Local Search Methods for the Post Enrolment-based Course Timetabling Problem

Lisa Ann Taylor

School of Mathematics

Cardiff University

A thesis submitted for the degree of

Master of Philosophy

September 2013

Summary

The work presented in this thesis concerns the problem of post enrolment-based course timetabling. The motivation for this is the increasing importance of the automation of timetabling due to the growth in popularity of Higher Education in recent years. There were 464,910 accepted applicants to universities in the United Kingdom in 2012 which is a 12% rise in five years[1]. This will inevitably lead to an expansion in the number of courses, modules and teachers. As a result, the ability to manually construct timetables has become increasingly impractical.

A two-stage approach is investigated that aims to use heuristic and metaheuristic approaches to obtain a satisfactory timetable that suits the needs of the staff and students at educational institutions. The first stage consists of using selection heuristics to construct an initial solution. Two approaches that then attempt to find feasibility are presented. The first applies a tabu search algorithm with a number of neighbourhood operators that navigate the search space for feasible solutions. The second approach implements a PartialCol algorithm.

The second stage aims to improve the solution quality by minimising the number of soft constraint violations. The feasibility ratio could be an indicator of the connectivity of the search space, so methods of increasing the feasibility ratio are presented. If the feasibility ratio can be increased then the number of soft constraint violations would be expected to decrease.

These techniques were applied to the 24 instances provided for track two of the International Timetabling Competition 2007. The conclusions of the experimentation and investigative processes show that the PartialCol algorithm was more successful, in terms of finding feasible solutions, than the method that employs the neighbourhood operators. However, improvements to the soft constraint penalty were achieved using these neighbourhood operators.

[1] http://www.ucas.com/data-analysis/data-resources/data-tables/data-summary

Declaration

This work has not previously been submitted in substance for any other degree or award at this or any other university or place of learning, nor is it being submitted concurrently in candidature for any other degree or other award.

Signed .....(candidate) Date .....

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of MPhil.

Signed .....(candidate) Date .....

Statement 2

This thesis is the result of my own independent work/investigations, except where otherwise stated. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed .....(candidate) Date .....

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed .....(candidate) Date .....

Acknowledgements

I would like to thank the many people without whom this research project would not have been possible. Firstly, my supervisors, Dr. Jonathan Thompson and Dr. Rhyd Lewis, thank you for your guidance, support and patience; I know it cant have been easy at times.

I would also like to extend my gratitude to the Cardiff University School of Mathematics for making this research nancially possible.

I express my appreciation to my thesis examiners, Professor Sanja Petrovic and Dr. Iskander Aliev. Your thoughtful and detailed comments were valued greatly.

I am deeply grateful to all of my good friends at Cardiff University, I truly believe I have made friends for life. Thank you for your support throughout good and bad times, for making my days bearable with numerous tea breaks, doughnut Thursdays and endless laughter. I would particularly like to thank Cheryl Voake-Jones for proof reading every single page of my thesis and inserting all the commas.

A special thank you needs to go to Stuart for his personal support, patience and culinary skills! Thank you for always putting a smile back on my face. Your faith in me never waivered even when I didnt have faith in myself.

I cannot find the words to express my gratitude to my family; my brother, sister, the stuck-ons and my perfect nieces and nephews who have become such a big part of my life for such little people! Most of all, I would like to thank my Mum and Dad. You have encouraged me to pursue my interests and have supported any decisions I have made in my life. You have offered unconditional support both financially and emotionally. I consider myself extremely lucky to have such a loving family around me and I love you all so much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Combinatorial optimisation problems involve finding a grouping, ordering, or assignment of a discrete, finite set of objects that satisfies given conditions. Examples of this type of problem include but are not limited to; vehicle routing, job scheduling, knapsack problem, and travelling salesman. All of these problems arise in situations where discrete choices must be made, and solving them amounts to finding an optimal solution among a finite number of alternatives. This thesis focusses on methods for solving a combinatorial optimisation problem, specifically, the post enrolment-based course timetabling problem.

Optimality relates to some benefit/cost criterion, which provides a quantitative measure of the quality of each solution. This area of discrete mathematics is of great practical use and has attracted much attention over the years. Many combinatorial optimisation problems are NP-hard. It is generally believed that NP-hard problems cannot be solved to optimality within polynomially bounded computation times. Consequently, there is much interest in approximation algorithms that can find near-optimal solutions of provable quality within reasonable running times.

It is worth noting here that a search space is defined as a space containing all possible solutions to a problem. If one solution can be transformed to another solution by some operation then

these solutions are neighbours. The constraints of a problem can vary the shape and size of the landscape. A fitness landscape is a way of visualising a problem and represents the relationship of the costs between neighbouring problems. The landscape can be spiky, meaning there are many peaks and troughs, with the troughs representing the local minima. Conversely, the landscape can be flat so there are not many local minima and it can mean that there is a strong relationship between neighbouring solutions. There have been methods presented that measure the landscape ruggedness (Angel and Zissimopoulos [2001]) but generally the number of peaks cannot be determined.

The automation of timetabling can be applied to many different areas, such as; sporting event scheduling (Kendall [2008]), transportation (Atkin et al. [2007]) and employee management (Ernst et al. [2004]). For assistance, there are specialist timetabling software available to use. However, these can be costly and it is very difficult to create a system that can cater for everybody's needs and take into account all requirements. These requirements are called 'constraints'.

The necessity of these constraints can vary, the most important constraint for one institution may be the least important for another. Many problems do have common constraints but the need for the constraint to be satisfied can vary. They can be split into two categories; hard and soft constraints. Hard constraints are requirements that must be satisfied in order for the timetable to be acceptable. If they are not satisfied the timetable will be rejected. Soft constraints are secondary constraints that will make the timetable more desirable. If a constraint is not adhered to we call it a violation. There must be no violations of hard constraints whereas there may be violations of soft constraints, but the fewer of these implies a more appealing timetable. The number of constraints in a problem can vary from relatively easy to solve problems with a low number of constraints to highly constrained problems. A feasible solution is one where all hard constraints are satisfied and all events are placed. A valid timetable is where no hard constraints are violated, however some events may be unplaced if it is not possible to insert them into a feasible period.

1.1 Timetabling in education

Timetabling is a problem commonly found in all types of educational institutions. The ability to construct a satisfactory timetable varies in difficulty according to many different factors, such as the number of students, teachers and courses. Furthermore, each problem is unique to the institution and can even vary for each department in each institution. Timetables are often created by hand, which can take many tedious and gruelling hours and can be reproduced several times a year as conditions change.

The reason why problems differ can be explained by fully understanding what institutions are trying to schedule. There are three main types of educational timetabling that are currently popular areas of study in Operational Research; exam timetabling, university course timetabling and school timetabling. Although they are similar in many ways, they have different constraints.

For exam timetabling the number of students required to attend each exam is already known. This means one constraint may be that the exam hall needs to be suitable for that number of students. Sometimes there is a set length of time in which to timetable the exams e.g. two weeks. Alternatively, the problem can be to timetable the exams in the shortest length of time possible.

University course timetabling can either be done before or after enrolment (post-enrolment). If a problem is post-enrolment, the number of students required to attend is known and therefore a constraint will be that no student can be required to attend two or more courses at the same time. However, if the number of students is unknown, i.e. before they enrol, then this cannot be a constraint. In the curriculum-based problem, conflicts between courses are set according to the curricula instead of the students that attend them. Teachers schedules and time constraints can also take precedence e.g. teacher five cannot take classes on a Wednesday.

School timetabling in many ways is similar to university course timetabling but the constraints and their importance differ. The main difference from the university course problem is that

university courses can have common students, whereas school classes are disjoint sets of students. If two courses have common students then they conflict, and they cannot be scheduled in the same period. Moreover, in the university problem, availability of rooms (and their size) plays an important role, whereas in the school problem they are often neglected because, in most cases, we can assume that each class has its own room and class sizes are uniform.

1.2 Aims and structure of this thesis

We will be giving a detailed description of the work we have completed on the post enrolment-based course timetabling problem. We investigate multi-stage techniques for solving highly constrained problems and post enrolment-based course timetabling in particular.

Stage one investigates the best ways of implementing local search to find a feasible solution. We will also investigate algorithms using both heuristic and metaheuristic methods. We begin by comparing selection heuristics for producing initial solutions. Two different methods are then compared that search for feasible timetables. Method one relaxes constraints that are difficult to satisfy, and presents methods that attempt to eliminating these relaxations. Method two leaves events unplaced and strives to find positions that do not violate any hard constraints.

Further phases extend the investigation to finding an improved solution by minimising the number of soft constraint violations. We experiment with a range of heuristics to reduce the soft constraint penalty. We aim to investigate ways of increasing the feasibility ratio which is the ratio of the number of moves that would maintain feasibility and the number of possible neighbourhood moves in the search space. It is hoped that by increasing this ratio we will increase the connectivity of the search space. We investigate the effects on the feasibility ratio when changes are made to the problem. These algorithms are applied to well-known instances presented by the international timetabling competition in 2007[1] and the results are compared with those from the literature.

[1] http://www.cs.qub.ac.uk/itc2007/

This thesis consists of a further four chapters; the following chapter provides an overview of the work previously completed on educational timetabling, the third chapter describes the two approaches to solving a post enrolment-based course timetabling problem, while the fourth chapter focusses on minimising the number of violations of the soft constraints. Chapter five covers the conclusions made.

1.3 A note on implementation and computational experimentation

All algorithm implementations presented in this thesis are programmed in C++, using Microsoft Visual Studio 2010 and code optimisation set to maximise speed (/O2). The computational experiments were executed on a PC under Windows 7 with a 3.00GHz processor.

Chapter 2

Literature review

Producing a suitable timetable is often tackled by universities every semester and can take a lot of administrative effort. Current approaches typically involve slight automated computer input and a large amount of manual input, if automation is used at all. This chapter will give an outline of the literature on these automated processes. Many surveys and reviews are available which give an overview of timetabling and the literature (Bardadym [1996], Burke et al. [1997], Burke and Petrovic [2002], Carter and Laporte [1996], Carter [1986], de Werra [1985], Petrovic and Burke [2004], Schaerf [1999]).

Virtually every school, college and university throughout the world will encounter the important task of timetabling. The previous United Kingdom (UK) Government pledged their commitment to Further Education and set a target of 50% of people under the age of 30 to undertake some form of Higher Education[1]. The present UK Government removed the target but states, in their widening participation policy[2], that all those with the potential to benefit from successful participation in Higher Education should have the opportunity to do so. This will inevitably lead to an expansion in diversity and, as a result, the ability to manually construct timetables suitable for the increasing number of courses, modules, teachers and students will progressively

[1] http://www.bis.gov.uk/assets/BISCore/corporate/MigratedD/publications/F/future_of_he.pdf
[2] https://www.gov.uk/government/policies/widening-participation-in-higher-education–4

become impractical. Automated timetabling is therefore currently a popular research area.

To introduce the problem of timetabling, a thorough literature search has been undertaken. We provide briefly the history of this field, by listing the key contributors which have had the most influence on this thesis. This chapter can be used as a helpful resource, as it provides many important references for anyone who wishes to study this topic.

2.1 Variants of educational timetabling problems

In education we can distinguish three types of timetabling problem: examination (Carter et al. [1996]), university (Arntzen and Løkketangen [2005]) and school (Colorni et al. [1998]). Each of these three problems has a slightly different focus and so different techniques are necessary for solving them.

Examination scheduling is the task of scheduling the exams of a set of university courses, avoiding overlaps of exams for students and spreading out the exams for students as much as possible. Generally, multiple exams can be scheduled in the same room at the same time providing seating capacity constraints are not exceeded. Also, there is usually more flexibility in the number of timeslots used.

University timetabling requires scheduling all the lectures of a set of university courses, avoiding overlaps of lectures containing common students. Generally, only one event per room is allowed and it will be over a fixed time period, such as a week or a fortnight.

The school timetabling problem consists of producing a schedule for all the classes required by a school, perhaps over a weekly time scale. The school timetable is affected by many parameters and must satisfy a large number of constraints. A possible constraint could be to avoid teachers being required to teach two classes simultaneously.

2.2 Graph colouring models

The nature of the timetabling problem is closely related to other combinatorial optimisation problems such as graph colouring.

Arguably the most common hard constraint in timetabling is to ensure staff, students and resources are not expected to be present at two or more events simultaneously. This is known as a 'first-order conflict' or, more informally, as a 'clash'. If this is the only constraint to be considered then the problem can easily be represented by graph colouring.

Graph colouring arises naturally in a variety of real-world applications, such as; timetabling (Sabar et al. [2009], Burke et al. [2007]), register allocation (Chaitin et al. [1981]), frequency assignment (Gamst [1986]), round-robin sports scheduling (Lewis and Thompson [2011]), crew scheduling (Gamache et al. [2007]), printed circuit testing (Garey et al. [1976]), satellite range scheduling (Zufferey et al. [2008]) and manufacturing (Glass [2002]). Note that this list is not exhaustive.

Graph-based techniques are often used to construct solutions to problems in a step-by-step manner. Improvement techniques are then employed to obtain a higher quality solution. The descriptions of some of these improvement techniques appear later in this chapter.

To convert a simple timetabling problem to a graph colouring problem, we first consider a simple and undirected graph, $G = (V, E)$. G comprises a set of n vertices $V = \{v_1, ..., v_n\}$ (representing the events) and a set of edges E, which join various pairs of different vertices representing the events that should not be scheduled in the same timeslot. The vertices are allocated to timeslots by means of colouring the vertices so that each colour represents a timeslot that is available in the timetabling problem. The task is to colour the vertices so no two adjacent vertices are assigned the same colour (timeslot) and the number of colours used does not exceed the number of timeslots available. A common objective is to minimise the number of timeslots used and therefore find the smallest number of colours k for a given graph G (its chromatic

number $\chi(G)$), such that G has a legal k-colouring. Identifying $\chi(G)$ is well known to be an NP-complete problem (Karp [1972]).

Perhaps the simplest approach to solve a graph colouring problem is a greedy heuristic. An example of this may be colouring each vertex in turn with the lowest indexed colour such that no violations are incurred, introducing new colours when necessary. The number of colours can differ depending on how the vertices are ordered however, there will always be at least one ordering that results in an optimal solution. Greedy heuristics tend to give worse solutions than some more complex heuristics.

The first comparison of timetabling and graph colouring occurred in 1966 where Peck and Williams [1966] presented the first colouring heuristic, 'Largest Degree First'. In this method, the vertex with the largest degree is coloured first. The degree of a vertex v in a graph G is the number of edges in G which have v as an endpoint. In terms of timetabling, this means first scheduling the event that has clashes with the largest amount of events. The first colour is selected and is assigned to as many vertices as possible by scanning down an ordered list of vertices (assigning as many events as possible to the first timeslot). Another colour is introduced and assigned to as many vertices as possible and so on using as many colours as needed until all vertices are assigned a colour. It may be possible to reduce the number of colours by using optimisation techniques, discussed later in this chapter. In 1967, Welsh and Powell [1967] also used this heuristic. However, the vertices were chosen to be coloured in index order, one-by-one, rather than scanning down an ordered list. This approach produced worse results.

Carter et al. [1996] considered five construction heuristics, as listed below. These were used to decide which event is next to be scheduled (vertex to be coloured). The authors assigned each event a value, calculated using one of the following criteria:

- Largest Degree (LD): chooses the next event to be scheduled by ranking the events in descending order by the number of clashes with all other events. Priority is given to the event with the greatest number of conflicts. The rationale behind this rule is that events

conflicting with many others are harder to schedule and so should be assigned first.

- Saturation Degree (SD): events are ordered, in ascending order, by the number of remaining feasible timeslots. This is a dynamic method that has to be recalculated after each iteration. This rule was originally proposed by Brélaz [1979]. It is argued that the most difficult event to schedule is the one that has the least flexibility in terms of choice of period. Ties are broken using the largest degree.

- Largest Weighted Degree (LWD): ranks the events in descending order by the number of students involved in clashes with all other events. Priority is given to the event with the largest degree. We might expect that this rule would give priority to core courses where large numbers of students have identical clashes.

- Largest Enrolment (LE): schedules the event next that has the largest number of students registered for the event. It is often the case that events with a large enrolment are difficult to schedule as they create more clashes.

- Random Ordering (RO): this rule is mainly considered for benchmark comparisons. If the above strategies have a significant impact on solution quality, then they should be better than just randomly selecting the next event.

Colour degree was also proposed by Brélaz [1979].

- Colour Degree (CD): prioritises those events that have the largest number of clashes with the events that have already been scheduled. This is a dynamic method that has to recalculate the number of clashes each time an event is scheduled.

It was observed in the literature that when being employed on its own, SD performs the best in most cases (Burke et al. [2007], Burke and Newall [2004], Carter et al. [1996]). This can be explained by its ability to dynamically order the events according to the number of remaining valid timeslots.

Lewis et al. [2012] reviewed the literature surrounding methods for the general graph colouring problem and presented a broad comparison of six high-performance algorithms. These included a TabuCol algorithm (Galinier and Hao [1997]), PartialCol algorithm (Blöchliger and Zufferey [2008]), Hybrid Evolutionary algorithm (HEA) (Galinier and Hao [1999]), AntCol algorithm (Dowsland and Thompson [2008]), Hill-climbing algorithm (Lewis [2009]) and back-tracking DSATUR algorithm (Culberson and Luo [1996]), which will be discussed further later in this chapter. A number of trials were conducted as a comparison on artificially generated graphs and also graphs from real-world problems. The real-world problems were namely, exam timetabling, social networks and round-robin scheduling. The results showed each algorithm out-performed all others on at least one occasion. Methods that relied solely upon local search methods, TabuCol and PartialCol, perform badly on spiky landscapes but fare better on flatter landscapes. These algorithms perform poorly in isolation but can perform better when used in conjunction with other search operators. The HEA was the most consistent algorithm and also relatively quick to run.

