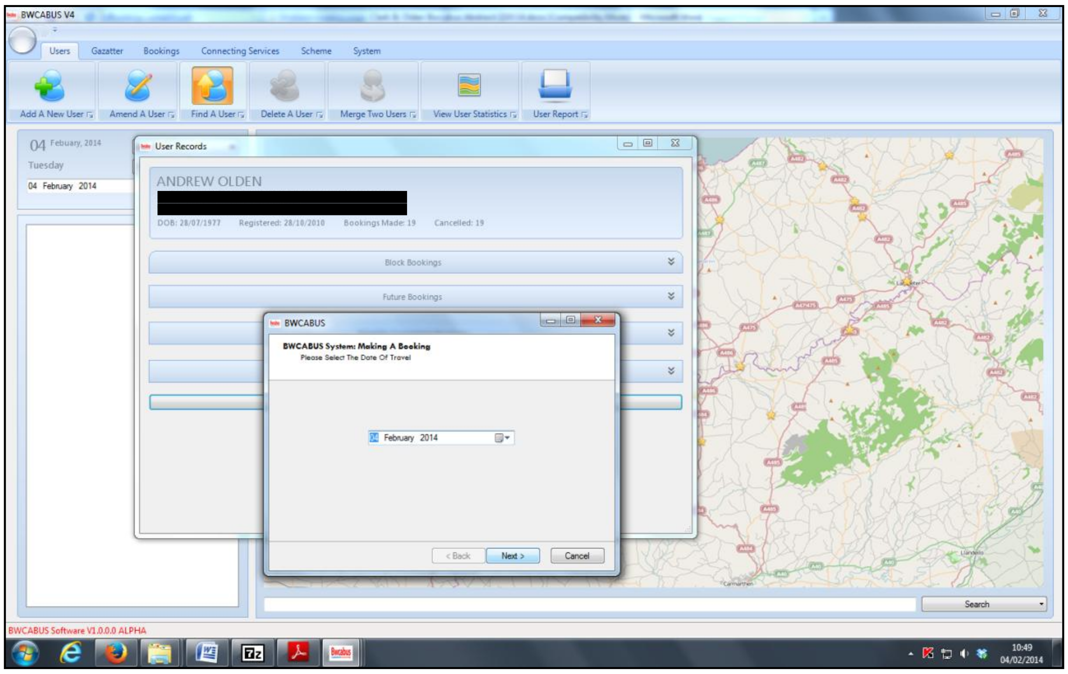
- Load all bookings from system database for required day
- Split bookings into groups or journeys
  - Based upon the start and perceived end times of each booking in the system, any bookings running concurrently are grouped together.
- ***For each journey determine optimal journey pattern a (s1)***
  - If unique location <=5 Use Combinetrics
  - If unique locations > 5 Use Simulated Annealing
    - If unique locations > 8 decrease cooling temperature
- ***Acquire required journey information (s2)***
- determine optimal journey pattern **b** (including new locations)
  - If unique location <=5 Use Combinetrics
  - If unique locations > 5 Use Simulated Annealing
    - If unique locations > 8 decrease cooling temperature
- Validate optimal journey pattern **b** based on effects on pattern **a**
  - If **b** is valid add to list
- Check if journey can be made as a standalone trip, separate to others using journey pattern **a** as a constraint model.
  - If possible Add to list

The highlighted sections (***s1*** and ***s2***) in the high-level overview may be operated independently of each other. That is to say when acquiring information required to make a new booking the system is optimizing an existing days journeys using multi threading. By the time the acquisition process is complete the schedule for that day will have been optimised ready to attempt the integration of the new booking.

The system operates in a number of modes, dependant on the end users and uses Web Services to enable the interrogation of a central database (located in South East Wales). Where the user is a ‘scheme manager’ located in the local authority

(West Wales) the system enables the modification of existing bookings, such as swapping the bus a booking takes place on, or the time, location and number of people travelling. Call centre users (located in North Wales) who take requests for bookings from end users face a wizard based interface as shown in Figure 1.

Figure 1 – Call Centre User Interface



04 Febuary, 2014
Tuesday
04 February 2014

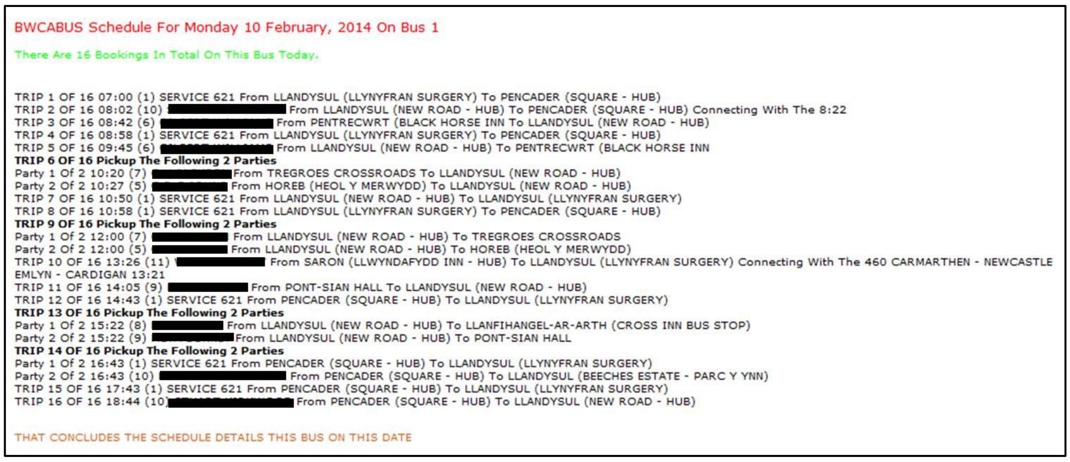
User Records
ANDREW OLDEN
[redacted]
DOB: 28/07/1977 Registered: 28/10/2010 Bookings Made: 19 Cancelled: 19

BWCABUS
BWCABUS System: Making A Booking
Please Select The Date Of Travel
04 February 2014

BWCABUS Software V1.0.0.0 ALPHA

Details of the journeys a bus is required to make can also be viewed via a web page, as shown in Figure 2. Mobile communication technologies are used to send the details of the schedule directly to each bus twice a day.