The graph colouring heuristics discussed above were widely studied in early timetabling research and are still being employed today as either; the initialisation method for metaheuristics, or through fully integrating them with metaheuristics in different ways. For instance, TabuCol (Hertz and de Werra [1987]) was the first application of tabu search to graph colouring in 1987. TabuCol has since been improved by several researchers and used as part of more elaborate colouring algorithms (Dorne and Hao [1999], Fleurent and Ferland [1996], Galinier and Hao [1999]).

Burke et al. [2007] present a hyper-heuristic approach using all six of the low-level heuristics described above in both exam and university course timetabling. A list of these heuristics is produced to determine the order in which the events should be scheduled. Tabu search is employed to search for the list of low level heuristics which are used to construct the complete solutions. A move in tabu search involves randomly changing two heuristics in the heuristic list. This new heuristics list is then checked against the tabu list and a 'failed list', which

is updated with the heuristic list if it cannot produce a feasible solution. Events are ordered using the current heuristic in the list and the first two events are scheduled. A steepest descent method is then used on the timetable in each iteration to improve the quality of the solution as quickly as possible. The steepest descent method tries to move events to different timeslots and terminates when no events can be moved that improve the quality of the timetable.

Qu et al. [2009] developed an adaptive hybridisation of these heuristics. The authors randomly generate a heuristic sequence of length n (n being the number of events). This sequence is initialised with all elements set to SD, since SD has been shown to provide good solutions. LWD, LE or LD are randomly hybridised into the sequence. Different rates (each with even chance) of hybridisation are tested. The i^{th} heuristic is used to order the remaining events and the event at the top of the list is scheduled first. A number (e.g. $n \times 50$) of sequences are tested and the soft constraint penalty is calculated and saved. If a feasible solution is not produced then the sequence is discarded. The results of these runs are then observed to develop an adaptive approach. The sequences are categorised into three groups; the best 5%, the worst 5% and the remaining sequences. The average appearances of LD, LWD and LE at each position are calculated and regression analysis is used to plot the trendlines. This analysis can then be used to intelligently hybridise the heuristics in the sequence.

Sabar et al. [2009] presented a constructive heuristic that the authors claim outperforms the early heuristics described above as well as several others, such as, fuzzy techniques (Asmuni et al. [2005]), neural networks (Corr et al. [2006]) and ant algorithms (Eley [2007]). The vertices are ordered in decreasing order of the number of clashes with other events. A 'roulette wheel' is then divided up into segments that represent the fitness of the event. Each event's fitness is calculated by the number of events it clashes with divided by the total number of overall clashes. These segments are positioned one after each other in the roulette wheel. A random number is then generated to determine which event should be scheduled next. The number of available clash-free timeslots for the event is calculated and, if it is greater than zero, it is scheduled in the timeslot contributing the minimum penalty. If there is more than one timeslot

with the same minimum penalty, one is chosen randomly. It is removed from the roulette wheel and the probabilities are re-calculated.

Lü and Hao [2010b] present a MACOL algorithm - a memetic algorithm for graph colouring. It consists of four main components:

1. Initial population generator - uses a randomised version of the DANGER colouring heuristic proposed by Glover et al. [1996]. In order to decide which uncoloured vertex should be coloured next the degree of danger for each uncoloured vertex is calculated. This gives an idea of how dangerous it is if vertex i is not coloured next and is defined by:

$$danger(i) = F(different\_colour(i)) + k_u \cdot uncoloured(i) + k_a \cdot \frac{share(i)}{avail(i)} \tag{2.1}$$

   where *different_colour(i)* is the number of different colours assigned to neighbours of vertex i; *uncoloured(i)* is the number of neighbours of vertex i that are uncoloured; *avail(i)* is the number of colours available to vertex i; *share(i)* is the number of colours available to vertex i that are also available to its uncoloured neighbours; F is the monotonic increasing function $F(y) = \frac{C}{(max\_colour-y)^k}$; k_u, where $max\_colour \neq y$, k_a and k are non-negative constants; *max_colour* is the largest colour index allowed and C is an arbitrary positive constant. However, Lü and Hao chose the next vertex and colour using a probabilistic version of their heuristic rather than based upon the 'dynamic vertex dangers measure'. If a solution is too similar to another in the population it will be discarded in order to build a diversified population.

2. Tabu search procedure - a conflicting vertex is moved from its original colour class to another. It is forbidden to move back for l iterations where $l = \mu \cdot f(S) + r(10)$. In this case, $\mu = 1$, $f(S)$ is the cost of solution S and $r(10)$ is a random number in the set $\{1, ..., 10\}$.

3. Multi-parent crossover operator - An adaptive multi-parent crossover operator (AMPaX)

is used. A legal k-colouring is a collection of k independent sets and the general idea is to maximise the size of the sets coloured k by a crossover operator, therefore, less vertices are left unassigned. Colour classes of offspring are built one by one resulting in k steps. AMPaX builds a colour class by considering the colour class with maximal cardinality of all parents ≥ 2. Vertices in that colour class are then removed from parents for the next step. Once a parent has been used to build a colour class it cannot be considered for a number of steps. This is repeated k times; if any vertices are left unassigned then they are assigned to a random colour class. Offspring are then improved by the tabu search algorithm described in step 2.

4. Population updating rule - the decision of updating the population, P, with an offspring colouring is based upon two factors: the diversity of the solutions and the solution quality. The distance, $D_{i,P}$, between a k-colouring, S_i, and a population member is calculated as the least number of 'moves' needed to transform the k-colouring to any other k-colouring in the population. The solution quality is $h_{i,P} = f(S_i) + e^{\beta/D_{i,P}}$ where $\lambda = 0.08$ (in this case) and $\beta = \lambda n$, n is the maximal value of $D_{i,P}$. A good quality solution will have a small value of $h_{i,P}$ and a poor quality solution will have a large value. If the quality of the offspring is worse than all parents, it is still accepted with probability of 0.2.

This method was compared with four local search algorithms and seven hybrid algorithms which cover the best known results for the 24 instances tested. The MACOL algorithm presented in this paper obtained better or equivalent results in all but two of the instances tested compared with the other algorithms.

Evolutionary algorithms have also been a popular approach for graph colouring in recent years. These include memetic algorithms (Burke et al. [1996b], Özcan and Ersoy [2005]), genetic algorithms (Dorne and Hao [1998], Ross et al. [1998]), multi-objective evolutionary algorithms (Côté et al. [2005], Paquete and Fonseca [2001]) and ant colony optimisation (Eley [2007]).

This conversion to a pure graph colouring problem only exists when considering first-order

conflict constraints. When other constraints are involved then this will add extra complications. However, it is still the case that nearly all timetabling problems will still feature this underlying graph colouring problem in their definition.

2.3 Problem instances

Problems vary between institutions but may also vary from one year to another, making it a difficult task for researchers to compare the performance of algorithms. Random datasets have been made available in the past that are useful in testing individual aspects of systems. However, they do not provide the complexity of real-life data so may not provide a good basis for deciding the methods that would perform best at a specific institution. Several generic problem models have been developed with publicly available instances over recent years, meaning that researchers have real-world models on which they can test and compare the performance of algorithms.

The first International Timetabling Competition took place in 2002. This is a competition based upon a reduction of a typical university timetabling problem in which a set of events need to be scheduled in 45 timeslots, subject to three hard constraints. The competition entrants were judged on firstly having a feasible solution and then penalised for violating any of the three soft constraints. The datasets were made available for competition entrants and have been publicly available online since the beginning of the competition[3], and have since been used by Chiarandini et al. [2003], Kostuch [2005], Lewis and Paechter [2004], Rossi-Doria and Paechter [2004], Socha et al. [2003], Yang and Jat [2011]. For instance, Lewis and Paechter [2004] present an evolutionary algorithm using a number of different problem-specific crossover operators that produce feasible offspring. The authors use saturation degree to form an initial solution. They then experiment with four crossover operators which encourage the useful groups of genes to produce fitter offspring and a genetic repair function is used to deal with unplaced events in

[3]http://www.idsia.ch/Files/ttcomp2002/

the gene transfer. A mutation operator is also implemented which swaps two genes provided no hard constraints are violated.

Bosquez et al. [2010] present a family of 29 sort then fix algorithms which schedule events beginning with the most constrained. Each algorithm defines the most constrained using different criteria. These algorithms ignore soft constraints but produce promising results for finding feasibility and could be used as a pre-processing step before other optimisation algorithms.

The second International Timetabling Competition was held in 2007[4]. The instances presented for this competition have since been widely used in research (Abdullah and Turabieh [2012], Lü and Hao [2010a], Nothegger et al. [2012], Turabieh and Abdullah [2011]). For example, Abdullah and Turabieh [2012] propose a memetic algorithm hybridising a tabu search algorithm and a genetic algorithm used for solving the instances from track one and track three of the competition.

The third International Timetabling Competition was help in 2011[5]. The purpose of this competition was to further research solely in High School Timetabling (Post et al. [2013]). There were four finalists in this competition, and they each used different techniques. Domrös and Homberger presented an evolutionary algorithm whilst Fonseca et al. presented an algorithm based on an Adaptive Large Neighbourhood Search. Kheiri at al. used Simulated Annealing and Iterated Local Search and finally, Sørenson et al. presented a hyper-heuristic to effectively exploit a suite of neighbourhood move operators. These methods were all described in short papers included in the conference proceedings from PATAT 2012[6]

Carter et al. [1996] introduced problem instances for examination timetabling that are publicly available[7,8]. These are often referred to as the Carter instances and have been utilised in a large number of research papers (Asmuni et al. [2005], Burke et al. [2007], Burke and Newall

[4]http://www.cs.qub.ac.uk/itc2007/
[5]http://www.utwente.nl/ctit/hstt/itc2011/welcome/
[6]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.308.4488&rep=rep1&type=pdf
[7]http://www.cs.nott.ac.uk/ rxq/data.htm
[8]ftp://ftp.mie.utoronto.ca/pub/carter/testprob/

[2004], Casey and Thompson [2003b], Di Gaspero and Schaerf [2001], Qu et al. [2009], Sabar et al. [2009]).

Benchmark datasets for graph colouring from The Centre for Discrete Mathematics and Theoretical Computer Science (DIMACS) Implementation Challenges are publicly available[9]. There have been ten DIMACS Implementation Challenges since 1993 that relate to graph theory. For instance, the tenth challenge focussed on the two related problems of graph partitioning and graph clustering, and the ninth challenge focussed on shortest path problems. A large amount of research has been conducted on graph colouring using these benchmark datasets (Blöchliger and Zufferey [2008], Dorne and Hao [1998], Galinier and Hao [1999], Lü and Hao [2010b], Malaguti et al. [2008], Porumbel et al. [2009]).

Problem instances relating to real-life data collected from Purdue University were introduced by Rudová and Murray [2003] and are now publicly available[10]. Much work has taken place to design and develop an intelligent system for timetabling at Purdue University (Murray et al. [2007], Rudová et al. [2011]).

Other datasets have been used over the years that are publicly available, such as, the University of Nottingham Benchmark Data, the University of Melbourne Benchmark Data and a Random Exam Timetabling Problem Generator, which are all available online[11].

Extensive work has also been carried out to study and compare different heuristics on specific timetabling instances. Colorni et al. [1998] construct a timetable of classes for an Italian high school as a benchmark for their investigation. Dowsland [1998] examines solutions to three real life scheduling problems in Swansea, Wales; a laboratory scheduling problem at the Business Management School, the problem of examination scheduling at Swansea University and the problem of nurse rostering at a local hospital.

[9] http://dimacs.rutgers.edu/Challenges/
[10] http://fi.muni.cz/ hanka/purdue_data/
[11] http://www.cs.nott.ac.uk/ rxq/data.htm

2.3.1 Constraints

This section will outline some of the main constraints that educational institutions may face.

A common hard constraint amongst all institutions is the requirement to avoid any student being required to attend two or more different events at the same time. It is often the case that additional constraints to the 'clash' constraint are imposed by an institution. These are categorised as either hard or soft constraints. Hard constraints must be satisfied to provide a valid solution. For example, the room allocated for an event needs to be big enough to accommodate all students expected to attend. Soft constraints can be violated but should be obeyed wherever possible to provide a more desirable solution. An example of a soft constraint may be to minimise the number of students attending events in consecutive timeslots. A constraint that specifies that a student cannot attend events that are 'close' to each other is known as a 'second-order conflict' and is usually considered a soft constraint.

In general, the problem of educational timetabling is NP-complete (Asratian and de Werra [2002]). To obtain a practical timetable in reasonable time depends on the nature of the problem instance being tackled. Requirements and resources vary between universities, therefore some problems are inevitably easier to solve than others. For example, some universities may have an abundance of rooms or a small number of events to schedule. In the case of problems that are more difficult to solve, it is possible that only a small proportion of the search space will be occupied by feasible timetables. It is even possible that a feasible timetable is impossible due to the combination of constraints, resulting in the need for the relaxation of some constraints. Thus, there seems an implicit need for powerful and robust methods for tackling these sorts of problems.

Although some timetabling considerations are common across most universities, specific requirements vary from one to another as typically constraints are unique to each institution. Burke et al. [1996a] carried out a survey on examination scheduling in British universities, focussing on the constraints of examination scheduling. Many of the constraints discussed are

also relevant in university course timetabling. Of the 95 surveys distributed, 56 institutions replied. Figure 2.1 lists 18 constraints that the institutions gave importance to. Although some of them may not be relevant in university course timetabling, the diversity and complexity of the problem is evident. These constraints can vary in importance between institutions and could be categorised as a hard constraint for one institution and a soft constraint for another.

Figure 2.1: A list of possible constraints that educational institutions may impose.

There must not be more students scheduled to a room than there are seats.
Some events may only be scheduled within a particular set of timeslots.
Events with the most students must be scheduled early in the timetable.
Some events must only take place in particular rooms.
Large rooms must be scheduled in preference to smaller ones.
Event A must be scheduled before event B.
No student is scheduled events in two consecutive timeslots.
No student is scheduled more than one event in any particular day.
Each students events must be evenly spread throughout the timetable.
No student is scheduled events in two consecutive days.
Events must be scheduled to rooms in the relevant department.
Specific rooms must be provided for disabled students.
Religious convictions must be respected.
Time must be provided for students to travel between sites.
Availability of part time students should be respected.
One event must precede all others in a group.
One event must be the last in a group.
Other specific departments preferences.

In the early years of researching automated timetabling, the computational capacity did not exist to consider great numbers of constraints. For instance, in the case of Broder [1964] only first-order conflicts are considered. Cole [1964] proposes an algorithm for producing a solution to a problem consisting of four constraints:

1. The events corresponding to a certain subject either *must* or *must not* occur in consecutive time periods.

2. No event of subject A must precede any event of subject B.

3. The events of subject C must all appear in odd (morning) time periods unless they are also required to be consecutive, in which case the first event must be in an odd period.

4. The total number of students sitting events in any one period is either bounded above or is restricted by the accommodation available and all students sitting the same event must be in the same room.

Since then, with the progression of computer capability, many authors have considered more constraints that apply to real-life problems, while also making the timetable 'nicer' by considering soft constraints.

2.4 Optimisation

Lewis [2008] suggests that metaheuristic algorithms for timetabling can be loosely separated into three main classes: one-stage optimisation algorithms, multi-stage optimisation algorithms, and algorithms that allow relaxations. The author conducted a comprehensive survey of the field of metaheuristics and timetabling using these three categories.

2.4.1 One-stage optimisation

In general, single stage approaches allow the violation of both hard and soft constraints, and searches for a solution that can adequately satisfy both. This is usually achieved through weighting the constraints in order to give the hard constraints much higher penalties when violated than soft constraints. The search will be attempting to find feasibility and optimality at the same time.

Carter et al. [1996] developed one of the most prominent one-stage algorithms. The authors developed EXAMINE, a computerised examination scheduling system. Their paper describes how EXAMINE is used to carry out many experiments, on both real life and randomly gener-

ated problems, to determine the best algorithm for solving timetabling problems by considering only first-order conflicts. They first find the largest clique in the underlying graph colouring model, using the Carter and Gendreau algorithm (Carter and Gendreau [1984]), since the length of the schedule is at least as large as the size of any clique. They schedule this clique first and then use five sorting criteria to schedule the remaining events. This method of finding the largest clique first works well for real problems but was not as successful for random problem instances because they have no structure and usually have a small maximal clique. Generally, fewer timeslots were needed on most instances, particularly harder problems. Backtracking was used, which removes events already scheduled in order to schedule an event that is in conflict with every timeslot. This reduces the overall length of the schedule by an average of 50%. Proximity costs w_s were then introduced, which are incurred whenever a student has to attend two examinations scheduled s time periods apart. The weights imposed are as follows: $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$ and $w_5 = 1$. Using these weights, cost values are evaluated for each student and then summed to give a total cost accumulated for all students. When the proximity cost is used, the number of time periods increased significantly; 'spreading out' the events has a detrimental impact on the solution quality. SD used the minimum number of time periods in most cases. For comparison, the authors also ran tests to minimise the (proximity) cost per student using backtracking, having fixed the length of the schedule to the minimum value for which each strategy was able to find a feasible solution. Again, SD was superior in terms of solution quality and computing time.