Figure 2 – Schedule Web View



```
BWCABUS Schedule For Monday 10 February, 2014 On Bus 1

There Are 16 Bookings In Total On This Bus Today.

TRIP 1 OF 16 07:00 (1) SERVICE 621 From LLANDYSUL (LLYNYFRAN SURGERY) To PENCADER (SQUARE - HUB)
TRIP 2 OF 16 08:02 (10) [redacted] From LLANDYSUL (NEW ROAD - HUB) To PENCADER (SQUARE - HUB) Connecting With The 8:22
TRIP 3 OF 16 08:42 (6) [redacted] From PENTRECWRT (BLACK HORSE INN To LLANDYSUL (NEW ROAD - HUB)
TRIP 4 OF 16 08:58 (1) SERVICE 621 From LLANDYSUL (LLYNYFRAN SURGERY) To PENCADER (SQUARE - HUB)
TRIP 5 OF 16 09:45 (6) [redacted] From LLANDYSUL (NEW ROAD - HUB) To PENTRECWRT (BLACK HORSE INN
TRIP 6 OF 16 Pickup The Following 2 Parties
Party 1 Of 2 10:20 (7) [redacted] From TREGROES CROSSROADS To LLANDYSUL (NEW ROAD - HUB)
Party 2 Of 2 10:27 (5) [redacted] From HOREB (HEOL Y MERWYDD) To LLANDYSUL (NEW ROAD - HUB)
TRIP 7 OF 16 10:50 (1) SERVICE 621 From LLANDYSUL (NEW ROAD - HUB) To LLANDYSUL (LLYNYFRAN SURGERY)
TRIP 8 OF 16 10:58 (1) SERVICE 621 From LLANDYSUL (LLYNYFRAN SURGERY) To PENCADER (SQUARE - HUB)
TRIP 9 OF 16 Pickup The Following 2 Parties
Party 1 Of 2 12:00 (7) [redacted] From LLANDYSUL (NEW ROAD - HUB) To TREGROES CROSSROADS
Party 2 Of 2 12:00 (5) [redacted] From LLANDYSUL (NEW ROAD - HUB) To HOREB (HEOL Y MERWYDD)
TRIP 10 OF 16 13:26 (11) [redacted] From SARON (LLWYNDAFYDD INN - HUB) To LLANDYSUL (LLYNYFRAN SURGERY) Connecting With The 460 CARMARTHEN - NEWCASTLE
EMLYN - CARDIGAN 13:21
TRIP 11 OF 16 14:05 (9) [redacted] From PONT-SIAN HALL To LLANDYSUL (NEW ROAD - HUB)
TRIP 12 OF 16 14:43 (1) SERVICE 621 From PENCADER (SQUARE - HUB) To LLANDYSUL (LLYNYFRAN SURGERY)
TRIP 13 OF 16 Pickup The Following 2 Parties
Party 1 Of 2 15:22 (8) [redacted] From LLANDYSUL (NEW ROAD - HUB) To LLANFIHANGEL-AR-ARTH (CROSS INN BUS STOP)
Party 2 Of 2 15:22 (9) [redacted] From LLANDYSUL (NEW ROAD - HUB) To PONT-SIAN HALL
TRIP 14 OF 16 Pickup The Following 2 Parties
Party 1 Of 2 16:43 (1) SERVICE 621 From PENCADER (SQUARE - HUB) To LLANDYSUL (LLYNYFRAN SURGERY)
Party 2 Of 2 16:43 (10) [redacted] From PENCADER (SQUARE - HUB) To LLANDYSUL (BEECHES ESTATE - PARC Y YNN)
TRIP 15 OF 16 17:43 (1) SERVICE 621 From PENCADER (SQUARE - HUB) To LLANDYSUL (LLYNYFRAN SURGERY)
TRIP 16 OF 16 18:44 (10) [redacted] From PENCADER (SQUARE - HUB) To LLANDYSUL (NEW ROAD - HUB)

THAT CONCLUDES THE SCHEDULE DETAILS THIS BUS ON THIS DATE
```

Results

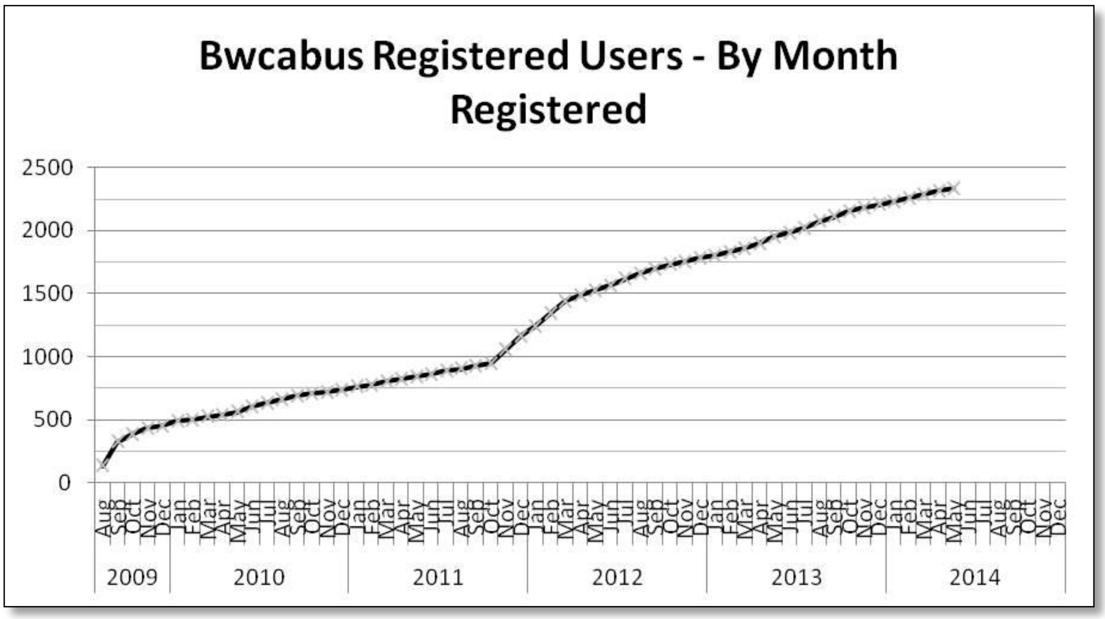
Over 2300 members have registered to use Bwcabus since August 2009. Table 1 presents a breakdown of the membership profile and highlights the popularity of the scheme with users under 25 years of age and users over 60 years old.

Table 1 – Bwcabus Registered members Profile

	Number	*Percentage of Total Membership*
Total Registered Users	2334	
Active Users in last 12 months	528	23%
Members who have never used the service	1052	45%
Female members	1548	66%
Male Members	786	34%
Members under 25	511	22%
Members 25-44	414	18%
Members 45-59	349	15%
Members 60 or over	1060	45%
Members with a mobility impairment	168	7%

Figure 3 shows that Bwcabus membership levels are continuing to increase despite the maturity of the scheme. On average 32 new members register each month (2013-14 figures).

Figure 3 – Growth of Registered Bwcabus members from August 2009 – May 2014



*The service area and number of vehicles was doubled in December 2011

In total 90,118 passenger journeys have been completed. Table 2 shows the yearly breakdown of passenger numbers. 2013 was a record year for the Bwcabus service, with 26,947 passenger journeys completed. 2014 data indicates a continued growth in passenger journeys, an 11.5% increase recorded from January – April 2014 as compared to the same period in 2013.