Di Gaspero and Schaerf [2003] also use weightings in the objective function to penalise violations of hard constraints. The one-stage algorithm begins with a random initial timetable with some hard constraints relaxed. They experiment with various combinations of different neighbourhood operators. These combinations can be; a neighbourhood union which selects moves from any neighbourhood considered in the union, a neighbourhood composition which performs an ordered sequence of moves from different neighbourhoods, or a token-ring search which performs each neighbourhood in turn starting from the best solution found by the previ-

ous neighbourhood. A kick operator is used to avoid the search getting stuck at a local optima. The authors apply this approach to the course timetabling problem.

2.4.2 Multi-stage optimisation

Multi-stage approaches were introduced later, an early example was provided by Thompson and Dowsland [1998] in 1998. The first stage aims to produce a timetable that obeys all hard constraints, therefore producing a feasible timetable. The further stages then aim to minimise any soft constraint violations.

Kostuch [2005] describes an algorithm, submitted to the first International Timetabling Competition, that consists of three stages. Stage one attempts to construct a feasible initial timetable using graph colouring and a maximum matching algorithm. The solution is penalised if a student has an event in the last slot of the day, therefore the five penalised end-of-day slots were left empty. Stage two uses simulated annealing, see Section 2.6.1.2, to sequence the timeslots created in stage one to reduce soft constraint violations and optimise the schedule. All possible timeslot swaps are tested in a deterministic order and feasibility is maintained at all times throughout this stage. Finally, stage three uses simulated annealing to swap individual events between timeslots whilst maintaining feasibility. This stage takes the majority of computational time but has the most significant impact on the objective function.

Gunawan et al. [2012] also presents a three phase approach to solving the university course timetabling problem that consists of a pre-processing stage, a construction phase and an improvement phase. The pre-processing phase produces a set of teachers willing to teach each course and a set of time period preferences for each teacher. The construction phase employs a Lagrangian relaxation approach to produce an initial feasible solution. The improvement phase uses simulated annealing with the concept of a tabu list to re-allocate teachers and find a new set of days and timeslots to improve the initial solution.

Other examples include Arntzen and Løkketangen [2005], Casey and Thompson [2003a] and

Socha et al. [2003].

2.4.3 Algorithms that allow relaxations

In order to temporarily simplify a problem, some aspects of it may be relaxed whilst ensuring that hard constraints are never violated. Attempts are made to minimise the number of soft constraint violations, whilst also giving consideration to the task of eliminating these relaxations. A typical relaxation is to include additional timeslots to schedule events and then attempt to reduce the number of timeslots at a later stage.

For example, Paechter et al. [1998] successfully investigate a university course timetabling problem using an evolutionary algorithm. An indirect representation can be described in two parts. The first is a permutation that determines the order in which the events should be scheduled. The second specifies a number of suggested timeslots for each event. An attempt is made to schedule the first event in the first suggested timeslot. If this is an unsuitable timeslot because it violates some hard constraints, then the next suggested timeslot is used, and so on. If there are no suitable suggested timeslots then a problem-specific heuristic is used to specify extra timeslots which do not incur a penalty. If the event is placed then the timeslot used is written back into the chromosome as the primary suggested timeslot. If the event still cannot be scheduled whilst obeying all hard constraints then it is considered unplaced. To reduce the number of unplaced events, the rule 'all events must be placed' is treated as a soft constraint. The soft constraints can then be weighted by the user and changed throughout a run. Targets can be set for each soft constraint to an acceptable level so that improvements are made in the appropriate areas once a target has been reached.

Another example of an algorithm that allows relaxations is proposed by Ceschia et al. [2012] for solving the post enrolment-based course timetabling problem. The hard constraints that must be satisfied are; the rooms must be compatible for the scheduled event and only one event is scheduled per pair of rooms and timeslots. The constraints that are relaxed are; conflicts

between events, the precedence constraint and events may be left unscheduled. The algorithm begins by creating an initial solution using a greedy algorithm. The greedy algorithm attempts to schedule events in a randomly selected timeslot and room. If this combination of room and timeslot violated any of the constraints (excluding the relaxations), then another pair is chosen and so on for a finite number of iterations. If this is unsuccessful then the event will remain unscheduled. Simulated annealing is used with neighbourhood operators, moves and swaps, to improve the solution quality and minimise the number of violations of the relaxed constraints.

There have been many different algorithms developed in order to attempt to minimise the number of soft constraint violations. In the next section we will outline some of the key approaches.

2.5 Local search

Local search is a method for producing good, but not necessarily optimal, solutions to combinatorial problems. In general, a combinatorial optimisation problem has a discrete finite search space S, and the quality of each solution contained in the search space can be measured by some function f. The solution with the best quality is called the global optimum. The neighbourhood $N(s)$ of a solution s in S is defined as the set of solutions which can be obtained from s by a move. Each solution $s\prime \in N(s)$ is called a *neighbour* of s.

Local search commences from some starting solution and selects a neighbour. The two solutions are compared and the new solution is accepted if it improves the cost function.

Two simple neighbourhoods that are frequently used are the move neighbourhood and the swap neighbourhood (Cambazard et al. [2012], Chiarandini et al. [2008], Kostuch [2005]). The move neighbourhood consists of moving events one at a time to different timeslots. The swap neighbourhood involves swapping the timeslots of two events. In minimisation problems, these moves and swaps are accepted if the value of the cost function is reduced and no hard constraints

are violated. The swaps neighbourhood can be extended to a timeslot swap neighbourhood, where the contents of two timeslots are swapped.

The Kempe chain neighbourhood was proposed by Morgenstern and Shapiro [1986] as a suitable neighbourhood for graph colouring. It has since been a popular local search method for timetabling problems. Kempe chains are sets of mutually conflicting events between two timeslots, these events timeslots can be interchanged without introducing conflict. Thompson and Dowsland [1998] use the Kempe chain neighbourhood in a simulated annealing implementation and showed that it achieved superior results over other neighbourhoods for the timetabling problem under consideration. The Kempe chain neighbourhood increases the connectivity of a search space and can therefore move large events with many clashes to attempt to reduce the cost function.

2.6 Metaheuristics

The Metaheuristics Network defines a metaheuristic as:

> 'A set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem.'[12]

Metaheuristics are widely used to solve important practical combinatorial optimisation problems. This section will discuss some of the applications of these metaheuristics.

[12]http://www.metaheuristics.net/index.php?main=1

2.6.1 Local search-based metaheuristics

Local search-based metaheuristics find good solutions by iteratively making changes to a single solution. These include tabu search, tabu threshold, simulated annealing and variable neighbourhood search. Some of these are explained in this section.

2.6.1.1 Tabu search

Tabu search was introduced by Glover [1986, 1989]. It uses a memory property so that when a move is performed its inverse is considered tabu for a number of iterations. This ensures that the search explores new areas of the search space and avoids revisiting solutions. This is usually done in the form of a tabu list of length l. After each move, the inverse of that move is added to the end of the tabu list and is a forbidden move for l iterations. In addition to this, an aspiration criterion is used whereby the search can override the tabu list if the move would obtain a new best solution. Tabu search also considers the best current move that is not on the tabu list regardless of whether it is improving or not. Therefore, once a local optimum has been reached only moves that degrade the solution are in the immediate neighbourhood, and the solution moves away from the local optimum.

The tabu list is a short term memory that remembers forbidden moves for a certain amount of iterations. The number of iterations for which these moves remain on the list is called the tenure. It is a critical parameter that greatly influences the performance of the method. If the tenure is too large it can restrict the search and if the tenure is too small there is a risk of the search cycling. The tabu tenure can be either static or dynamic. When the tabu tenure is static, the number of iterations for which a move is prohibited is fixed, as in Galinier and Hao [1997]. Dynamic tabu tenure varies throughout the search. A study by Montemanni and Smith [2001] compares a dynamic tenure with a static tenure and concludes that a dynamic tenure produces better results. The dynamic tenure that they present reduces the length of the tabu list in the same way as it is done for the temperature parameter in a simulated annealing

algorithm. After a number of iterations the tenure T is reduced by calculating $T = \beta T$ where $0 < \beta < 1$ and is defined by the user, as is the initial value of T. When T is reduced, the oldest moves which exceed the new length of the list become feasible.

Di Gaspero and Schaerf [2001] propose a tabu search for the examination timetabling problem. The algorithm is similar to the TabuCol algorithm proposed by Hertz and de Werra [1987] but with additional constraints represented with an edge-weight function that represents the number of students involved in a 'clash' and a node-weight function that indicates the number of students enrolled in each examination. The results are compared against existing literature on the Carter instances and are considered encouraging.

2.6.1.1.1 The TabuCol algorithm

TabuCol was proposed by Hertz and de Werra [1987] for graph colouring. Since then it has frequently been used as a local search operator in hybrid algorithms (Dorne and Hao [1998], Galinier and Hao [1999], Galinier et al. [2008]), although experiments reported by Blöchliger and Zufferey [2008] suggest that this method can obtain successful results when used independently. TabuCol operates in a search space of complete improper k-colourings (solutions with all events scheduled but may include clashes). It involves moving events that are involved in clashes to other timeslots using a neighbourhood operator. A 'tabu list' is maintained which forbids reassigning the events to the timeslots they have just left; they remain on the list for a number of iterations. The number of iterations is called the tenure. This avoids cycling and helps to escape from local optima. Moves are considered deterministically. An event is moved to the timeslot that is not tabu, that is, not on the tabu list, and makes the best improvement to the cost function. If the cost cannot be improved then the move incurring the smallest increase is accepted and any ties are broken randomly. A move that is tabu is accepted if it achieves the best solution found so far.

2.6.1.1.2 The PartialCol algorithm

PartialCol (Blöchliger and Zufferey [2008]) operates in a similar fashion to TabuCol but considers partial solutions where some events are unscheduled rather than improper solutions where all events are scheduled but may include clashes. The aim is to schedule any unplaced events whilst maintaining feasibility and minimising the number of soft constraint violations. Blöchliger and Zufferey [2008] proposed Foo-PartialCol, a local search approach to the graph colouring problem. The Foo-scheme is a reactive tabu tenure based on the fluctuation of the objective function. The tenure is increased if little improvement has been made over a period of time as it is assumed that the search has stagnated in a particular region of the search space. To counterbalance this, the tenure is reduced slowly along the search process. If a large change to the objective function has been made over a certain time period then the tenure is decreased by one (but is forbidden to be negative). PartialCol is a local search method with a search space that consists of partial k-colourings. An initial partial colouring is created by a greedy algorithm. A vertex is inserted into a colour group with the lowest given number, provided that there are no clashes. If this is not possible, it is inserted into an uncoloured list. The cost function is defined as the number of vertices in the uncoloured list. Once an attempt has been made to colour all vertices without conflict, the vertices on the uncoloured list are inserted into the colour group that causes the least amount of clashes. Any other vertices that are then causing a clash will be removed and placed on the uncoloured list. This is a very simple and efficient algorithm and obtained competitive results on a large sample of benchmark graphs.

2.6.1.2 Simulated annealing

The idea of Simulated Annealing (SA) comes from a paper published by Metropolis et al. [1953]. The authors present an algorithm which simulates the cooling of material in a heat bath. This is a process known as annealing. The algorithm simulates the cooling process by gradually

lowering the temperature of the system until it converges to a steady state. Kirkpatrick [1984] took the idea of this algorithm and applied it to optimisation problems. The idea is to use SA to investigate feasible solutions and converge to an optimal solution.

SA accepts all moves that improve the quality of the solution, it also accepts some worsening moves to avoid being trapped in a local optimum. It accepts these worsening moves with a probability of:

$$e^{-\Delta D/T} \tag{2.2}$$

where ΔD is the change in solution quality. This will be negative for an improving move and positive for a worsening move. T is the temperature, if T is large then many worsening moves are accepted and a large part of the search space is accessed. Usually, the temperature is lowered after every z iterations. The search stops when equilibrium has been reached and no worsening moves are accepted.

This method was successfully used by competition entrants. The winner of the first International Timetabling Competition, Kostuch [2005], used simulated annealing after the initial solution has been constructed. Timeslot swaps and swaps were used to optimise the solution whilst maintaining feasibility. The winner of the second International Timetabling Competition, Cambazard et al. [2012], also used simulated annealing with a number of neighbourhood operators to optimise the solution, see Section 2.7 for a more detailed description.

A two-stage approach was proposed by Thompson and Dowsland [1998] who used simulated annealing to produce a feasible schedule, and then used the same method to minimise second-order conflict, while maintaining feasibility. A neighbourhood based on Kempe chains was shown to improve solution quality. A similar method was employed by Merlot et al. [2003] who produced high quality results by using a slower cooling schedule.

Tarawneh et al. [2013] present an algorithm for solving the curriculum-based course timetabling problem. An initial solution is constructed using a sequential greedy heuristic having relaxed HC5. A steepest descent method is then used to find feasibility. To minimise the number of

soft constraint violations the authors use simulated annealing with memory. The memory saves moves that are not accepted and once simulated annealing has reached a local optimum the search employs one of the saved solutions to escape.

2.6.1.3 The great deluge

The great deluge algorithm was first introduced by Dueck [1993] and is similar to simulated annealing described in Section 2.6.1.2. McMullan and McCollum [2007] believe that the great deluge algorithm is more effective than a simulated annealing algorithm to avoid being trapped in local optima. The great deluge algorithm uses a boundary condition rather than a probability measure with which to accept worse solutions. For a minimisation problem, the boundary is initially set to a value higher that the expected penalty of the best solution. Then, the boundary is gradually decreased throughout the improvement process.

Abdullah et al. [2012] present an algorithm for solving the post enrolment-based course timetabling and curriculum-based course timetabling. Three phases are used to construct an initial solution; firstly, largest degree as described in Section 2.2 is used, phase two consists of the move and swap neighbourhood operators and phase three uses tabu search if a feasible solution has not yet been generated. To optimise the solution the authors present a hybridisation of an electromagnetic-like mechanism and the great deluge algorithm. The electromagnetic-like mechanism is used within the great deluge to calculate the decreasing rate.

2.6.2 Population-based metaheuristics

Evolutionary algorithms such as memetic algorithms, genetic algorithms, multi-objective evolutionary algorithms and ant colony optimisation are detailed in this section.

2.6.2.1 Genetic algorithms

Genetic algorithms are inspired by Darwin's theory of evolution. They are a popular method for solving timetabling problems (Burke et al. [1995], Safaai et al. [1999], Dorne and Hao [1998], Ross et al. [1998]). An initial set of solutions called the (parent) population are created. These solutions are then used to form a new population (the offspring). The solutions chosen for reproduction are chosen using a fitness criteria. The higher the fitness, the more chance they have of being chosen to reproduce. The offspring are produced using a crossover operator which select the chromosomes of each parent to use to create the offspring. The offspring can then be mutated which should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution.

Pillay and Banzhaf [2010] present the results of an informed genetic algorithm. Their approach involves two stages; stage one consists of applying a genetic algorithm (GA) to find a feasible solution and then applying a GA to minimise the soft constraint cost in stage two. In stage one, three ways of constructing the timetables for the initial population were tested:

1. Choose a random event and schedule it in a random timeslot;
2. Choose a random event and allocate it to the period that would incur the minimum cost;
3. Choose the most difficult event to schedule (determined using heuristics) and allocate it to the minimum cost period.

Method 2 produced timetables with lower cost than method 1, however, method 3 was the quickest to converge to a feasible timetable and most heuristics tested also produced timetables with lower soft constraint costs. Using SD gave the lowest run times and using Highest Cost (HC) gave the lowest soft constraint cost. HC is defined as the cost of scheduling an event in terms of its distance from events that it has students in common with. The authors decided to use a combination of SD and HC and to break ties using other low level heuristics of which

LWD and LE were shown to be the best. Tournament selection chooses the parents of the next generation. This randomly selects n elements, called the tournament, from the population and returns the timetable with the fewest number of clashes, if more than one then the soft constraint cost is used to break ties. The mutation operator then reschedules one or more events involved in clashes to the minimum cost period. Stage two parents are again chosen using tournament selection and a similar mutation operator is used. One or more events are rescheduled to ensure a feasible timetable is produced. This is repeated until the timetable is at least as fit as the parent timetable. The number of events to be rescheduled is chosen randomly between one and the maximum number of changes allowed. The results show that this algorithm is not only comparable to other evolutionary algorithms but also to different techniques applied to the Carter instances.

Although pure GAs such as this have shown some promise, most researchers have found that combining them with some form of descent method gives significant improvement. This combination is known as a hybrid algorithm or a memetic algorithm.

2.6.2.2 Hybrid genetic algorithms

Hybrid methods combine good characteristics of different metaheuristics e.g. a memetic algorithm (Moscato [1989]) that combines a GA with local search.

Abdullah et al. [2007] present an evolutionary algorithm together with local search called a memetic algorithm (Burke et al. [1996b], Özcan and Ersoy [2005]). A random graph colouring method is used to create an initial population of size 100. Roulette wheel selection, as described in Section 2.2, is used to select individuals for the next generation and a random light mutation (small alteration) is employed on 20% of the courses from 20% of the selected individuals. Local search is then applied to these solutions and added to the population and may be used in the next generation. This is repeated until there is zero cost or after 30 generations. This produces the best known results in the literature at the time of publishing for all but three of the datasets

from the first International Timetabling Competition.

Jat and Yang [2011] propose a two-phase approach. The first phase applies a guided search genetic algorithm followed by a second phase which uses a tabu search with local search to optimise the solution.

An overview of memetic algorithms for scheduling and timetabling problems can be seen in Burke and Landa Silva [2005].

2.6.2.3 Ant colony optimisation

Ant colony optimisation was first proposed by Dorigo et al. [1991b] and is used for solving hard combinatorial optimisation problems. It has been used for solving problems such as the travelling salesman problem (Dorigo et al. [1991a]), vehicle routing (Gambardella et al. [1999]) and scheduling (Merkle et al. [2002]) and many others.

Ant colony optimisation takes inspiration from ants finding the shortest route to a food source. When ants are walking to and from the food source they deposit pheromones that gradually evaporate over time. Ants have a probability of following the pheromone trail proportional to the strength of it. Initially, each ant randomly chooses a route to a food source. Due to random fluctuations one route will develop a stronger pheromone trail than the others and therefore attracts more ants. If the routes are not an identical distance to a food source then there is another factor to consider. The ants that randomly choose the shortest route will be the quickest to return to the nest and therefore this route receives pheromones earlier than other routes and it is then more likely that ants will choose this route over others. As more ants follow the shorter path the pheromone trail strengthens until no ants follow the longer route.

Deneubourg et al. [1990] investigated the behaviour of Argentine ants. The nest and a food source were joined by two paths identical in length. The ants initially chose a path at random. However, due to random fluctuation, one path eventually had a stronger pheromone trail than the other so all ants chose that path. This experiment was repeated many times with each of

the paths being used roughly 50% of the time.

An ant colony optimisation algorithm called AntCol, proposed by Costa and Hertz [1997], for colouring graphs was first introduced in 1997. Dowsland and Thompson [2004] successfully applied an algorithm based on AntCol to the examination scheduling problem. It is modified to deal with conflicting constraints and secondary objectives such as time windows and seating capacities. This adapted algorithm was able to find the best-known colourings quickly and consistently over a wide range of examination timetabling graphs. The algorithm was again improved in 2008 by Dowsland and Thompson [2008] by introducing two new evaluation functions based on the chromatic number. The authors show that the improved version of AntCol can be generalised successfully to give enhanced performance on arbitrary graphs. They obtain further improvement in solution quality by strengthening the diversification strategy previously suggested.

2.7 Submissions to track two of the ITC 2007

The research undertaken for this thesis focusses on Track two: Post enrolment-based course timetabling of the International Timetabling Competition 2007. The submission by Cambazard et al. [2012] was the winning entry from thirteen submitted entries.

Two approaches were studied by Cambazard et al., both comprising two stages. The first approach used local search to obtain a feasible solution from a randomly generated solution. The neighbourhood consisted of:

- moving an event to an empty position in the timetable;
- swapping the position of two events;
- swapping the positions of all events contained in two timeslots;

- a matching algorithm which reassigns the rooms to the events in a timeslot to minimise the number of unsuitable rooms allocated to events;
- a combination of moving an event to a different timeslot and the matching algorithm to insert it without violating the room constraints.

A more complex move based on the Hungarian method for solving assignment problems is used as a greedy intensification procedure. Cycling is avoided by maintaining a simple tabu list. The second stage begins once feasibility is established. Moves and the matching algorithm are used, whilst preserving feasibility, to minimise the number of soft constraint violations. Simulated annealing was introduced to obtain better results. The second approach also uses local search but on a colouring relaxation problem. The room constraints are relaxed to provide a list-colouring problem. Moves, swaps, timeslot swaps and the Hungarian-based method are used to eliminate the remaining hard constraint violations. Moves are used at this point to minimise the number of soft constraint violations. The local search solver described above is used to attempt to eliminate any room violations. The soft constraint violations are then minimised using the soft constraint solver, also described above. The authors concluded that local search-colouring was best for finding a feasible solution. If the problem was highly constrained, simulated annealing-colouring was best for optimising the soft constraint violations, otherwise simple simulated annealing worked best.

The solver submitted by Atsuta et al. [2007] was ranked second in track two of the competition. This general problem solver consists of a hybrid algorithm of tabu search and iterated local search using the shift neighbourhood. Constraints are weighted and the weight are dynamically controlled to improve performance.

A modular multi-stage heuristic solver by Chiarandini et al. [2008] was ranked third in the competition. It consisted of two stages, the hard constraint solver and the soft constraint minimiser. The hard constraint solver attempts to find a feasible solution by scheduling the event with the smallest number of possible time periods; if there is more than one event with this minimum

then one is chosen at random. The event is scheduled in the timeslot that can accommodate the fewest unscheduled events. An exact matching of the lectures and rooms is solved to make a feasible insertion; if it cannot be solved then the event is left unscheduled. If any events remain unscheduled after a sufficient number of repetitions, a tabu search procedure is used based on the PartialCol algorithm proposed by Blöchliger and Zufferey [2008] described in Section 2.6.1.1.2. An unscheduled event with the largest number of students attending it is chosen to be scheduled next. It is scheduled in the best timeslot and any events that then result in a hard constraint being violated are removed. The tabu search on the unscheduled events and a soft constraint optimiser are alternated until a feasible solution is found or a given time limit is reached. The soft constraint minimiser consists of a variable neighbourhood descent in which moves, swaps, timeslot swaps and Kempe chains are run until no further improvements to the number of soft constraint violations can be made. An exact matching is used wherever needed. Moves and swaps with the exact matching are used with simulated annealing until a given time limit is reached.

Nothegger et al. [2012] were ranked fourth in the competition using ant colony optimisation. The solver submitted by Nothegger et al. also contains two stages, an initial solution construction and an improvement stage. The initial solution construction procedure considers events in a uniform random order. It assigns them to a feasible period in a greedy randomised way taking pheromones into consideration. The improvement stage attempts to move the most costly event (in terms of the number of soft constraint violations) to a different timeslot, whilst removing at most one event to maintain feasibility. This improvement heuristic is applied to all events which violate soft constraints.

The solver submitted by Müller [2009] to the competition was submitted to all tracks; examination timetabling, post enrolment-based course timetabling and curriculum-based course timetabling. It was ranked fifth for track two and was the winning submission for tracks one and three. The construction stage uses an iterated forward search algorithm (Müller et al. [2005]). Each iteration consists of randomly choosing an event among the unassigned variables

with the smallest ratio of the number of possible timeslots to the number of hard constraints. It is then assigned to a timeslot that increases the cost function the least and violates the smallest number of hard constraints, any ties are broken randomly. If clashes do occur then the scheduled events are removed. Once a complete solution has been found, hill climbing is used to find a local optimum. A problem specific neighbourhood is randomly selected and is used to generate a random change in the current solution. Once at a local optimum, the great deluge technique is used to widen the search and try to improve the solution further. When the bound has reached its lower limit, simulated annealing is then employed and the temperature is increased when no improvement has been made for a number of iterations. Then the hill climbing stage is restarted. This continues until a given time limit is reached.

2.8 Highly constrained problems

If a problem is highly constrained, then the problem has relatively few, if any feasible solutions. As a result, when exploring the space of feasible solutions the search space is likely to be poorly connected or even disconnected. We will therefore need to consider such a problem differently and the heuristics used will need to be suitable. This chapter will cover some of the methods that have been used to solve a range of highly constrained problems. Although these problems might not be university course timetabling it is often the case that methods can be adapted to also solve timetabling problems.

Local search heuristics operate in a search space, S, and the landscape of this space for a local search method will depend on the particular neighbourhood and cost function being used. For every solution $s \in S$, a neighbourhood $N(s) \subset S$ is defined. A local search method starts at an initial solution and then moves repeatedly from the current solution to a neighbour solution in order to try to find better solutions. The quality is measured by an appropriate objective function. In highly constrained problems the feasible areas of the search space may be very small and few and far between.

Burke et al. [1996b] use a memetic algorithm to solve the exam timetabling problem. They found that the initial results were promising but it transpired that the algorithm does not perform quite as well on more highly constrained problems as some other methods such as Burke et al. [1995]. Burke et al. use a hybrid genetic algorithm for solving highly constrained timetabling problems. A random sequential graph colouring algorithm is used to produce a starting population. This may use more timeslots than the optimal amount. A genetic algorithm then evolves new timetables and can reduce the length of the timetable if possible. This approach guarantees a feasible timetable and there is no risk of creating a search space that only contains infeasible solutions.

Burke et al. [2010] combine integer programming (IP) and variable neighbourhood search for the nurse rostering problem. The problem can be formulated as an IP model introducing slack and surplus variables in the soft constraints. The multi-objective heuristic model can reduce the number of constraints by summing some of the slack/surplus variables so that the objective function can consist of sub-functions. Once the IP model has been solved variable neighbourhood search is used to refine the solution by swapping groups of consecutive shifts. The authors believe this method can be applied to other resource allocation problems, not only nurse rostering.

Xu and Qu [2012] present the first fitness landscape analysis on the delay-constrained least-cost multicast routing problem, a well-known NP-hard problem. To analyse the landscape the authors used two widely used landscape analysis measures; the fitness distance correlation and the autocorrelation analysis. Different delay bounds were added to a benchmark instance so that large amounts of simulations could be carried out. The landscape analysis reveals that the landscape is instance dependent and tailored algorithms may not work well on different instances. Therefore, adaptive and robust search methodologies are needed to obtain reliable and good results across instances.

Many authors describe the benefits of using evolutionary algorithms for solving a range of highly constrained problems. Colorni et al. [1991] also present a general methodology to apply

genetic algorithms (GA) to highly constrained combinatorial optimisation problems, and the timetabling problem in particular. The authors state 'The main goal of our research is to understand GA's limits and potentialities in addressing highly constrained problems'. The authors present sample results as they are still to be validated at time of publication however, they are said to be promising results.

Bufé et al. [2001] use a hybrid approach of an evolutionary algorithm and local search using specific mutation operators as both of these approaches have shown to be successful individually in the past. Genotype operations are used followed by replacing the genotype mutation with the phenotype mutation. This algorithm provided encouraging results.

Ray et al. [2000] present strategies to handle highly constrained problems. An evolutionary algorithm is used for a constrained multi-objective optimisation problem. It was shown that a constraint-constraint-mating scheme is effective for highly constrained problems. The algorithm was shown to be encouraging on both single and multi-objective problems.

2.9 Chapter summary

In this chapter, the literature on the various timetabling problems was introduced which included a brief history of the progress and expansion over the last 50 years. It has outlined the challenges of the problem and the methods which have been used to attempt to solve a range of optimisation problems.

It is shown that the early research into timetabling problems and its variants centred on exact methods; however their limitations with regards to problem size and the complexity of real-life problems quickly became apparent. The methods then evolved to include local search methods and metaheuristic approaches. In this thesis we will be investigating the best means of applying local search-based methods to the post enrolment-based course timetabling problem.

This chapter has highlighted the main heuristic approaches taken to solving the post enrolment-

based course timetabling problem. It has been shown that many techniques involve multiple stages which usually consist of a construction phase followed by optimisation stages. This approach will be followed within this research.

Chapter 3

Stage one of a two-stage approach: Finding a feasible solution to the University Course Timetabling Problem

This thesis investigates how university timetables can be constructed in an automated way, relating specifically to the complex and difficult task of university course timetabling. The problem is exacerbated by the fact that each university will have unique constraints; for example, part-time lecturers and the availability of resources and equipment. It is desirable to determine an algorithm that could be used in all cases, however this is a very difficult task as all problems are unique.

The aim of the thesis is to investigate a range of heuristic and metaheuristic approaches for solving the course timetabling problem. We investigate the best means of applying local search to attempt to find a successful method that can be adapted to fit other criteria.

When dealing with timetabling problems that are subject to both hard and soft constraints,

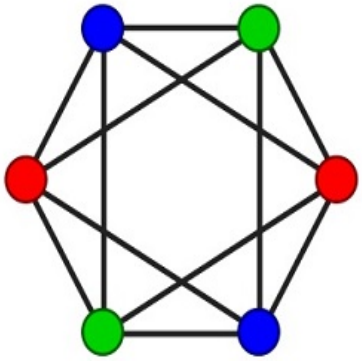
a common strategy is to make use of a two-stage (or even multi-stage) optimisation process (Chiarandini et al. [2008], Gogos et al. [2010]). An initial stage constructs a feasible solution, followed later by stage(s) that further improve the timetable. This approach can be regarded as a variation of the Greedy Random Adaptive Search Procedure (GRASP) metaheuristic (Feo and Resende [1995]). GRASP is an iterative process in which each iteration consists of two stages: construction and local search. The construction stage builds a feasible solution whose neighbourhood is subsequently investigated until a local minimum is found using the local search stage.

The work carried out for this thesis consists of an initial stage which attempts to satisfy the hard constraints, whilst stage two involves eliminating as many soft constraint violations as possible subject to the hard constraints remaining satisfied. This method is attractive because solely focussing on satisfying the hard constraints in stage one increases the likelihood of finding a feasible solution. Moreover, if feasibility is established in stage one, then it will be guaranteed after completion of stage two. However, in stage two, the connectivity of the feasible solutions within the search space is reduced, perhaps even to the point at which it may be disconnected.

In this chapter we will consider and compare two approaches to stage one of solving the problem. The first uses selection heuristics and neighbourhood operators to attempt to place all events in the timetable, but as a result, two hard constraints have to be relaxed and the requirement is to minimise the number of these constraint violations. The second approach uses a partial solution method which temporarily leaves events unplaced if no feasible period is available. In the latter method, no hard constraints are relaxed. We consider the issue of searching a highly constrained search space and the means of increasing its connectivity, in order to improve the solution quality.

The fundamental idea of timetabling is to assign ‘events’ to ‘timeslots’ and ‘rooms’ whilst obeying various constraints. A constraint that is usually common to all problems is, ‘event A must not be assigned to the same timeslot as event B’, which can be represented as a graph colouring problem (de Werra [1985]).

Figure 3.1: An example of a Graph Colouring assignment using three colours (timeslots).



In Figure 3.1 the nodes represent events (such as a course). An edge appears between two nodes when any student from the population attends both events. A 'clash' is when a student would be scheduled to attend more than one event at any time. In order to avoid clashes, events joined by edges must be coloured differently and the colours represent different timeslots. Thus an edge between two nodes corresponds to the events that cannot be placed in the same timeslot.

Real world problems usually involve additional constraints, such as room sizes and equipment availability. Some of these constraints can be included in the graph colouring technique however, we will investigate further methods of solving multi-constraint problems.

The objective for a successful timetable solution may vary from case to case. For example, one university may wish to minimise the number of timeslots (Bullnheimer [1998]), whereas the priority for another university may be to minimise the disruption to students (Cambazard et al. [2012]). The timetable produced needs to be practical both for the university and for the participants whilst using resources effectively and efficiently. Resources generally relate to the availability of suitable rooms and equipment.

Timetabling is a problem common to all universities; however, it is difficult to compare algorithms as the problem varies between each university. Typically constraints are individual to a university and each problem is unique.

3.1 International Timetabling Competition

This thesis utilises the data provided by the International Timetabling Competition 2007 (ITC 2007)[1] (McCollum et al. [2010]). Over the past few years generic problem models and instances have become publicly available, the ITC 2007 for example. This was introduced to give researchers real-world problems on which they can compare their methods and test emerging techniques. The ITC 2007 was built on the success of the first International Timetabling Competition which was conducted in 2002[2]. The aim of the competition was to narrow the gap that exists between research and practice. The organisers believe that by attracting a more multi-disciplined approach, more significant advances in research were made (McCollum et al. [2010]).

In the first International Timetabling Competition, finding a feasible solution was reasonably straightforward so submissions focussed on minimising the number of soft constraint violations (Cambazard et al. [2012]). Therefore, ITC 2007 introduced more hard constraints to move further in the direction of real-world timetabling and to generate new approaches to the task. This made the problem of finding a feasible solution more complex and algorithms had to take account of this accordingly.

The competition was split into three tracks. The first track, examination timetabling (Gogos et al. [2010]), can be thought of as post-enrolment, as a student that is enrolled on a course with an associated exam is also expected to sit the exam. The second track is post enrolment-based course timetabling (Cambazard et al. [2012]), where the number of students enrolled on a course is already known at the time of the timetable construction. The timetable needs to be created in such a way that every student can attend all the courses on which they are enrolled. Finally, the third track is curriculum-based course timetabling (Lü and Hao [2010a]), which forms a weekly schedule that complies with criteria set by the institution rather than enrolment data. Although these tracks overlap in many ways they have very different constraints.

[1] http://www.cs.qub.ac.uk/itc2007/
[2] http://www.idsia.ch/Files/ttcomp2002/

Our research focusses on track two: post enrolment-based course timetabling (Lewis et al. [2007]). Track two consists of 24 instances in total. There were eight 'early' instances released at the start of the competition, eight 'late' instances released two weeks before the deadline of the competition and eight 'hidden' instances that were not released to the competition entrants but were used to help the judges determine a winner. In each instance, there are 45 non-overlapping timeslots of uniform length available (five days with nine timeslots in each).