Table 2 – Bwcabus Passenger Journeys by Year of Operation (December 2009 – April 2014)

Year	*Passenger Journeys*	*Number of Operating Days*	*Average number of passengers per day*
2009	4,544	109	41
2010	12,586	301	42
2011	13,246	304	44
2012	23,771	309	77
2013	26,947	306	88
2014**	9,024	101	89
Total	**90,118**	**1405**	**63**

*The service area and number of vehicles was doubled in December 2011

**Data for January to April 2014 only

The booking system performance is measured by the number of referrals generated for manual scheduling as a proportion of the total number of demand responsive bookings made. A referral is generated by the system, when it cannot offer the passenger a time and a manual scheduler takes over to see if the journey can be accommodated. Table 3 shows the system performance between1st May 2013 – 30th April 2014.

Table 3 – Bwcabus Booking System Performance: Booking Referrals by Month (1st May 2013 – 30th April 2014)

Month	*Number of Referrals*	*Booking Rate (%)*
May	285	77.63
June	194	83.12
July	210	81.01
Aug	283	78.47
Sep	202	85.11
Oct	194	85.48
Nov	225	86.13
Dec	176	84.87
Jan	208	82.51
Feb	243	82.25
Mar	362	77.69
Apr	213	83.24
Average	**233**	**82.29**

A survey of 100 Bwcabus passengers undertaken in July 2013 highlighted:

- 70% either agreed or strongly agreed that they are now making trips that they would not have been able to make prior to the Bwcabus
- 74% of respondents agreed or agreed strongly that the Bwcabus has provided them with better opportunities to access travel
- 42% either agreed or strongly agreed that they have reduced the number of trips made by car since using the Bwcabus

Conclusions

The Bwcabus system overcomes a number of design challenges:

- Optimisation: how the system would optimise journeys to form the schedule vs. the demands of passengers, who expect the bus to be available when they want to travel.
- Manual Intervention: coping with manual [human] input which can introduce journeys onto the schedule that break system rules and would result in the system being unable to make logical sense of the journey ordering.
- Operational Efficiency: joining up similar journeys (based on origin, destination, direction of travel, journey time), so that passengers travel together on a fewer number of bus trips, with constraints to ensure maximum detour values (a factor of the original journey time) are not exceeded.

The implementation of the Bwcabus scheduling system demonstrates a solution to providing 'dynamic' demand responsive transport scheduling in rural areas. This approach has proven that providing rural communities with an integrated rural public transport network can increase the frequency of public transport use, improve accessible by public transport, and encourage a reduction in car use.

References

Baugh, J., G. Kakivaya, and J. Stone (1998). Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. Engineering Optimization 30 ,91–123.

Gerrard, M. (1974). Comparison of taxi and dial-a-bus services. Transportation Science 8, 85–101

Hall, C., H. Andersson, J. Lundgren, and P. Varbrand (2009). The integrated dial-a-ride problem. Public Transport 1, 39 –54

Uchimura, K., H. Takahashi, and T. Saitoh (2002) . Demand responsive services in hierarchical public transportation system. IEEE Transactions on Vehicular Technology 51, 760–766.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Bullet TimeTabler Education – System demonstration

Pedro Fernandes · Armando Barbosa · Luis Moreira

Keywords Automatic timetabling · Higher Education · Decision support system · Heuristics · Combinatorial optimization · Timetables

1 Introduction

The problem of creating timetables for Educational Institutions is typically defined as the scheduling of a set of lessons involving teachers and students, on a set of classrooms, in certain time slots, considering a number of constraints (Schaerf 1999; Bonutti et al. 2012).

Throughout an academic year, in every Educational Institution, a considerable number of days and human resources are spent trying to find, manually, a solution that respects all the existing rules (a feasible solution) and that, at the same time, can meet the expectations of all participants (a quality solution).

Due to its combinatorial nature and associated complexity, this is one of the most studied problems by the scientific community and by the Operational Research area in particular (Schaerf 1999; Murray et al. 2007).

This paper presents the product Bullet TimeTabler Education (BTTE), an automatic and optimized generator of timetables. This software application is the result of the work carried out by Bullet Solutions since 2005. BTTE has been updated and improved over the years with the contributions from almost all the Portuguese Higher

Pedro Fernandes (✉)
Bullet Solutions, Porto, Portugal
E-mail: pedro.fernandes@bulletsolutions.com

Armando Barbosa
Bullet Solutions, Porto, Portugal
E-mail: armando.barbosa@bulletsolutions.com

Luis Moreira
Bullet Solutions, Porto, Portugal
E-mail: luis.moreira@bulletsolutions.com

Education Schools (Universities and Polytechnic Institutes), as well as some foreign institutions.

The BTTE software is successfully used in more than half of the Portuguese Higher Education Schools, including the 10 major ones.

2 Bullet TimeTabler Education

2.1 Application suite

The Application Suite consists of software modules that exchange data with each other. In this paper only two of them are mentioned: BTTE – automated generation of timetables; and Bullet Calendar (BC) – edition of timetables and management of events and resources.

BTTE is the intelligent centrepiece of the system, the calculation engine and therefore it is the main focus of this paper.

Figure 1 shows the application diagram.

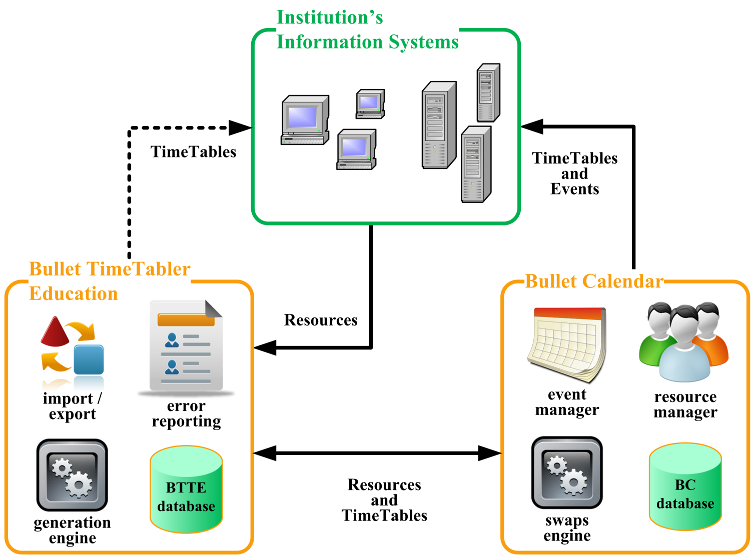


Fig. 1 Application diagram

BTTE is an innovative software application that automatically generates timetables for Higher Education Institutions. It combines and optimizes several objectives in accordance with the interests of the Institution, optimizing schedules for teachers and students as well as the classrooms' occupation, among other goals.

BC is a module that allows an efficient daily management of all activities and resources in an Educational Institution, through a decision support tool. This application can work independently or integrated with BTTE.

Figure 2 shows the Application Suite's workflow.