Each of the 24 datasets comprises; a set of rooms in which events can take place, a set of students who attend the events, and a set of features in certain rooms that are required by events. The sizes of these sets varies between each dataset as shown in Table 3.1. This table also shows the density of each problem instance. This is calculated by:

$$Problemdensity = \frac{n}{45 * m} \tag{3.1}$$

where n is the number of events, m is the number of rooms and 45 is the constant number of timeslots. This does not take into account how constrained the problem is in terms of the multiple other constraints, however it does give an indication of how much room there is for the events to be reassigned to different timeslots and rooms, thus, demonstrating how difficult it may be in some instances to find a suitable assignment for all events when the timetable is so dense.

A time limit was imposed on the algorithm run time; although a time limit may not be necessary in the 'real world', it was essential for a fair competition. A benchmarking program was created which competitors could execute on their own machines. When executed the program would perform a number of computational operations involved in timetabling. The time taken to execute this program would be used to calculate the time limit for this machine. We have not considered this time limit in the results presented in this thesis.

Track two incorporates five hard constraints:

Table 3.1: Statistics of each instance in the ITC 2007.

Instance	Number of events	Number of rooms	Number of features	Number of students	Density of problem
Early1	400	10	10	500	89%
Early2	400	10	10	500	89%
Early3	200	20	10	1000	22%
Early4	200	20	10	1000	22%
Early5	400	20	20	300	44%
Early6	400	20	20	300	44%
Early7	200	20	20	500	22%
Early8	200	20	20	500	22%
Late1	400	10	20	500	89%
Late2	400	10	20	500	89%
Late3	200	10	10	1000	44%
Late4	200	10	10	1000	44%
Late5	400	20	10	300	44%
Late6	400	20	10	300	44%
Late7	200	10	20	500	44%
Late8	200	10	20	500	44%
Hidden1	100	10	10	500	22%
Hidden2	200	10	10	500	44%
Hidden3	300	10	10	1000	67%
Hidden4	400	10	10	1000	89%
Hidden5	500	20	20	300	56%
Hidden6	600	20	30	1000	67%
Hidden7	400	20	30	1000	44%
Hidden8	400	20	30	1000	44%

HC1 - No student attends more than one event at the same time;

HC2 - The room is big enough for all the attending students and satisfies all the features required by the event;

HC3 - Only one event is put into each room in any timeslot;

HC4 - Events are only assigned to timeslots that are pre-defined as available for those events;

HC5 - Where specified, events are scheduled to occur in the correct order in the week.

HC1, HC2 and HC3 were included in the 2002 competition. HC4 and HC5 were added to these and included in the 2007 competition. There were also three soft constraints:

SC1 - a student has a class in the last slot of the day;

SC2 - a student has more than two classes consecutively;

SC3 - a student has a single class on a day.

A timetable is judged by its 'Distance to Feasibility' which is the sum of the number of students enrolled on the unplaced events. The soft constraints contribute towards the 'Soft Cost' which is the sum of all students affected by any of the soft constraints being violated. The quality of the timetable is determined by the soft cost. This problem can be more formally represented by the following mathematical model.

Parameters:

n the number of events

p the number of features

q the number of students

m the number of rooms

b_i the size of event i where $i = 1, ..., n$

E set of events $= \{e_1, ..., e_n\}$

S set of students $= \{s_1, ..., s_q\}$

R set of rooms $= \{r_1, ..., r_m\}$

T set of timeslots $= \{t_1, ..., t_{45}\}$

F set of room features $= \{f_1, ..., f_p\}$

D set of days $= \{d_1, ..., d_5\}$

G set of pairs of events where the precedence constraint applies $= \{(e_1, e_2), (e_1, e_4)...\}$

Variables:

$$x_{etr} = \begin{cases} 1 & \text{if event } e \text{ is scheduled in timeslot } t \text{ and room } r \\ 0 & \text{otherwise} \end{cases}$$

$$y_{s,t} = \sum_{t \in T} x_{et} \cdot z_{se} \qquad \forall e \in E,\ s \in S$$

i.e. $y_{s,t} = 1$ if student s attends an event in timeslot t, $y_{s,t} = 0$ otherwise.

$$w_{s,d} = \begin{cases} 1 & \text{if } \sum_{t=(d-1)\times 9+1}^{(d\times 9)} y_{s,t} = 1 \qquad \forall s \in S, d \in \{1, ..., 5\} \\ 0 & \text{otherwise} \end{cases}$$

i.e. $w_{s,d} = 1$ if the student s attends only one event on the day d and $w_{s,d} = 0$ otherwise.

Constants:

$$a_{et} = \begin{cases} 1 & \text{if event } e \text{ is pre-defined as available for timeslot } t \\ 0 & \text{otherwise} \end{cases}$$

$$c_{er} = \begin{cases} 1 & \text{if room } r \text{ is} \geq \text{the size of event } e \text{ and contains the features needed by event } e \\ 0 & \text{otherwise} \end{cases}$$

$$z_{se} = \begin{cases} 1 & \text{if student } s \text{ is required to attend event } e \\ 0 & \text{otherwise} \end{cases}$$

$v_{e_ie_j} =$ number of common students between event e_i and event e_j

The objective function is,

$$\text{minimise} \sum_{i=1}^{3} SC(i)$$

Where,

A student has a class in the last slot of the day

$$\text{SC(1)}, \quad \sum_{s=1}^{q} \sum_{t \in \{9,18,\ldots,45\}} y_{s,t}$$

A student has more than two classes consecutively

$$\text{SC(2)}, \quad \sum_{s=1}^{q} \sum_{d=1}^{5} \sum_{t=(d-1)\times 9+1}^{d\times 9-2} y_{s,t} \cdot y_{s,(t+1)} \cdot y_{s,(t+2)}$$

A student has a single class on a day

$$\text{SC(3)}, \quad \sum_{s=1}^{q} \sum_{d=1}^{5} w_{s,d}$$

Subject to,

No student attends more than one event at the same time,

$$\text{HC1}, \quad \sum_{r_1, r_2 \in R: r_1 \neq r_2} x_{e_i t r_1} \cdot x_{e_j t r_2} \cdot v_{e_i e_j} = 0 \qquad \forall\, t \in T,\ e_i, e_j \in E,\ r_1 \neq r_2$$

The room is big enough for all the attending students and satisfies all the features required for the event,

HC2, $$\sum_{r \in R} \sum_{t \in T} c_{er} \cdot x_{etr} = 1 \qquad \forall\, e \in E$$

Only one event is put into each room in any timeslot,

HC3, $$\sum_{e \in E} x_{etr} \leq 1 \qquad \forall\, t \in T,\, r \in R$$

Events are only assigned to timeslots that are pre-defined as available for those events,

HC4, $$\sum_{r \in R} \sum_{t \in T} x_{etr} \cdot a_{et} = 1 \qquad \forall\, e \in E$$

Where specified, events are scheduled to occur in the correct order in the week,

HC5, $$\sum_{r \in R} \sum_{t \in T} t \cdot x_{e_i tr} < \sum_{r \in R} \sum_{t \in T} t \cdot x_{e_j tr} \qquad \forall\, e_i, e_j \in G$$

An assignment of events to timeslots and rooms can be defined by a rectangular matrix $X_{m \times 45}$ where the rows represent the rooms and the columns represent the timeslots. If $X_{r,t} = e$ (where $0 \leq e \leq n-1$) then event e is assigned to room r and timeslot t. If $X_{r,t} = -1$ then the cell representing room r and timeslot t in the timetable matrix is not assigned an event. An example of an assignment is given in Figure 3.2.

Figure 3.2: An example of an assignment.

	t_1	t_2	...	...	t_{45}
r_1	-1	e_7			e_{n-4}
r_2	e_{n-2}	-1			e_{n-1}
...					
...					
r_m	e_2	-1			e_{10}

Before any assignments to the timetable matrix are made, the data provided for each instance is processed and stored in matrices. The size and features of each room can then be used to determine which rooms are suitable for which events and are stored in a binary matrix, $RE_{m \times n}$, where the rows represent the rooms and the columns represent the events. If $RE_{r,e} = 1$ then

event e can be placed in room r without exceeding the size and also the features needed for the event are satisfied. If $RE_{r,e} = 0$ then event e cannot be placed in room r.

A binary matrix, $SE_{q \times n}$, was constructed to represent the students that are required to attend each event. $SE_{s,e} = 1$ if the student s has to attend event e and $SE_{s,e} = 0$ otherwise.

The binary matrix, $ET_{n \times 45}$, was constructed to represent the timeslots that are predefined as available for each event. $ET_{e,t} = 1$ if event e can be placed in timeslot t according to the criteria set by HC4.

The data provided for HC5 is stored in a matrix, $EE_{n \times n}$, which shows any precedence between two events. For example if $EE_{e_i,e_j} = 1$ then event e_i needs to scheduled before event e_j. If $EE_{e_i,e_j} = -1$ then event e_j needs to be scheduled before event e_i. If $EE_{e_i,e_j} = 0$ then event e_i and event e_j share no precedence.

Finally, a matrix, $CL_{n \times n}$, represents the conflicts between two events i.e. a graph where the nodes represent the events and the edges between the nodes show that the events share at least one student. $CL_{n \times n}$ stores in each cell, the number of students shared by two events.

3.2 Initial construction and neighbourhood operators

The two key components of a local search method are the initial solution and the neighbourhood operators, these will both be described in this chapter. The neighbourhood operators explained have been used in stage one of the two-stage process.

3.2.1 Initial solution construction

In this section we compare techniques for producing an initial solution to the timetabling problem. The aim is to identify the best solution from the heuristics examined, and then we will go on to improve that solution in accordance with a set of constraints.

When constructing a solution, events should be scheduled beginning with the most difficult event to place. There are many different ways of determining the difficulty of placing an event, all of which relate to the feasibility of satisfying each constraint. The order in which the events are selected for placement into the timetable can be critical to the outcome achieved.

We tested five different methods of determining the order of selecting events using the following heuristics which were proposed by Carter et al. [1996].

- 'Largest Degree' ranks the events in descending order by the number of conflicts with all other events, and priority is given to events with the greatest number of conflicts.
- 'Largest Weighted Degree' is similar to 'Largest Degree', however it weights each conflict, by the amount of students involved in the conflict and places the event with the largest amount first.
- 'Colour Degree' places the events in order of which one has the largest amount of conflicts with the events that are already scheduled. This is a dynamic method as the number of conflicts gets recalculated after each iteration.
- 'Largest Enrolment' ranks events in descending order of the number of students enrolled on the event. Priority is given to the event with the highest number of students enrolled.
- 'Saturation Degree' ranks the events in ascending order by the number of feasible timeslots remaining in the timetable. Priority is given to the event with the least amount of available timeslots. The number of feasible timeslots remaining for each event needs to be recalculated after each iteration. Saturation degree can be compared with the DSATUR algorithm for graph colouring (Brélaz [1979]), where the next node to be coloured is chosen among those with the largest number of already differently coloured neighbours i.e. the nodes with the minimum saturation degree.

The selection heuristics Largest Degree, Largest Weighted Degree and Largest Enrolment are examples of static heuristic ordering which has the order of the events to be placed predeter-

mined before the process has begun. Colour Degree and Saturation Degree are examples of dynamic heuristic ordering which recalculates the next event to be placed after every insertion into the timetable.

For each of the selection heuristics, the chosen events are placed in the timeslot that is hardest to fill. That is, all timeslots and rooms are ranked in ascending order of the number of events that can feasibly be inserted into that period that have not already been placed. In the case of a tie the earliest period is chosen. This method attempts to fill more difficult timeslots earlier in the process, leaving easier timeslots available for events later on when the timetable is more constrained. We conducted some initial experiments in which we tested the above five heuristics. These showed that they were not successful in placing all events into the timetable for any of the 24 datasets, and were forced to leave many events unplaced to ensure a valid timetable. In order to place all the events, HC1 and HC5 were relaxed (see Section 3.1 for definition). These constraints were chosen to be relaxed because during initial experiments it was noted that as each run progressed, otherwise feasible timeslots were being rejected due to these constraints. To measure the effectiveness of these solutions, the 'distance to feasibility' was determined in each case. The distance to feasibility is the total number of students that are due to attend unscheduled events. Due to the relaxation of HC1 and HC5 we were able to schedule all the events in all datasets. The focus of this method is to schedule all events and minimise any violations of the relaxed constraints. However, to find the distance to feasibility we need to find which events are causing the constraint violations and therefore would not have been scheduled had we not relaxed the constraints. To do this, the events are listed in descending order of the total number of clashes and ordering violations that they are involved in. The event at the top of the list is removed from the timetable and the list is then recalculated. This process continues until there are no further constraint violations.

Using the above methods the average distance to feasibility, expressed in terms of the total number of students, is shown in Table 3.2. The distance to feasibility in terms of the number of events is shown in brackets.

Table 3.2: The average distance to feasibility of the initial timetables constructed using different heuristics. The value shown in brackets is the distance to feasibility in terms of the number of events that need to be left unplaced to produce a feasible timetable. The values shown in bold text are the lowest values of the distance to feasibility generated by the selection heuristics.

Instance	Largest enrolment	Largest degree	Largest weighted degree	Saturation degree	Colour degree
Early1	6133 (222)	6095 (219)	6320 (243)	**5748 (206)**	7397 (269)
Early2	6166 (227)	6116 (226)	6085 (223)	**5976 (219)**	7260 (263)
Early3	10101 (132)	9987 (133)	9899 (135)	**9104 (121)**	9377 (125)
Early4	10055 (139)	10365 (146)	10198 (150)	**9230 (134)**	9978 (143)
Early5	4118 (252)	4350 (264)	4406 (279)	**3887 (243)**	4739 (290)
Early6	4247 (251)	4299 (255)	4284 (259)	**3945 (230)**	4631 (270)
Early7	4687 (125)	4778 (127)	4734 (131)	**4179 (113)**	4697 (128)
Early8	4870 (136)	4898 (140)	4965 (144)	**4434 (126)**	4606 (131)
Late1	6103 (212)	**5849** (213)	6129 (214)	5995 **(211)**	7176 (249)
Late2	6195 (224)	**6063 (220)**	6150 (229)	6422 (234)	7476 (269)
Late3	9277 (123)	8746 (116)	8777 (122)	**7902 (108)**	9056 (123)
Late4	9213 (131)	8847 (125)	8977 (128)	**8605 (121)**	8756 (126)
Late5	4202 (251)	4322 (256)	4422 (268)	**4144 (242)**	4702 (282)
Late6	4303 (262)	4364 (266)	4500 (277)	**4223 (254)**	4702 (283)
Late7	4012 (114)	4179 (121)	4222 (123)	**3825 (110)**	3935 (112)
Late8	3827 (108)	3813 (110)	4202 (122)	**3533 (101)**	4053 (116)
Hidden1	7110 (63)	6944 (65)	6911 (66)	**6166 (54)**	6694 (65)
Hidden2	6967 (131)	7045 (130)	7136 (139)	**6820 (126)**	7153 (131)
Hidden3	8287 (174)	8257 (178)	8461 (187)	**8231 (170)**	8964 (188)
Hidden4	7146 (202)	6830 (194)	6993 (208)	**6740 (192)**	7972 (230)
Hidden5	3875 (295)	3846 (301)	3987 (306)	**3771 (289)**	4578 (349)
Hidden6	6804 (373)	6858 (376)	6709 (376)	**6512 (358)**	7646 (419)
Hidden7	15391 (264)	16271 (274)	15916 (278)	**15003 (255)**	16059 (277)
Hidden8	8314 (237)	8679 (253)	8701 (254)	**7813 (228)**	8609 (247)
Average	6725 (193)	6741 (196)	6795 (202)	**6342 (185)**	7092 (211)

Inspection of Table 3.2 shows that the lowest distance to feasibility was achieved by using saturation degree in all but two instances (Late1 and Late2). Therefore, Saturation Degree was chosen as the best option for producing an initial solution. This is also supported in the literature (Carter et al. [1996], Casey and Thompson [2003b], Burke et al. [2007]).

The results also show how difficult it will be to eliminate any hard constraint violations. In all but two of the results more than half of the events need to be removed for the timetable to be valid.

The following sections will investigate improving the initial solution using local search methods. Local search methods are used for solving optimisation problems. The search is called 'local' because it can only move from one solution to a neighbouring solution. Solutions are neighbours if there is a relation defined by the search space of the neighbourhood operator being used. Typically, a solution will have more than one neighbouring solution.

3.2.2 Optimisation strategy

In order to achieve an acceptable initial solution it was necessary to relax constraints HC1 and HC5. Our focus is now to minimise the number of violations arising from the relaxation of these constraints. This section examines the search operators which can be used to try to minimise the HC1 and HC5 violations that occur. During the optimisation process, HC2, HC3 and HC4 violations are forbidden.

To compare a solution to its neighbour it is necessary to define a cost function. We used two different cost functions for comparison.

Cost1 is the total number of **students** that are required to attend more than one event at any time, plus the total number of **students** that are required to attend the events that violate the precedence constraint.

Cost2 is the total number of the **events** that contain common students that are scheduled at the

same time, plus the total number of **events** that violate the precedence constraint. Table 3.3 shows the value of cost1 and cost2 for each instance using the initial solution (produced using Saturation Degree).

Table 3.3: The average value of the two cost functions for the initial timetables in each instance.