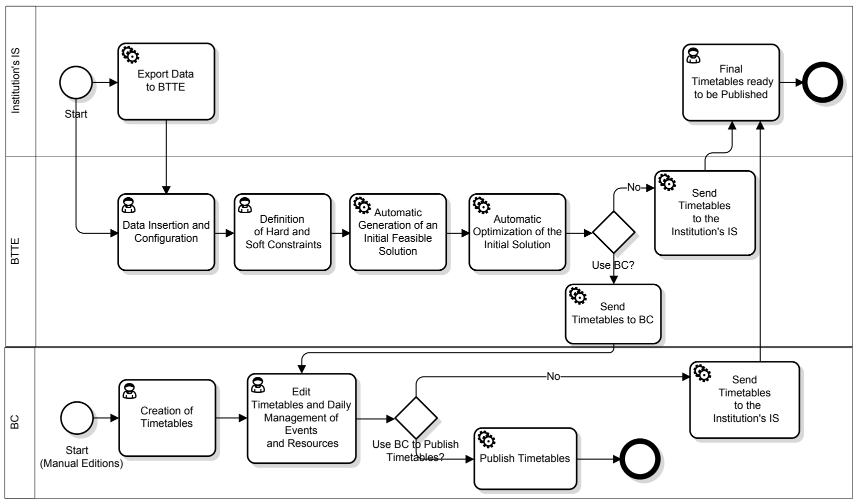

Institution's IS
BTTE
BC
Start
Export Data to BTTE
Final Timetables ready to be Published
Data Insertion and Configuration
Definition of Hard and Soft Constraints
Automatic Generation of an Initial Feasible Solution
Automatic Optimization of the Initial Solution
Use BC?
No
Send Timetables to the Institution's IS
Send Timetables to BC
Start (Manual Editions)
Creation of Timetables
Edit Timetables and Daily Management of Events and Resources
Use BC to Publish Timetables?
No
Publish Timetables
Send Timetables to the Institution's IS

Fig. 2 Application Suite's workflow

2.2 Model definition

BTTE does not schedule individual students, only groups. The assignment of students to groups occurs in a separate process, where the students apply for the existing timetables, choosing the most suitable ones. In some cases, the Institution itself defines the association of students to the groups. The conflicts are based on the curricula structure and the Portuguese situation can be included in the Curriculum-Based Course Timetabling problem (Bonutti et al. 2012).

A complete analysis of the proposed model, the fundamental concepts and the constraints (hard and soft) involved in the problem, can be found in our previous work (Fernandes et al. 2013).

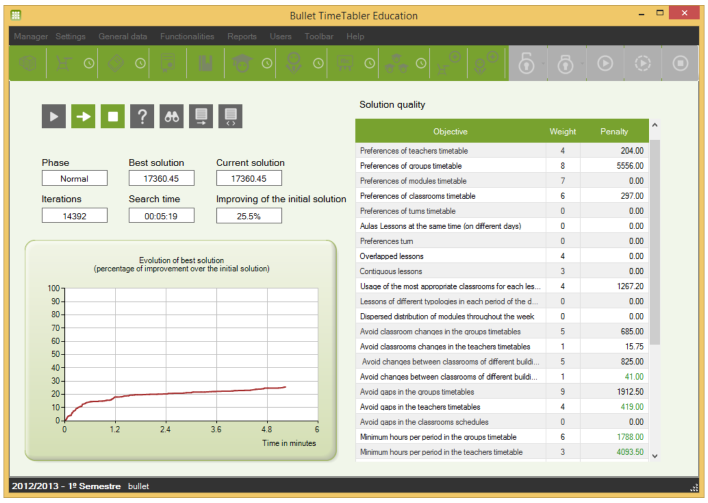
2.3 Building the timetable

A sequential heuristic is used to build an initial timetable from an empty timesheet. Once the initial solution to the problem is found (the starting point), the optimization phase is initiated; based on appropriate methods, better solutions are progressively searched. In BTTE, the search for new solutions is based on neighbourhood structures. Besides the different construction methods of neighbourhoods, the implemented optimization algorithms go through three major phases: normal, intensification and diversification. Each of these phases is specified to achieve a particular purpose, and their joint operation is the key to a final optimized outcome.

The heuristics were fully developed from scratch, adapted to the existing problem and created model.

Figure 3 shows a screenshot of the optimization phase.

A complete description of the proposed algorithms (construction heuristics and improvement heuristics) can be found in our previous work (Fernandes et al. 2013).


Bullet TimeTabler Education
Manager Settings General data Functionalities Reports Users Toolbar Help
Evolution of best solution (percentage of improvement over the initial solution)
Time in minutes
2012/2013 - 1º Semestre bullet

Phase	Best solution	Current solution
Normal	17360.45	17360.45

Iterations	Search time	Improving of the initial solution
14392	00:05:19	25.5%

Solution quality

Objective	Weight	Penalty
Preferences of teachers timetable	4	204.00
Preferences of groups timetable	8	5556.00
Preferences of modules timetable	7	0.00
Preferences of classrooms timetable	6	297.00
Preferences of turns timetable	0	0.00
Aulas Lessons at the same time (on different days)	0	0.00
Preferences turn	0	0.00
Overlapped lessons	4	0.00
Contiguous lessons	3	0.00
Usage of the most appropriate classrooms for each les...	4	1267.20
Lessons of different typologies in each period of the d...	0	0.00
Dispersed distribution of modules throughout the week	0	0.00
Avoid classroom changes in the groups timetables	5	685.00
Avoid classrooms changes in the teachers timetables	1	15.75
Avoid changes between classrooms of different buildi...	5	825.00
Avoid changes between classrooms of different buildi...	1	41.00
Avoid gaps in the groups timetables	9	1912.50
Avoid gaps in the teachers timetables	4	419.00
Avoid gaps in the classrooms schedules	0	0.00
Minimum hours per period in the groups timetable	6	1788.00
Minimum hours per period in the teachers timetable	3	4093.50

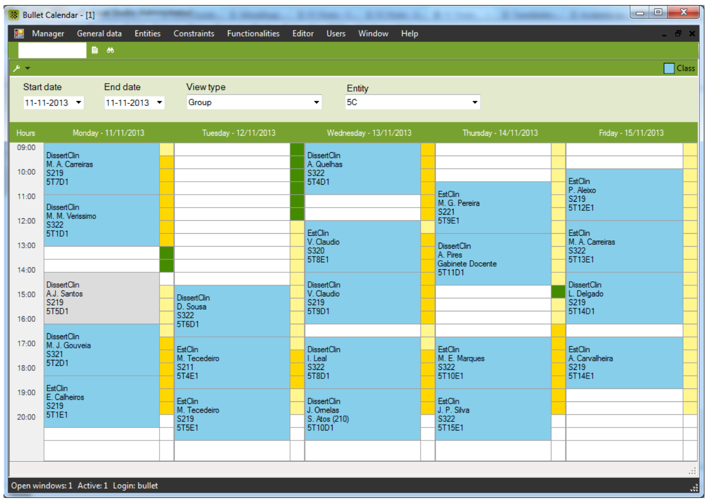
Fig. 3 BTTE – Optimization phase

2.4 Editing the timetable

When the automatic generation stage ends, BTTE sends the timetables to BC, where the user can edit timetables and manage events and resources.