Instance	Cost1	Cost2
Early1	2839	564
Early2	2676	629
Early3	6723	667
Early4	6476	725
Early5	3219	947
Early6	3170	947
Early7	2348	550
Early8	2666	530
Late1	3030	597
Late2	2877	685
Late3	3669	344
Late4	3713	454
Late5	3322	998
Late6	3473	1101
Late7	1845	359
Late8	1627	254
Hidden1	5669	194
Hidden2	4051	460
Hidden3	3154	586
Hidden4	2939	510
Hidden5	2852	1020
Hidden6	4572	1325
Hidden7	10259	1258
Hidden8	6622	922
Average	3908	693

3.2.2.1 The move operator

In an attempt to reduce the cost incurred, a simple neighbourhood operator was employed. This attempts to move a randomly chosen event, from those currently causing a violation to an alternative random timeslot, if this can be achieved without violating HC2, HC3 or HC4. We will call this a 'move operator'. We developed the code from scratch for a maximum matching

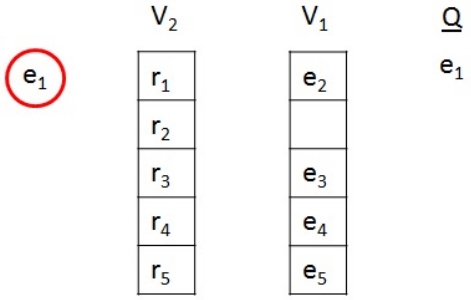
algorithm for bipartite graphs to be used once an event has been moved to a new timeslot to maintain feasibility of HC2, HC3 and HC4, and maximise the possibility of a successful move. There are two groups of vertices/nodes: V_1 are the events and V_2 are the rooms forming a bipartite graph. The connecting lines between the groups of nodes represent that the event can feasibly be placed in that room. Connecting lines cannot be formed between two nodes in the same group. Therefore, if the vertices encountered on the i^{th} step of the search, L_i, is an event, then L_{i+1} must be a room adjacent to the event in L_i which did not appear in previous steps of the search. Similarly if L_i is a room, then L_{i+1} is an adjacent event.

Figure 3.3 shows an example of a successful insertion of an event to a timeslot using the maximum matching algorithm. The example shows the two groups V_1, representing the events, and V_2, representing the rooms.

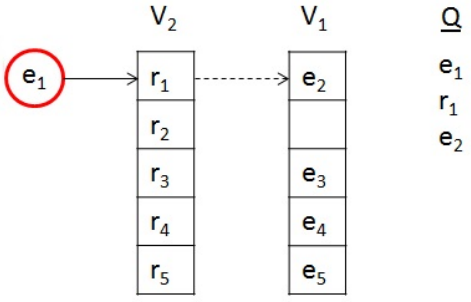
The maximum matching algorithm uses a 'First in First Out' queue data structure. An event is chosen and all adjacent nodes are added to the queue. The vertex at the front of the queue is deleted, and all nodes connected to it that have not been discovered in previous steps are added to the end of the queue. This process is repeated until the queue is empty, and the nodes predecessor are recorded at each step. Nodes are added to the queue in an arbitrary order. However, a heuristic could be added to control the order in which they are added. Once the search is complete, the results of the search can be traced and the events can be placed in the appropriate rooms. If all events cannot be placed then the move is rejected and the next timeslot is considered. A descent technique is used where a comparison is made between the old cost and the new cost, and provided that the cost is reduced and that HC2, HC3 and HC4 are not violated, the move is accepted. A move that does not change the cost (a sideways move) is always accepted as this can lead to other areas in the search space that have not yet been explored.

Two programs have been constructed; one program uses cost1 and the other uses cost2. On each iteration, the move operator is performed followed by the maximum matching algorithm. The process ended when 1,000 iterations had passed or until the cost function equalled zero.

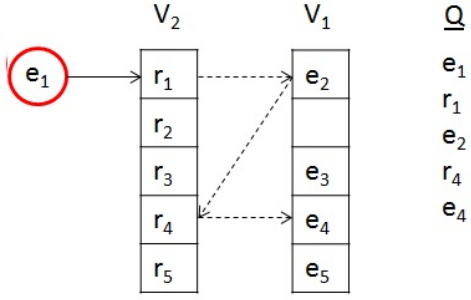
Figure 3.3: An example of how the Maximum Matching Algorithm is applied to insert an event into a timeslot



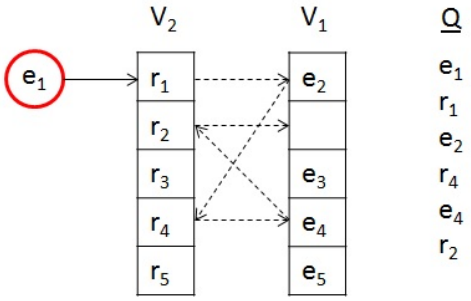
(a) An event, e_1, needs to be placed into the timeslot so it is added to the queue (Q)



(b) e_1 can be inserted into r_1 so r_1 is added to the Q. We discover e_2 is occupying r_1 therefore e_2 is also added to the Q



(c) e_2 can be moved to r_4 which is occupied by e_4 so these are also added to the Q



(d) e_4 can be scheduled in r_2 which is added to the Q. This room is currently empty so the process is complete. We can then follow the Q back to move the events to their amended rooms and successfully insert e_1 into the timeslot.

The decision to end the run after 1,000 iterations was made by observing when the cost function was reaching a steady state. This method showed some reduction in the cost functions. Clearly the more times the method is applied (number of runs undertaken), the more rigorous the results will be.

Further work was undertaken using a steepest descent method which explores all the neighbour solutions before selecting the one that makes the best improvement to the cost function without violating HC2, HC3 or HC4. Moves that incur an equal cost are accepted so that we can explore other areas of the search space. This was repeated until the stopping criteria was reached, either; 1,000 iterations were completed, the cost function had not improved in the preceding

100 iterations or the cost function had reached zero. The values of the cost functions using the steepest descent method are contained in Table 3.4.

Table 3.4: The average value of the cost functions of the timetables and the percentage of improvement from the initial costs once the move operator has been performed using a steepest descent method. The distance to feasibility is shown in brackets.

Instance	Cost1 (DTF)	% improvement	Cost2 (DTF)	% improvement
Early1	1470 (3419)	48%	265 (3375)	53%
Early2	1666 (4212)	38%	374 (4117)	41%
Early3	905 (1538)	87%	29 (1518)	96%
Early4	717 (1264)	89%	26 (1258)	96%
Early5	934 (803)	71%	74 (798)	92%
Early6	1037 (1007)	67%	91 (1000)	90%
Early7	429 (863)	82%	45 (846)	92%
Early8	385 (537)	86%	20 (537)	96%
Late1	1723 (3802)	43%	301 (3777)	50%
Late2	1946 (4715)	32%	423 (4610)	38%
Late3	925 (2034)	75%	40 (2023)	88%
Late4	829 (2026)	78%	44 (2015)	90%
Late5	978 (1035)	71%	101 (1040)	90%
Late6	985 (1032)	72%	101 (1024)	91%
Late7	389 (518)	79%	21 (525)	94%
Late8	361 (328)	78%	11 (325)	96%
Hidden1	624 (623)	89%	5 (628)	97%
Hidden2	929 (1995)	77%	72 (1998)	84%
Hidden3	1353 (3561)	57%	169 (3548)	71%
Hidden4	735 (898)	75%	30 (899)	94%
Hidden5	907 (755)	68%	91 (761)	91%
Hidden6	2623 (3722)	43%	541 (3677)	59%
Hidden7	3580 (7166)	65%	388 (7006)	69%
Hidden8	3797 (2939)	43%	155 (2933)	83%
Average	1260 (2116)	67%	142 (2093)	81%

Table 3.4 shows us that using a simple move operator and a steepest descent method can make significant improvements. On average, cost1 decreased by 67% and cost2 by 80% from the initial positions. It was felt that the run times could be improved upon as the computational time was approximately 70 seconds.

Tabu search (Glover [1987]) is an extension of the steepest descent method, and as such, should

achieve better results. Tabu Search accepts new inferior solutions when no more moves can be found to decrease the cost function value. This is used to ensure the search can continue after finding a local minimum. The search continues once a local minimum has been reached, whereas the steepest descent method would have stopped because there are no remaining moves that can be performed to improve the cost function. Tabu search will explore new regions of the search space and may achieve a lower cost than the local minimum obtained before. When a move is performed the reverse of the move is added to a list and is then considered "tabu" for the next T iterations, where T is the tabu list length. The reverse move will remain on the list for T iterations and whilst it remains on the list, it is forbidden to perform this move in order to avoid paths already investigated in the search space. Aspiration criteria are employed which allows the search to accept a tabu move if the move results in a better solution than the best solution found so far, this is usually measured by the cost function. An appropriate tabu list length will vary between problems, so we tested many different tabu list lengths such as; different constants and a variable length. When a tabu list length is too small, cycling can occur. If it is too large, the solution quality will deteriorate because too many moves would be refused. The variable tabu list length that was tested depends on the number of event clashes.

$$T = Random\{0, 9\} + \alpha * Event\ Clashes \tag{3.2}$$

where α is a variable (Blöchliger and Zufferey [2008]) which literature suggests 0.6 to be a popular value for graph colouring (Galinier and Hao [1999]). The search continues until either; 1,000,000 iterations have passed, the cost function has not improved in 1,000 iterations or the cost function reaches zero. The maximum number of iterations has significantly increased from the steepest descent method but this is because it needs longer to explore the solution space given that the cost function can increase as well as decrease.

There are many metaheuristics that could be used in this situation for example, Simulated Annealing (Kostuch [2005]), Genetic Algorithms (Burke et al. [1994]) or Ant Colony Optimisation (Socha et al. [2002]). Tabu Search was chosen because it requires less parameters to be

tuned which facilitates a quicker result. Table 3.5 shows the results of the move operator using a tabu search. As anticipated, these results show significant improvement over those achieved using previous methods. On average, cost1 reduced by a further 4% and cost2 by 5% than the steepest descent method.

Table 3.5: The value of the cost functions and the percentage of improvement made from the initial cost once the move operator has been performed using a tabu search. The distance to feasibility is shown in brackets.

Instance	Cost1 (DTF)	% improvement	Cost2 (DTF)	% improvement
Early1	1193 (2886)	58%	208 (2895)	63%
Early2	729 (2172)	73%	129 (2061)	79%
Early3	905 (1538)	87%	29 (1519)	96%
Early4	717 (1264)	89%	26 (1259)	96%
Early5	934 (803)	71%	74 (798)	92%
Early6	1037 (1008)	67%	91 (1001)	90%
Early7	429 (863)	82%	45 (847)	92%
Early8	385 (538)	86%	20 (537)	96%
Late1	1378 (3253)	55%	230 (3179)	61%
Late2	895 (2464)	69%	148 (2400)	78%
Late3	925 (2034)	75%	40 (2024)	88%
Late4	829 (2026)	78%	44 (2015)	90%
Late5	978 (1035)	71%	101 (1040)	90%
Late6	985 (1032)	72%	101 (1025)	91%
Late7	389 (519)	79%	21 (525)	94%
Late8	361 (329)	78%	11 (325)	96%
Hidden1	624 (623)	89%	6 (629)	97%
Hidden2	801 (1716)	80%	52 (1621)	89%
Hidden3	1353 (3561)	57%	170 (3549)	71%
Hidden4	735 (899)	75%	30 (899)	94%
Hidden5	907 (756)	68%	91 (761)	91%
Hidden6	2265 (3077)	50%	408 (3103)	69%
Hidden7	3580 (7166)	65%	388 (7007)	69%
Hidden8	3797 (2940)	43%	155 (2934)	83%
Average	1131 (1854)	71%	109 (1831)	86%

3.2.2.2 The swap operator

The swap operator was then introduced and used alongside the move operator to further improve the cost functions. The swap operator simply tries to swap the timeslots of two events scheduled in the timetable whilst maintaining feasibility of HC2, HC3 and HC4. This was incorporated into the tabu search described in Section 3.2.2.1. A move is a specific type of swap between an event and an empty timeslot and room. The search compared the change in the cost function of all possible moves and swaps. The operation, either a move or a swap, that incurred the biggest reduction in the cost function would be performed. The maximum matching algorithm was used to place the events in the new timeslots as before. Once it reached a local minimum, it again would perform worsening moves or swaps to search alternative areas of the search space. This process is continued either; for 1,000,000 iterations, until the solution has not been improved for 1,000 iterations or if the cost function equals zero.

Results are shown in Table 3.6 of the two cost functions. On average, cost1 decreased by a further 6% from moves with tabu, and cost2 decreased by a further 10%. Cost2 has decreased by 96% compared with the initial cost function. It is unclear why cost2 should perform better than cost1 but one hypothesis is that cost2s cost landscape is likely to be less spiky than cost1s, making the tabu search algorithm less susceptible to getting stuck in local optima.

3.2.2.3 Timeslot swap operator

At this point, having employed moves and swaps, HC2, HC3 and HC4 have no violations by definition, and HC1 is only violated in two of the 24 instances. It is therefore appropriate to focus on HC5 which relates to the ordering of events. To facilitate this, we employed the timeslot swap operator, which preserves the allocated room, after the move and swap operators to rearrange the order of events in the timetable.

Again, using a tabu search, all pairs of timeslots were considered for a swap, and the swap that made the best improvements to the number of HC5 violations were performed. Worsening

Table 3.6: The value of the cost functions and the percentage improvement from the initial costs once the move and swap operators have been performed using a tabu search. The distance to feasibility is shown in brackets.

Instance	Cost1 (DTF)	% improvement	Cost2 (DTF)	% improvement
Early1	529 (814)	81%	21 (515)	96%
Early2	499 (902)	81%	21 (515)	97%
Early3	678 (921)	90%	10 (543)	99%
Early4	646 (909)	90%	11 (652)	98%
Early5	932 (718)	71%	63 (717)	93%
Early6	902 (749)	72%	63 (763)	93%
Early7	286 (545)	88%	9 (230)	98%
Early8	328 (378)	88%	10 (313)	98%
Late1	605 (1210)	80%	21 (536)	96%
Late2	575 (1251)	80%	27 (633)	96%
Late3	669 (1140)	82%	10 (612)	97%
Late4	753 (1392)	80%	11 (671)	98%
Late5	861 (738)	74%	55 (680)	94%
Late6	862 (720)	75%	61 (695)	94%
Late7	301 (363)	84%	9 (252)	97%
Late8	361 (420)	78%	10 (299)	96%
Hidden1	566 (739)	90%	5 (550)	97%
Hidden2	531 (853)	87%	10 (470)	98%
Hidden3	606 (1614)	81%	22 (902)	96%
Hidden4	718 (824)	76%	19 (605)	96%
Hidden5	870 (690)	69%	71 (628)	93%
Hidden6	1481 (1576)	68%	92 (1178)	93%
Hidden7	1093 (2599)	89%	20 (969)	98%
Hidden8	3287 (2345)	50%	101 (2295)	89%
Average	793 (1017)	79%	31 (676)	96%

swaps may be performed once the search has reached a local minimum. The events contained in the timeslots can be swapped directly, as the rooms they are already placed in will not violate HC2 since that is forbidden in the solution construction, therefore the maximum matching algorithm does not need to be used. These timeslot moves do not affect HC1 since all events in the timeslot are swapped, so no extra student clashes will occur.

The timeslot swap made little or no improvement in most cases. This is due to the constraint HC4 which states all events have predefined timeslots into which they can be placed. Up to

20 events are being moved in the timeslot swap, so it is highly likely that one of them would violate HC4 if the swap were to take place, therefore, the swap would be disregarded and the next swap would be attempted, and so on. Consequently, the timeslot swap will not be used in further investigations.

3.2.3 Further optimisation using more complex operators

In this section we will consider further move operators which typically involve making much larger alterations to a timetable.

3.2.3.1 The Hungarian method

A further, more disruptive neighbourhood is now considered. Assignment problems can be formed by choosing a set of k events from different timeslots that share no precedence violations. The neighbourhood involves determining whether these events can be re-arranged in a way that maintains feasibility and reduces the SCP.

The Hungarian method is often used for assignment problems (Mills-Tettey et al. [2007], Bertsekas and Castañon [1993], Vizuete Luciano et al. [2012]) and was used in the winning algorithm of ITC 2007, Cambazard et al. [2012]. In the case of our timetabling problem, the Hungarian method (Kuhn [1955]) chooses a set of n events from different timeslots that share no precedence constraints. A non-negative matrix is formed where the element in the i^{th} row and j^{th} column represents the cost, if the i^{th} event was to be placed in the timeslot that the j^{th} event was removed from. The matrix is square so each event can only be assigned to one timeslot. The Hungarian method will iterate several times, until an optimal solution is obtained where all events are assigned into timeslots and required rooms minimising the hard constraint violations.

The cost of the events in each timeslot is known, they do not affect each other since they share no precedence. This method returns the optimal assignment of the randomly chosen k events

into the timeslots. Different values of k were tested, $k = 3$, $k = 4$ and $k = 5$. These small values of k were chosen due to the computational time that this method can take; since k is small this method is sufficiently efficient for our needs.

For example assume $k = 5$ and the matrix demonstrated in Figure 3.7 is formed.

Table 3.7: Example of an assignment problem with five events.

Event	Current timeslot	4	9	32	24	11
12	4	**11**	X	9	20	5
45	9	8	**7**	15	30	7
73	32	3	9	**0**	12	23
123	24	8	17	X	**4**	16
320	11	24	8	13	9	**22**

The current cost incurred by these five events is 44. Note that it is not feasible for event 12 to be allocated to timeslot nine, or for event 123 to be allocated to timeslot 32; these allocations are marked by an X. This neighbourhood allocates each event to a distinct timeslot such that the total cost is minimised. These assignment problems can be solved efficiently in a number of ways. We use the Hungarian Algorithm introduced by Kuhn [1955]. This method works as follows:

1. Set up an k x k cost matrix.
2. Subtract from each row the minimum value in that row.
3. Subtract from each column the minimum value in that column.
4. To do this, select a row (or column) containing at least one zero and draw a line through or cover the corresponding column (or row). The aim is to use the minimum number of lines to cover all the zeros in the matrix.
5. Suppose you use l lines. If $l = k$ then stop as the solution is optimal. If $l \neq k$ then let $v =$ the minimum value of the uncovered elements, subtract v from all uncovered elements and add v to any elements covered by two lines.

6. Return to step 4.

The optimal assignment of the matrix in Figure 3.7 is given in Figure 3.8.

Table 3.8: The optimal assignment of an assignment problem with five events.

Event	Current timeslot	4	9	32	24	11
12	4	11	X	9	20	**5**
45	9	**8**	7	15	30	7
73	32	3	9	**0**	12	23
123	24	8	17	X	**4**	16
320	11	24	**8**	13	9	22

The cost is reduced from 44 to 25. Events 12, 45 and 320 are re-allocated whereas 73 and 123 stay in the same timeslot.

Due to the computational time that this method takes, a steepest descent method would not be appropriate and was not used.

Since this is a highly constrained problem and events cannot easily be rescheduled, the events are regularly returned to their original positions and no improvement is made to the cost function. This was subsequently taken into account when choosing events for each iteration. The first event is chosen randomly, the following three events are then chosen so that they can be swapped with the first event and each other without violating HC4. However, this did not greatly improve the success of the Hungarian method, and it made little improvement with a high computational time.

3.2.3.2 Kempe chains

Kempe [1879] published a proof of the Four Colour Conjecture in 1879. This stated that a map in a plane can be coloured by four colours, so that no adjacent regions have the same colour. Eleven years later Heawood [1890] found the proof contained errors that could not be corrected. However, Kempe chains as they were known thereafter, could be used to prove that

it is possible for every map to be coloured using five colours. They can also be used to test for reducibility of a map (Birkhoff [1913]).

A Kempe chain neighbourhood represents a sub-graph of a graph. The sub-graph must be connected and contain exactly two colours; connected components of the sub-graph are then swapped producing an alternative feasible graph using the same number of colours as the original.

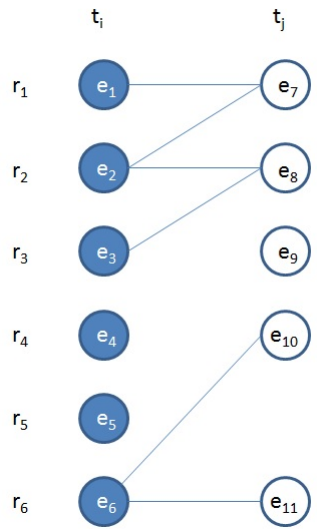
The Kempe chain neighbourhood involves swapping a subset of events in two distinct timeslots. Two timeslots are chosen at random, and a room within either timeslot is chosen randomly only from those that contain an event. The other timeslot is then considered, and any events that have students in common with this initial event are added to the chain. Further iterations of this process are undertaken until all events that have students in common are part of the chain. The events in the chain are then swapped to their respective alternative timeslot, with the room allocation for each event being made by the maximum matching algorithm described in Section 3.2.2.1. If a Kempe chain move is found to break HC4 it is automatically rejected.

Let n_i be the number of events to be swapped in the first timeslot, and n_j be the number of events to be swapped in the second timeslot. If $n_i > n_j$, there must be at least $n_i - n_j$ empty rooms in the second timeslot for a feasible solution to be possible and vice versa.

An example of a Kempe chain swap is illustrated in Figure 3.4. Edges exist between the vertices (representing events) that have students in common and therefore, would cause a clash if scheduled in the same timeslot. The figure shows two Kempe chains. One contains events $\{e_1, e_2, e_3, e_7, e_8\}$ and the other contains $\{e_6, e_{10}, e_{11}\}$. Note that this neighbourhood also contains moves (e.g. event e_5 can move to timeslot t_j, event e_4 may also be able to move to timeslot t_j depending on room suitability). The neighbourhood will also contain feasible swaps between clashing pairs of events.

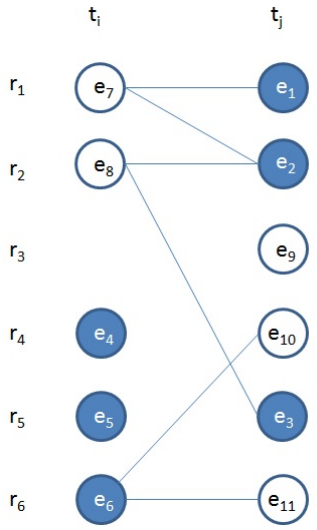
If we consider Kempe chain $\{e_1, e_2, e_3, e_7, e_8\}$ then if the move is accepted, the solution shown in Figure 3.5 may be the result. Note that the room allocation is determined via a matching

Figure 3.4: An example of two timeslots before a Kempe chain move has been performed.



algorithm.

Figure 3.5: An example of two timeslots once a Kempe chain move has been performed.

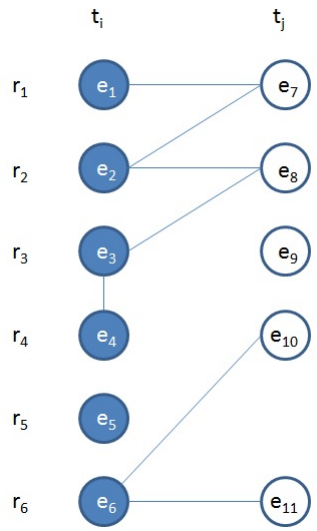


Each Kempe chain move may not preserve the satisfaction of the remaining hard constraints. For example, the Kempe chain that was not used in this example $\{e_6, e_{10}, e_{11}\}$ cannot lead to a feasible solution, because if event $\{e_6\}$ is exchanged with events $\{e_{10}, e_{11}\}$, there will be seven events assigned to timeslot t_i.

It is also possible that the Kempe chain neighbourhood can be used to reduce the number of

HC1 violations. Consider the Figure 3.6.

Figure 3.6: An example of the Kempe chain move with an infeasible solution.



The way our Kempe chains are generated means the chain will still consist of $\{e_1, e_2, e_3, e_7, e_8\}$. The blue events, e_1, e_2 and e_3, from t_i will be moved to t_j provided that feasible rooms can be assigned and the white events, e_7 and e_8, from t_j will be moved to t_i. The clash occurred between events, e_3 and e_4, since e_3 was part of the Kempe chain and was moved to the other timeslot and e_4 was not included in the chain the events are now scheduled in different timeslots and the clash no longer exists.

Tabu search was again employed for this neighbourhood, and the results are shown in Table 3.9. At this stage in the process, and with fewer violations to try and eliminate, it is even more difficult to move events around and try to further reduce the number of violations. It is therefore promising that the Kempe chains have made a small improvement to the move and swap neighbourhoods.

Figure 3.7 is a boxplot that represents the minimum, median and maximum improvement made to the cost function by Kempe chains. Cost2 was used as this has made better improvements on average than cost1. The maximum improvement to the cost function in three instances was 100% and the lowest was 71%. Many of the instances were reduced, on average, by over 90%

Table 3.9: The value of the cost functions and the percentage improvement from the initial costs once the move and swap operators have been performed followed by Kempe chains. The distance to feasibility is shown in brackets.

Instance	Cost1 (DTF)	% improvement	Cost2 (DTF)	% improvement
Early1	446 (686)	84%	18 (366)	97%
Early2	467 (844)	83%	21 (490)	97%
Early3	588 (798)	91%	10 (207)	99%
Early4	537 (755)	92%	8 (277)	99%
Early5	879 (677)	73%	59 (577)	94%
Early6	890 (739)	72%	60 (640)	94%
Early7	193 (367)	92%	7 (80)	99%
Early8	240 (276)	91%	7 (122)	99%
Late1	541 (1081)	82%	21 (490)	96%
Late2	536 (1166)	81%	27 (547)	96%
Late3	449 (766)	88%	8 (150)	98%
Late4	568 (1049)	85%	8 (158)	98%
Late5	810 (694)	76%	55 (580)	94%
Late6	827 (690)	76%	57 (615)	95%
Late7	193 (232)	90%	7 (108)	98%
Late8	288 (334)	82%	9 (129)	96%
Hidden1	451 (589)	92%	5 (241)	97%
Hidden2	452 (726)	89%	10 (134)	98%
Hidden3	595 (1585)	81%	21 (655)	96%
Hidden4	571 (655)	81%	19 (384)	96%
Hidden5	824 (653)	71%	71 (537)	93%
Hidden6	1444 (1537)	68%	92 (1152)	93%
Hidden7	928 (2207)	91%	18 (459)	99%
Hidden8	3268 (2331)	51%	101 (2055)	89%
Average	708 (893)	82%	30 (465)	96%

demonstrating how successful Kempe chains can be.

The order in which the operators are run can have a big impact on the results (Hansen and Mladenović [1999]). Tests were completed to determine what order the operators should be called upon. From the results it was shown that the ordering had little impact on the cost function but did affect the computational time. It was decided that the operators were best run in ascending order of size. Smaller operators are best used earlier in the process, as they take less computational time to make reasonable progress in reducing the cost function, followed by

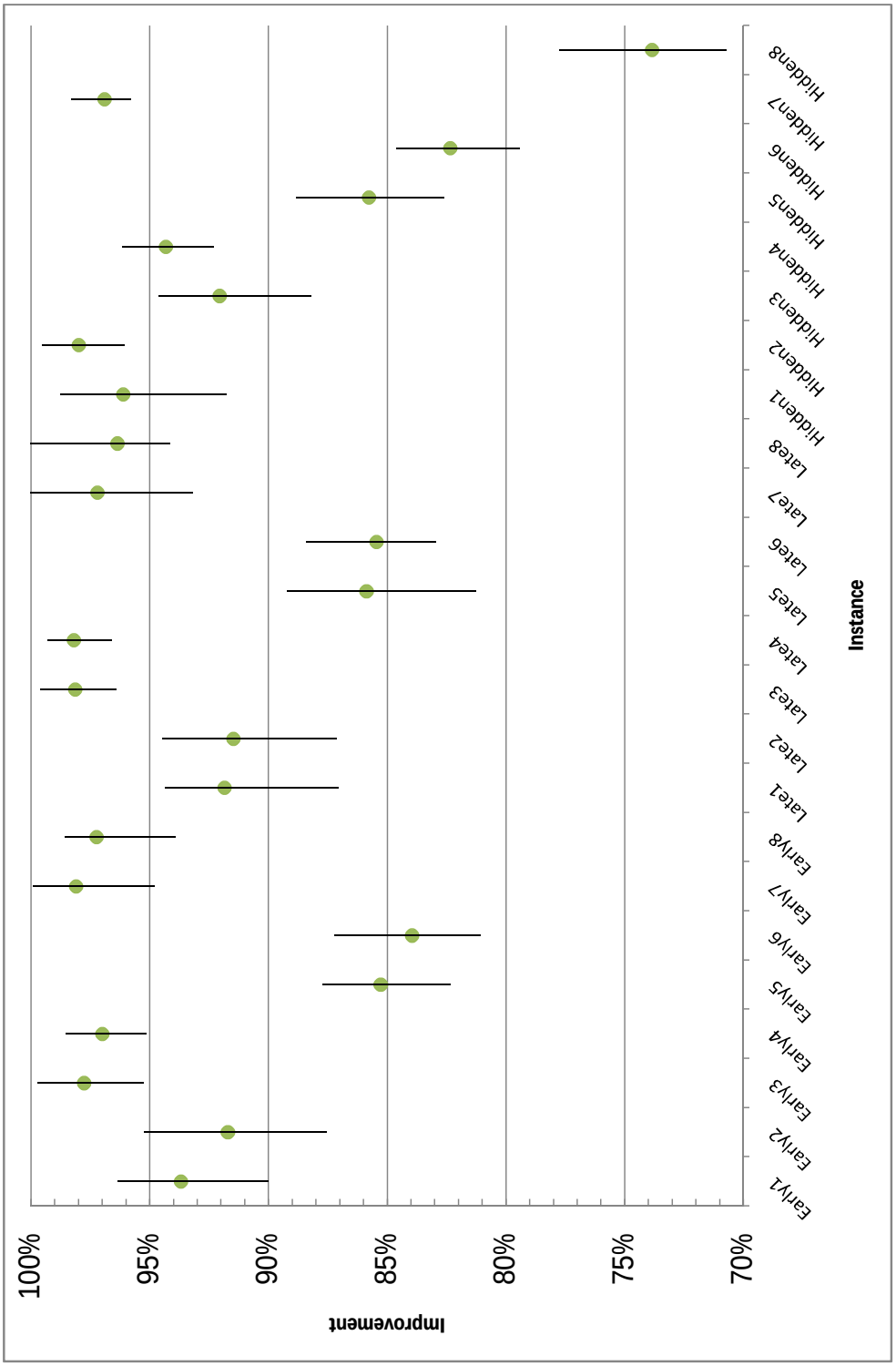
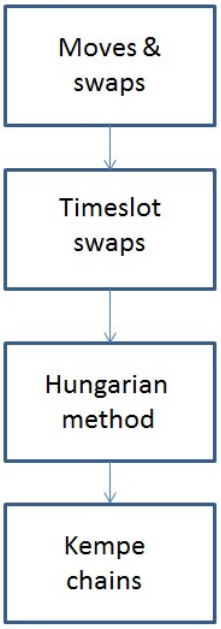


Figure 3.7: A boxplot to show the minimum, maximum and average improvement made to the cost by moves, swaps and Kempe chains in each instance.

the larger operators that will perform less iterations before reaching a local minimum thus taking less time. Refer to Figure 3.8 for a flow chart of the the final ordering of the neighbourhood operators presented in this section.

Figure 3.8: A flow chart outlining the process of the neighbourhood operators.



3.3 Partial solution method

The method described in Section 3.2 schedules all events in the timetable which can potentially 'block' positions, thus making it more difficult to find periods that can satisfy the hard constraints. The rationale for using the partial solution method is that it initially leaves events out to reduce the risk of blocking. For this approach HC1 and HC5 will not be relaxed, and events will be left unplaced if there is no feasible position to insert them. The five selection criteria; Largest Enrolment, Largest Degree, Largest Weighted Degree, Saturation Degree and Colour Degree described in Section 3.2.1 are tested. However, the difference is that no hard constraints are violated, and events are left unplaced if necessary. Results from testing the five selection criteria methods to produce an initial solution are presented in Table 3.10. The numbers shown is the final distance to feasibility as defined in Section 3.1 and the numbers shown in brackets are the number of events left unscheduled.

Colour degree has been shown to provide the best starting solution as it had the lowest DTF in the highest number of instances (nine of 24 instances highlighted in bold in Table 3.10). Colour degree selects the order of the events to be placed by prioritising the events that have the largest number of conflicts with the events that have already been scheduled. This is recalculated at each stage. The next event to be scheduled is placed into the timeslot with the lowest number of events that can be feasibly scheduled in it. If no feasible timeslots are available for the next event to be scheduled, then the event is added to an 'unplaced event' list to be input into the timetable later in the process. This is where this method differs from the method described in the previous section, where all events were inserted in the timetable, and to achieve this HC1 and HC5 were relaxed. In this method, no constraints are relaxed and the timetable remains valid at all times, and as a consequence some events may need to be left unplaced. As before, saturation degree was also shown to be a useful selection criterion. Saturation degree ranks events in ascending order of the number of feasible timeslots that remain in the timetable for each event, that is, the number of timeslots that are feasible and empty. The event with

Table 3.10: The average distance to feasibility (DTF) of the initial timetables constructed using five different heuristics. (The number of events left unscheduled).