BC is a module that is used continually, since it can easily respond to all the changes that inevitably occur during the year. If, at any time, profound changes are needed, the user can get back to BTTE and generate a complete solution from scratch, or just generate the affected events keeping the rest of the solution untouched in BC.

Figure 4 shows a screenshot of a swap operation using BC.


Bullet Calendar - [1]
Manager General data Entities Constraints Functionalities Editor Users Window Help
Class
Open windows: 1 Active: 1 Login: bullet

Start date	End date	View type	Entity
11-11-2013	11-11-2013	Group	5C

Monday - 11/11/2013	Tuesday - 12/11/2013	Wednesday - 13/11/2013	Thursday - 14/11/2013	Friday - 15/11/2013
DissertClin M. A. Carreiras S219 5T7D1	DissertClin D. Sousa S322 5T6D1	DissertClin A. Quelhas S322 5T4D1	EstClin M. G. Pereira S221 5T9E1	EstClin P. Aleixo S219 5T12E1
DissertClin M. M. Verissimo S322 5T1D1	EstClin M. Tecedeiro S211 5T4E1	EstClin V. Claudio S320 5T8E1	DissertClin A. Pires Gabinete Docente 5T11D1	EstClin M. A. Carreiras S322 5T13E1
DissertClin A.J. Santos S219 5T5D1	EstClin M. Tecedeiro S219 5T5E1	DissertClin V. Claudio S219 5T9D1	EstClin M. E. Marques S322 5T10E1	DissertClin L. Delgado S219 5T14D1
DissertClin M. J. Gouveia S321 5T2D1		DissertClin I. Leal S322 5T8D1	EstClin J. P. Silva S322 5T15E1	EstClin A. Carvalheira S219 5T14E1
EstClin E. Calheiros S219 5T1E1		DissertClin J. Omelas S. Atos (210) 5T10D1		

Fig. 4 BC – Swap operation

Some of the main features of BC are: totally flexible agenda, with easy configuration of time slots and working hours; real-time information about the occupation of the Institution's resources; quick editing of timetables with several views; quick analysis of all the possibilities and limitations of swapping a specific event; total flexibility for changing any resource associated with any event; fast Web publishing of timetables; exporting and printing schedules and dozens of different reports.

3 Conclusions

The main conclusions that can be extracted from the work developed are closely linked with the commercial success of the BTTE application.

In our previous work (Fernandes et al. 2013), twenty real cases of Portuguese institutions, users of BTTE, were analysed. The fact that results with good quality, with savings of 85% on the time spent in the process, were obtained in all analysed cases, leads to the conclusion that the algorithms used in the BTTE application have a considerable level of robustness and ease of adaptation to the quite diverse real scenarios that were used for their evaluation.

Recently, after testing different parametrisation of the heuristics and with the improvement of the data structures that support the product, very interesting results were obtained. In some cases, optimized timetables were produced 10 times faster when compared with the results presented in Fernandes et al. (2013). These results will be published in the short term.

The implementation and use of the BTTE application allowed significant improvements in processes directly and indirectly related with the creation of timetables, resulting in additional productivity gains.

It was possible to observe the organization and centralization of information in most of the institutions and the elimination of redundant information. An increase in speed and efficiency in the workflow and in the information flow was also observed.

There is a better control of the process by top management, particularly the real needs of the distribution of teaching service. Scenarios where the generation of timetables was far from top management and dispersed by various departments, were common in the past, resulting in the recruitment of resources that later on proved to be unnecessary. There have been considerable savings in hiring teachers, after using the BTTE application.

References

Bonutti, A., De Cesco, F., Di Gaspero, L., & Schaerf, A. (2012). Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research, 194(1)*, 59-70.

Fernandes, P., Pereira, C. S., & Barbosa, A. (2013). Automatic timetabling in Higher Education Institutions: a real scenario and solution. In *Proceedings of the 6th Multidisciplinary International Conference on Scheduling: Theory and Applications*, 151-170.

Murray, K., Müller, T., & Rudová, H. (2007). Modeling and Solution of a Complex University Course Timetabling Problem. *Practice and Theory of Automated Timetabling VI, E. K. Burke and H. Rudová, Eds. Springer-Verlag Berlin*, 189-209.

Schaerf, A. (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review, 13(2)*, 87-127.

10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Bullet TimeTabler Education: latest improvements towards a more efficient timetabling

Pedro Fernandes · **Carla Sofia Pereira** ·
Armando Barbosa

Keywords Automatic timetabling · Higher Education · Decision support system · Heuristics · Combinatorial optimization · Timetables

1 Introduction

In a time where the need to reduce costs has become part of day to day reality of all Educational Institutions, it is unthinkable to continue manually performing those tasks that can be automated and optimized – the creation of timetables.

The problem of creating timetables for Educational Institutions is typically defined as the scheduling of a set of lessons involving teachers and students, on a set of classrooms, in certain time slots, considering a number of constraints (Schaerf 1999; Bonutti et al. 2012).

Due to its combinatorial nature and associated complexity, the automatic creation of timetables for Educational Institutions is a problem studied by the scientific community since the decade of 1960, and by the Operational Research area in particular (Schaerf 1999; Murray et al. 2007). Over almost 50 years, hundreds of studies were published, with many different formulations of the problem and solving techniques.

In our previous work (Fernandes et al. 2013), a new automatic and optimized generator of timetables for Higher Education Institutions was presented – the product Bullet TimeTabler Education (BTTE), which is successfully used in more than

Pedro Fernandes (✉)
Bullet Solutions, Porto, Portugal
E-mail: pedro.fernandes@bulletsolutions.com

Carla Sofia Pereira
CIICESI-ESTGF, Instituto Politécnico do Porto, Felgueiras, Portugal
INESC TEC (formerly INESC Porto), Porto, Portugal
E-mail: carla.pereira@eu.ipp.pt

Armando Barbosa
Bullet Solutions, Porto, Portugal
E-mail: armando.barbosa@bulletsolutions.com

half of the Portuguese Higher Education Schools, including the 10 major ones. In Fernandes et al. (2013), among other things, we describe a new model of the problem, the algorithms developed for solving it, and we also analyse twenty real cases of Portuguese institutions, users of BTTE.

This paper presents important updates to the aforementioned previous work (Fernandes et al. 2013) and the next research directions that were chosen by Bullet Solutions to improve the BTTE software.

2 Recent work and achievements

2.1 Information structure

As indicated in Fernandes et al. (2013), the data structure created for BTTE allows updating and manipulating all the existing data maps in real time. These data maps are crucial to the developed heuristics (both constructive and improvement), because they can, at any time, provide a clear view of the existing possibilities. They support the classification of the most urgent event in each moment, they help defining which is the best slot to place it, what is the impact of placing the event in other slots, which are the best events to swap or move, just to name a few possibilities.

A deep reformulation of this data structure was made. Tests and validations that proved to be redundant were eliminated, and the storage of the data maps was also improved. All these changes were made keeping the same quality control.

In the end, the preliminary tests showed that the speed performance of the system was improved by about 50%.

2.2 Algorithms

Besides the reformulation of the data structure, several improvements were made on the heuristics.

As showed in Fernandes et al. (2013), there are two different calculation phases in BTTE. First, a sequential heuristic is used to build an initial timetable from an empty timesheet (construction heuristics). Once the initial solution to the problem is found (the starting point), the optimization phase is initiated (improvement heuristics); based on appropriate methods, better solutions are progressively searched. In BTTE, the search for new solutions is based on neighbourhood structures.

In the construction phase, after the gains with the reformulation of the data structure were obtained, the focus of the research was directed to the implemented escape methods (crucial to achieve feasible solutions in very restricted problems). After analysing the critical points of the methods with the help of a performance profiling software, it was clear that almost all of the calculation effort was concentrated on the search for new possible allocations of the events previously placed. It was also possible to see that in the more complex cases, where some critical factors identified in Fernandes et al. (2013) are involved, the calculation time needed to complete a single move increased hugely. A new method to solve that type of cases was developed, with a performance about 5 times faster than the previous one (basically, for the very complex cases, where speed is fundamental, a new possible move is now obtained 5 times faster than before).

In the improvement phase, a different parametrisation of the heuristics was tested. In one of the attempts, by drastically reducing the number of neighbours created in each iteration of the normal phase (the phase where it is supposed to explore the solution space in a freer way), we found out that although the number of iterations where the solution was not improved was much higher than before (as expected), solutions of similar quality, in the end of the phase, were achieved 10 times faster.

2.3 Preliminary results

Besides the preliminary general tests made with some of the case studies presented in Fernandes et al. (2013), Cases 18 and 20 were tested live with the clients. In the two cases, this beta version of the heuristics needed respectively about 4 and 2 hours to calculate an initial solution (half of the time than before); moreover, the first big boost in the optimization phase ended about 5 times faster (about 4 hours for Case 18 and 24 hours for Case 20). These first live results, together with the other general tests performed, confirmed that very promising results were already achieved.

3 Future work

Currently, we are finishing the development of a logging framework that will be attached to the heuristics, which will provide access to crucial information. BTTE is used by dozens of heterogeneous clients and the potential of the information that will be collected, in real diverse scenarios, is huge, since it will allow much quicker adaptations in the future and the incorporation of additional intelligence in the critical points of the heuristics: urgency criterion of the events, escape methods, neighbourhood structures and the right parametrisation of all the variables involved.

Soon, we will be able to analyse new indicators, such as the type of moves that are more appropriated to restore the path of a feasible solution (escape methods), the number of iterations where a best solution is found in each of the phases and in each of the runs of the improvement heuristics, between many others. With the additional knowledge that will be obtained from this data, crossing with other information already known such as the critical factors that introduce complexity into the problem (Fernandes et al. 2013), we believe that it will be possible to place BTTE in an even higher level of quality.

We expect to present the results and conclusions of our current work in the Conference, not only the results obtained with the new heuristics after the conclusion of the tests, but also, at least, some preliminary analysis of the information collected with the logging framework, that we believe will provide precious information to future research directions.

References

Bonutti, A., De Cesco, F., Di Gaspero, L., & Schaerf, A. (2012). Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research, 194(1)*, 59-70.

Fernandes, P., Pereira, C. S., & Barbosa, A. (2013). Automatic timetabling in Higher Education Institutions: a real scenario and solution. In *Proceedings of the 6th Multidisciplinary International Conference on Scheduling: Theory and Applications*, 151-170.

Murray, K., Müller, T., & Rudová, H. (2007). Modeling and Solution of a Complex University Course Timetabling Problem. *Practice and Theory of Automated Timetabling VI, E. K. Burke and H. Rudová, Eds. Springer-Verlag Berlin*, 189-209.

Schaerf, A. (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review, 13(2)*, 87-127.

A Web-Software to handle XHSTT Timetabling Problems

George H.G. Fonseca · Thaise D. Delfino · Haroldo G. Santos

Abstract This work presents a Web-Software to handle XHSTT timetabling problems. The XHSTT format is complex and virtually only timetabling researchers are able to work with this file format. The software goal is to abstract the user from XHSTT knowledge and make any person able to specify and solve timetabling problems through XHSTT format. This software may popularize this file format and aid the new real world instances specification.

Keywords Timetabling · XHSTT format · Web-Software

1 Introduction

The diversity of School Timetabling models encountered around the world motivated the definition of an XML format (XHSTT) to express different entities and constraints considered when building timetables [6]. Based in this format definition and aiming to stimulated the research in this area, the Third International Timetabling Competition (ITC2011) occurred in 2012 [5]. It motivated the development of several efficient algorithms for Timetabling problems [2,7,1].

A problem faced in timetabling research is the gap between research and practice - many algorithms are proposed and validated in fictitious instances of the problem. Additionally, many of these algorithms as the ones developed for ITC2011 does not implement an intuitive user interaction mode. This makes it difficult to apply these algorithms to solve real problems. Thus, the objective of this paper is to describe a Web-Software developed to assist the specification of XHSTT timetabling problems and the solution of these problems through recognized quality solvers.

Systems and Computing Department
Federal University of Ouro Preto
Ouro Preto, Minas Gerais, Brazil 35400-000

2 XHSTT format

XHSTT format was proposed by Post *et al.* [6] to specify timetabling problems. It is split into four main entities:

Times
Set of timeslots available for assignments;

Resources
Set of resources involved in the problem (usually teachers, classes and rooms);

Events
Set of events that must be scheduled (usually lessons);

Constraints
Set of constraints that must be satisfied to a solution be feasible and/or good.

A solution of an XHSTT specified problem consists in an assignment of timeslots and resources to the events respecting the given mandatory constraints and maximizing the attendance of desirable constraints. A detailed explanation of the file format is presented by Kingston [3].

3 Software User Interface

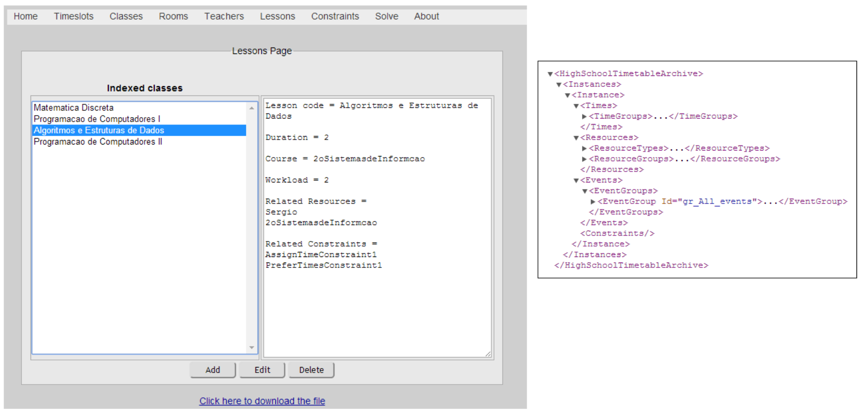
The developed software provides a HTML interface for the user to specify any instance of timetabling problem in XHSTT format. The interface design of the software was made intending to find an equilibrium point between learning curve and similarity to the XHSTT format. After specified, the XHSTT modeled problem may be submitted to the ITC2011 winner solver [2]. Figure 1 presents the general interface of our software and the automatically generated entities of XHSTT file in the instance creation.

Note that some entities creation were automated intending to make easier for the user to operate the software. As one can see in Figure 1, the elements *TimeGroups*, *ResourceGroups* and *EventGroups* are created automatically. The Times specification is made based only on the number of days in the schedule and the number of timeslots by day. The concept of Role was also abstracted from the user. Our software only allows the user to create two roles for an event: Teacher and Room, denoted respectively as 0 and 1.

After initializing an instance, the user should specify the time-slots available for the assignment. Figure 2 presents the automated *Times* entity creation through our software.

The next step in XHSTT instance generation through our software is the available resources input. The resource creation through our system is quite easy. We consider only three types of resource: classes, teachers and rooms. To create any of them, the user interface is similar as follows at Figure 3.

Defined the available time-slots and resources, one needs to specify the events to be scheduled. Figure 4 presents the automated *Event* entity creation through our software.

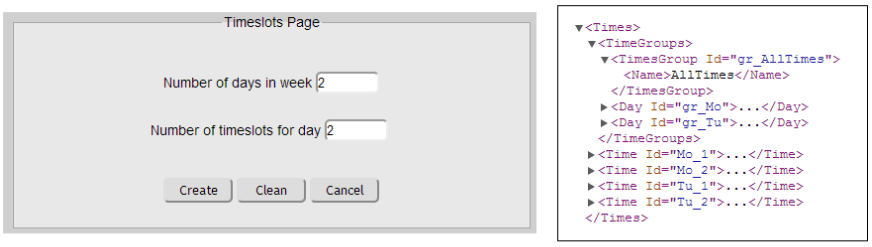


Matematica Discreta
Programacao de Computadores I
Algoritmos e Estruturas de Dados
Programacao de Computadores II

```
Lesson code = Algoritmos e Estruturas de
Dados

Duration = 2

Course = 2oSistemasdeInformcao

Workload = 2

Related Resources =
Sergio
2oSistemasdeInformcao

Related Constraints =
AssignTimeConstraint1
PreferTimesConstraint1
```

```
▼<HighSchoolTimetableArchive>
  ▼<Instances>
    ▼<Instance>
      ▼<Times>
        ▶<TimeGroups>...</TimeGroups>
        </Times>
      ▼<Resources>
        ▶<ResourceTypes>...</ResourceTypes>
        ▶<ResourceGroups>...</ResourceGroups>
        </Resources>
      ▼<Events>
        ▼<EventGroups>
          ▶<EventGroup Id="gr_All_events">...</EventGroup>
          </EventGroups>
        </Events>
        <Constraints/>
      </Instance>
    </Instances>
  </HighSchoolTimetableArchive>
```

Fig. 1 Main user interface of software and XHSTT tags generated when a new instance is created.

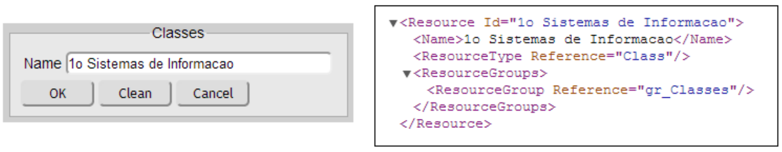


Number of days in week: 2

Number of timeslots for day: 2

```
▼<Times>
  ▼<TimeGroups>
    ▼<TimesGroup Id="gr_AllTimes">
        <Name>AllTimes</Name>
      </TimesGroup>
    ▶<Day Id="gr_Mo">...</Day>
    ▶<Day Id="gr_Tu">...</Day>
    </TimeGroups>
  ▶<Time Id="Mo_1">...</Time>
  ▶<Time Id="Mo_2">...</Time>
  ▶<Time Id="Tu_1">...</Time>
  ▶<Time Id="Tu_2">...</Time>
  </Times>
```

Fig. 2 *Times* entity through our software.



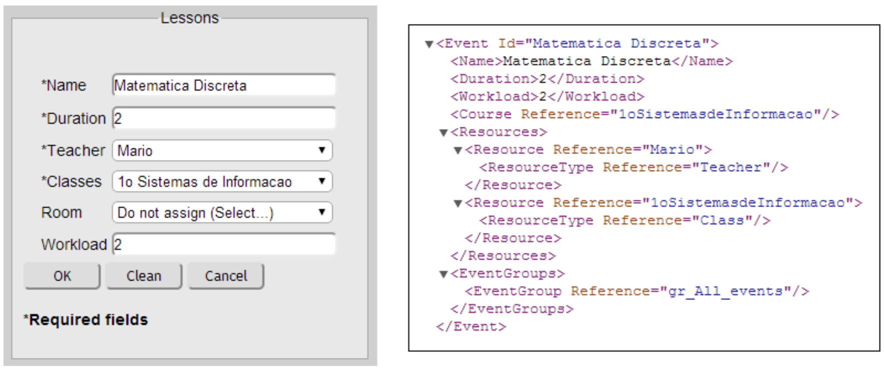
Name: 1o Sistemas de Informacao

```
▼<Resource Id="1o Sistemas de Informacao">
    <Name>1o Sistemas de Informacao</Name>
    <ResourceType Reference="Class"/>
  ▼<ResourceGroups>
      <ResourceGroup Reference="gr_Classes"/>
    </ResourceGroups>
  </Resource>
```

Fig. 3 *Resource* entity creation through our software.

After specify the time-slots, resources and events, the user should input the constraints to his problem. To handle the constraints entities, we create one window for each constraint from XHSTT format. Figure 5 presents the automated *Constraint* entity creation with our software, specifically an *Avoid Unavailable Times Constraint*.

At this point the whole instance is specified and ready to be solved. The user may also change any information at any moment. Now the user should specify the execution time limit and the solver for the instance as Figure 6 points out. After the specified running time the software will automatically generate a HTML page containing the planning timetables by resource and by event as well as the violated constraints.



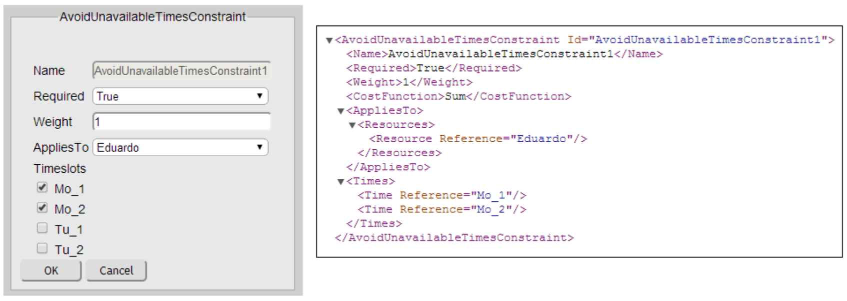
Lessons

*Name	Matematica Discreta
*Duration	2
*Teacher	Mario
*Classes	1o Sistemas de Informacao
Room	Do not assign (Select...)
Workload	2

*Required fields

```
▼<Event Id="Matematica Discreta">
   <Name>Matematica Discreta</Name>
   <Duration>2</Duration>
   <Workload>2</Workload>
   <Course Reference="1oSistemasdeInformacao"/>
 ▼<Resources>
  ▼<Resource Reference="Mario">
     <ResourceType Reference="Teacher"/>
   </Resource>
  ▼<Resource Reference="1oSistemasdeInformacao">
     <ResourceType Reference="Class"/>
   </Resource>
  </Resources>
 ▼<EventGroups>
    <EventGroup Reference="gr_All_events"/>
  </EventGroups>
</Event>
```

Fig. 4 *Event* entity creation through our software.



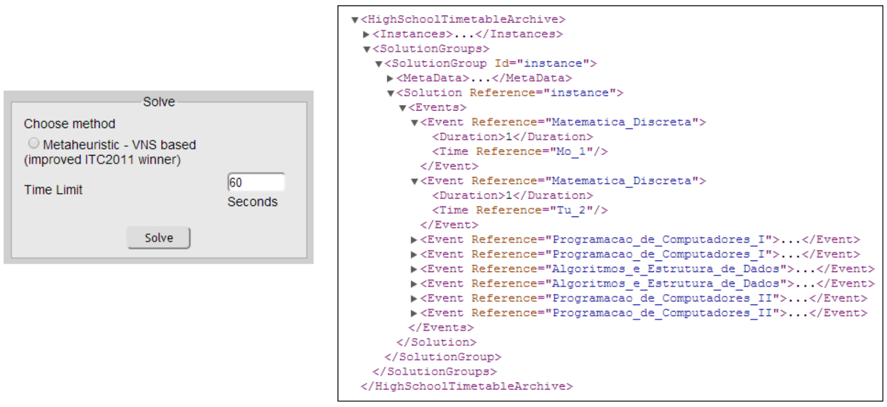
AvoidUnavailableTimesConstraint

Name	AvoidUnavailableTimesConstraint1
Required	True
Weight	1
AppliesTo	Eduardo

Timeslots
- [x] Mo_1
- [x] Mo_2
- [ ] Tu_1
- [ ] Tu_2

```
▼<AvoidUnavailableTimesConstraint Id="AvoidUnavailableTimesConstraint1">
   <Name>AvoidUnavailableTimesConstraint1</Name>
   <Required>True</Required>
   <Weight>1</Weight>
   <CostFunction>Sum</CostFunction>
  ▼<AppliesTo>
   ▼<Resources>
      <Resource Reference="Eduardo"/>
    </Resources>
   </AppliesTo>
  ▼<Times>
    <Time Reference="Mo_1"/>
    <Time Reference="Mo_2"/>
   </Times>
 </AvoidUnavailableTimesConstraint>
```

Fig. 5 *Constraint* entity creation through our software.



Solve

Choose method
○ Metaheuristic - VNS based (improved ITC2011 winner)

Time Limit 60 Seconds

```
▼<HighSchoolTimetableArchive>
 ▶<Instances>...</Instances>
 ▼<SolutionGroups>
  ▼<SolutionGroup Id="instance">
   ▶<MetaData>...</MetaData>
   ▼<Solution Reference="instance">
    ▼<Events>
     ▼<Event Reference="Matematica_Discreta">
        <Duration>1</Duration>
        <Time Reference="Mo_1"/>
      </Event>
     ▼<Event Reference="Matematica_Discreta">
        <Duration>1</Duration>
        <Time Reference="Tu_2"/>
      </Event>
     ▶<Event Reference="Programacao_de_Computadores_I">...</Event>
     ▶<Event Reference="Programacao_de_Computadores_I">...</Event>
     ▶<Event Reference="Algoritmos_e_Estrutura_de_Dados">...</Event>
     ▶<Event Reference="Algoritmos_e_Estrutura_de_Dados">...</Event>
     ▶<Event Reference="Programacao_de_Computadores_II">...</Event>
     ▶<Event Reference="Programacao_de_Computadores_II">...</Event>
     </Events>
    </Solution>
   </SolutionGroup>
  </SolutionGroups>
 </HighSchoolTimetableArchive>
```

Fig. 6 Instance solution through our software.

4 Concluding Remarks

This paper presented a Web-Software for handling XHSTT timetabling problems. The software is available at https://sites.google.com/site/georgehgfonseca/software/timetabler. We invite the interested reader to use and contribute to our software. The proposed software aims to be easy to use, even by people who does not know about the XHSTT format. Indeed the great majority of people responsible by the timetabling in schools does not know yet about this format. In this sense the software may also help to popularize this file format to express timetabling problems.

The software may also assists researchers and practitioners to contribute with new real instances of the problem from their institutions.

As future research we suggest to:

– Introduce more solvers to the software;
– Evaluate the user experience with the software;
– Improve the usability of the software;
– Integrate this software with HSEval evaluator software [4].

References

1. Fonseca, G., Santos, H.: Variable neighborhood search based algotihms for high school timetabling. Computers and Operational Research (2014)
2. Fonseca, G., Santos, H., Toffolo, T., Brito, S., Souza, M.: A sa-ils approach for the high school timetabling problem. Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012) pp. 493–496 (2012)
3. Kingston, J.H.: A software library for school timetabling (2012). Available at http://sydney.edu.au/engineering/it/~jeff/khe/, May 2012
4. Kingston, J.H.: The hseval high school timetable evaluator (2014). Available at http://sydney.edu.au/engineering/it/~jeff/hseval.cgi, February 2014
5. Post, G., Di Gaspero, L., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. In: Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (2012)
6. Post, G., Kingston, J., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., Schaerf, A.: XHSTT: an XML archive for high school timetabling problems in different countries. Annals of Operations Research pp. 1–7
7. Sorensen, M., Kristiansen, S., Stidsen, T.: International Timetabling Competition 2011: An Adaptive Large Neighborhood Search algorithm, pp. 489–492 (2012)

Author Index

ISBN 9780992998400

9 780992 998400

PATAT 2014

10th International Conference on the Practice and Theory of Automated Timetabling

York, United Kingdom, Tuesday 26th - Friday 29th August 2014