Instance	Largest Enrolment	Largest Degree	Largest Weighted Degree	Saturation Degree	Colour Degree
Early1	**1369** (66)	1452 (59)	1711 (91)	1678 (62)	1377 **(50)**
Early2	1988 (89)	1981 (76)	1959 (90)	2017 (74)	**1710 (70)**
Early3	**349** (7)	702 (11)	547 (17)	779 (9)	354 **(6)**
Early4	976 (18)	1045 (15)	1357 (31)	1225 (18)	**911 (14)**
Early5	**625** (49)	647 (43)	957 (83)	696 **(42)**	672 (43)
Early6	**542** (44)	650 (43)	825 (64)	843 (49)	545 **(35)**
Early7	707 (24)	560 (18)	857 (30)	698 (20)	**501 (14)**
Early8	954 (30)	742 (21)	916 (31)	**427 (12)**	725 (21)
Late1	1820 (83)	1881 (71)	**1754** (71)	1813 **(64)**	1867 (69)
Late2	2192 (96)	2463 (100)	2320 (106)	2222 (78)	**1968 (77)**
Late3	1174 (21)	1110 (16)	1404 (31)	1378 (19)	**436 (7)**
Late4	1771 (29)	**1067 (16)**	1843 (32)	1609 (23)	1136 **(16)**
Late5	726 (56)	**620 (43)**	683 (57)	1027 (61)	745 (48)
Late6	**609 (49)**	856 (57)	680 (57)	883 (53)	869 (60)
Late7	847 (27)	839 (26)	798 (30)	**694 (19)**	815 (25)
Late8	474 (17)	202 (7)	525 (20)	183 **(5)**	**169 (5)**
Hidden1	**0 (0)**	177 (1)	273 (8)	**0 (0)**	270 (3)
Hidden2	1337 (37)	1278 **(25)**	**1208** (40)	1452 (28)	1573 (31)
Hidden3	2236 (55)	2048 (47)	2686 (71)	2129 (45)	**1805 (39)**
Hidden4	686 (26)	366 (12)	931 (42)	**337 (10)**	368 (11)
Hidden5	548 (58)	481 (42)	592 (62)	576 (44)	**353 (31)**
Hidden6	1854 (127)	1990 (123)	1928 (130)	2257 (120)	**1632 (95)**
Hidden7	4402 (87)	4134 (75)	4598 (96)	**4074 (67)**	4164 (74)
Hidden8	1406 (51)	1370 (46)	2044 (68)	**872 (28)**	1494 (46)
Average	1233 (48)	1194 (41)	1392 (57)	1245 (40)	1102 (37)

the least amount of available feasible timeslots is placed in the timetable first. Table 3.10 shows that saturation degree, as well as largest enrolment, found a feasible solution to the 'Hidden1' instance. Therefore, saturation degree will also be used for the following method as a comparison.

Once the initial timetable has been produced and we have a list of unplaced events the main objective is to then find a period for all the unplaced events whilst retaining a feasible timetable. A combination of tabu search and the maximum matching algorithm is used.

The cost function for this process is the number of students required to attend the events on the unplaced list. We assess the cost of each event on the unplaced list, in every timeslot to find the best feasible timeslot to insert them. The maximum matching algorithm, as explained in Section 3.2.2.1, is used to rearrange the events in the timeslot to accommodate the unplaced event. The timetable must remain feasible at all times, so any timeslots that would violate HC4 would not be considered. Any events that contain common students, or have a precedence with the event to be placed, would need to be removed from the timetable and added to the end of the unscheduled list of events. A record is kept of the students required to attend the events that would need to be removed if the unscheduled event were to be placed in that timeslot. The timeslots are ranked in ascending order of the number of students attending the events that would need to be replaced. Priority is given to the timeslot that when the event is inserted, the events that need to be removed result in the least number of students having events unscheduled. Any events that need to be removed from the timetable to keep it free of hard constraint violations are then transferred to the end of the unplaced event list. This process is repeated for all the events in the unplaced event list until either; they are all placed or it reaches a maximum number of iterations.

Similar to the previous method, tabu search (Glover [1987]) was used to avoid being trapped in local minima. When an unplaced event is being inserted into a timeslot, and consequently another event needs to be removed to retain a feasible timetable, placing the removed event back into the same timeslot would be tabu for T iterations, where T is the tabu list length. This

method was tested with many constant tabu list lengths, and also a variation of the variable tabu list length as described in the previous chapter. The tabu list length has been adapted here to depend on the number of unplaced events, as clashes do not exist in this version.

$$T = \text{Random}\{0, 9\} + \alpha * \text{number of unplaced events} \tag{3.3}$$

(Blöchliger and Zufferey [2008]) where $\alpha = 0.6$ (Galinier and Hao [1999]).

Table 3.11 shows the results with each instance using colour degree to produce the initial solution. Table 3.12 shows the results using saturation degree. The success rate is the percentage of times the algorithm achieved a feasible timetable. If a feasible solution had not been found by 1,000,000 iterations the algorithm ended. The average time is the average computational time the algorithm took to finish either; by reaching 1,000,000 iterations or by finding a feasible solution. These averages were calculated using successful and unsuccessful runs. The average iterations shows the average number of iterations the algorithm took before finding a feasible solution or ending.

Colour degree produced feasible solutions in 100% of the 20 runs in all but two of the instances. Saturation degree also found feasible solutions in all instances however failed to find them 100% of the time in six instances.

Colour degree was more computationally efficient as it was significantly quicker than saturation degree in all of the runs. Saturation degree on average required many more iterations in 15 of the 24 instances however, even when colour degree needed more iterations it was still substantially quicker.

The results conclude that both selection heuristics performed well and achieved a feasible timetable at least 80% of the time in all instances.

In comparison with the neighbourhood operators method described in Chapter 3, which found feasible solutions in three instances, this method has clearly proven to be a superior method

Table 3.11: Results using Colour degree with TabuCol

Instance	Success rate	Avg iterations	Avg time (secs)
Early1	100%	63932	0.92
Early2	100%	237406	0.92
Early3	100%	1072	0.41
Early4	100%	3545	0.41
Early5	100%	10047	1.03
Early6	100%	15765	1.06
Early7	100%	4986	0.32
Early8	100%	1780	0.32
Late1	100%	220759	0.94
Late2	85%	551928	0.94
Late3	100%	154	0.36
Late4	100%	20403	0.36
Late5	100%	11329	1.02
Late6	100%	21442	1.01
Late7	100%	965	0.25
Late8	100%	1556	0.25
Hidden1	100%	25	0.08
Hidden2	100%	3020	0.25
Hidden3	100%	128323	0.73
Hidden4	100%	478	1.23
Hidden5	100%	20109	1.63
Hidden6	80%	622655	2.72
Hidden7	100%	218403	1.53
Hidden8	100%	215688	1.56

for solving this particular problem. The partial solution method produced feasible solutions in all instances and needed far less computational time.

3.3.1 Comparison of feasible timetables

We have used 20 different random seeds for each instance to determine different solutions. The partial solution method described in Section 3.3 produced feasible solutions in 98.5% of these final solutions. To determine how similar our feasible solutions are in each instance we have compared all pairs of the feasible solutions. The comparison can give us an idea of how many feasible solutions we have found. We found that, generally, the average percentage of events that

Table 3.12: Results using Saturation degree with TabuCol

Instance	Success rate	Avg iterations	Avg time (secs)
Early1	100%	137971	12.07
Early2	90%	345879	12.08
Early3	100%	1050	3.87
Early4	100%	1279	3.91
Early5	100%	11174	18.38
Early6	100%	16189	18.42
Early7	100%	4487	2.82
Early8	100%	3203	2.76
Late1	100%	264610	12.06
Late2	90%	494491	11.79
Late3	95%	59661	2.53
Late4	100%	6977	2.62
Late5	100%	11993	18.56
Late6	100%	24504	18.38
Late7	100%	1419	1.82
Late8	100%	1073	1.86
Hidden1	100%	3	0.54
Hidden2	100%	3869	2.47
Hidden3	95%	175647	6.32
Hidden4	100%	696	12.71
Hidden5	100%	75004	32.09
Hidden6	100%	419403	47.05
Hidden7	95%	209648	24.44
Hidden8	80%	305781	19.29

were in the same timeslot in both timetables being compared were higher on timetables with lower density and lower on high densities. There was a moderate negative correlation between the two variables with a Spearman's rank correlation coefficient of -0.536. The results are shown in Table 3.13 and the corresponding scatter plot in Figure 3.9. The majority of the percentages are low, we therefore have fairly different feasible timetables in each instance. This means we have many different starting points for the optimisation of these timetables and it encourages us to think that improvement methods may work in the following chapter. There is a clear outlier in Figure 3.9 where even though the instance has a high density it seems to have found many different feasible solutions as similarity is also high. It is also interesting to note that we have shown in this chapter that some local search approaches have struggled to find

a feasible solution so it is interesting to find that the partial solution method found mulitple different feasible solutions in each instance.

Table 3.13: The comparison of pairs of timetables to determine how similar the feasible timetables are.

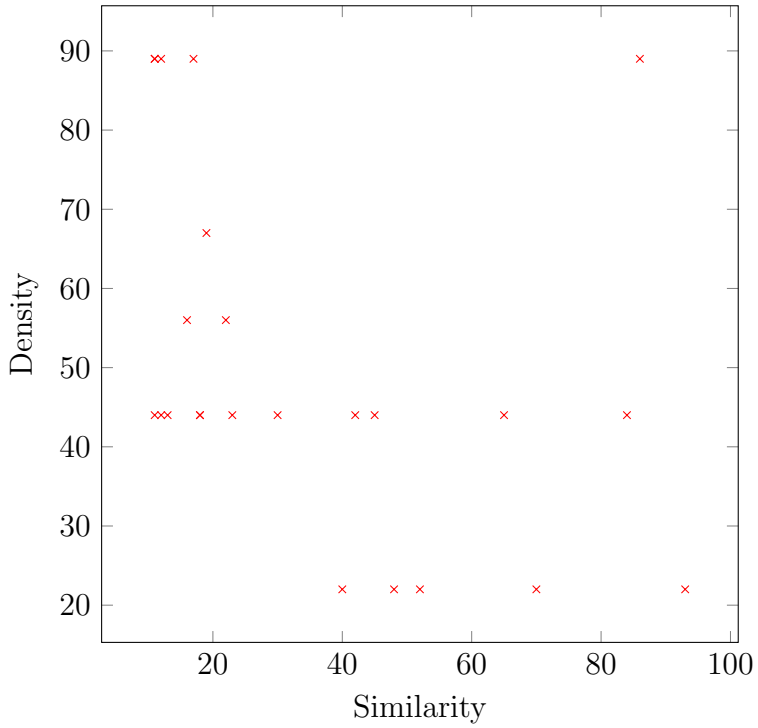
Instance	Average similarity	Timetable density
Early1	17%	89%
Early2	11%	89%
Early3	70%	22%
Early4	48%	22%
Early5	18%	44%
Early6	13%	44%
Early7	40%	22%
Early8	52%	22%
Late1	11%	89%
Late2	12%	89%
Late3	84%	44%
Late4	30%	44%
Late5	18%	44%
Late6	12%	44%
Late7	45%	44%
Late8	65%	44%
Hidden1	93%	22%
Hidden2	42%	44%
Hidden3	19%	67%
Hidden4	86%	89%
Hidden5	16%	56%
Hidden6	22%	56%
Hidden7	23%	44%
Hidden8	11%	44%

3.4 Conclusions

In this chapter we have presented two approaches for finding feasibility with the post enrolment-based course timetabling problem.

The first approach relaxed HC1 and HC5 to enable all events to be scheduled. The focus was then to attempt to eliminate any violations of HC1 and HC5 using various neighbourhood

Figure 3.9: A scatter plot of the relationship between the similarity of the feasible solutions and the density of each instance



operators. The number of violations was significantly reduced mainly by the move and swap operators and the more complex Kempe chains. This method found feasible solutions in three of the 24 instances.

We used two cost functions in the first approach based upon the two relaxed constraints, HC1 and HC5. The first cost function, cost1, was the sum of all students that are required to attend more than one event at any time, plus the sum of all students that are required to attend the events that violate the precedence constraint. The second cost function used, cost2, was the sum of all the events that contain common students that are scheduled at the same time, plus the sum of all events that violate the precedence constraint.

We found that using cost2 produced higher reduction in the cost function so this could be seen as a more successful cost function to use. Throughout the first method of adding neighbourhood operators, cost2 had made a higher percentage of improvement to the cost function. For instance, after moves, swaps and Kempe chains had been applied cost1 had reduced the initial

cost by 82% and cost2 had reduced the initial cost by 96%.

The second approach did not relax any constraints and as a result was forced to leave some events unscheduled to maintain a valid timetable. The events are then scheduled in the least disruptive timeslot. The least disruptive timeslot is the one where no clashes exist once the event is scheduled. If this scenario does not exist it is scheduled in the timeslot that results in the least number of events being removed to ensure no clashes are present. This method produced a feasible solution in 98.5% of the runs (20 random seeds on 24 instances). We have also seen that many of the 20 solutions found for each instance vary significantly from each other which gives us a strong starting point for the next chapter which will try to optimise these feasible solutions in terms of the number of soft constraint violations. It is also worth noting that although we were not considering the time limit imposed by the competition, as outlined in Section 3.1, our run times were within the limit with an average run time of 0.55 seconds.

It is obvious to conclude from this chapter that leaving events unscheduled, to later place them in the timetable, whilst at all times maintaining a feasible solution is far superior to relaxing constraints and trying to move around a highly constrained solution space to find a feasible solution.

Chapter 4

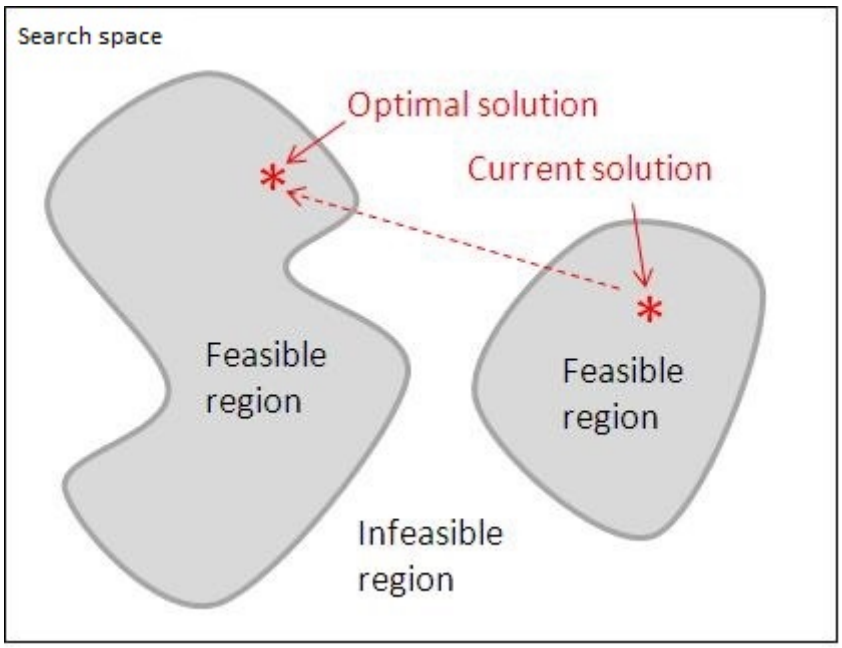
Stage two of a two-stage approach: Improve solution by minimising soft constraint violations

The post enrolment-based course timetabling problem is represented by a search space containing all possible (feasible and infeasible) timetables. The current timetable is contained within a feasible region of this search space and we now need to search for a higher quality solution. The aim of stage two is to minimise the number of soft constraint violations whilst maintaining feasibility.

A basic condition of a search method is that the solution space is connected. Two solutions within a solution space are connected if and only if there exists a sequence of neighbourhood moves that can transform one into the other. In highly constrained problems, such as the one we are studying, regions where feasible solutions exist may be poorly connected or even disconnected in the search space. The search process may not be able to move from one feasible solution to another unless infeasible solutions or solutions with a higher cost function value are permitted; see Figure 4.1. In this chapter, we will investigate methods that may improve the

connectivity which should enable a broader search to take place.

Figure 4.1: An example of how the search space could appear.



A solution is penalised equally for each occurrence of the following soft constraint violations:

SC1 - a student has a class in the last slot of the day;

SC2 - a student has more than two classes consecutively;

SC3 - a student has a single class on a day.

The sum of the number of violations of these constraints is called the Soft Constraint Penalty (SCP), this is the measure of the quality of the solution. The objective of this chapter is to minimise SCP whilst maintaining feasibility. The mathematical formulation can be seen in Chapter 3. The average SCP for each instance at the end of stage one is shown in Table 4.1.

Table 4.1: The average number of soft constraint violations in the feasible timetables for each instance.

Instance	Average SC1	Average SC2	Average SC3	Average SCP
Early1	1220	1526	155	2901
Early2	1302	1267	205	2774
Early3	1376	1139	888	3403
Early4	1080	672	1024	2776
Early5	593	1061	96	1750
Early6	576	1037	84	1697
Early7	732	336	511	1579
Early8	670	419	504	1593
Late1	1316	1528	219	3063
Late2	1260	1239	261	2760
Late3	1100	1008	853	2961
Late4	1333	758	1086	3177
Late5	562	1176	92	1830
Late6	694	994	120	1808
Late7	700	325	599	1624
Late8	637	559	409	1605
Hidden1	325	1943	120	2388
Hidden2	1252	1282	187	2721
Hidden3	1130	713	1289	3132
Hidden4	999	1029	843	2871
Hidden5	567	1214	59	1840
Hidden6	1225	1324	191	2740
Hidden7	2557	3184	701	6442
Hidden8	964	862	1237	3063
Average	1007	1108	489	2604

4.1 Disallow SC1 and SC3 violations

It is known that at least one feasible solution exists in all instances (McCollum et al. [2010]), it is also known that the optimal solution has been found for at least four of the 24 instances. Since we know that optimal solutions exist, this section focusses on treating soft constraints as hard constraints in order to eliminate the SCP and find an optimal solution. This experiment is actually run independently and does not begin with a feasible solution from the first stage. However, it does follow the same method as the method described in Section 3.3. Since the partial solution method was successful in finding feasible solutions with the five specified

hard constraints, we first experimented with classing SC1 and SC3 also as hard constraints. Therefore, violations of SC1 and SC3 were forbidden from the beginning of stage one.

Events are ordered dynamically, using colour degree, which prioritises the events that have the largest amount of conflicts with events that have already been scheduled. The timeslots are listed in ascending order of the number of events that can be feasibly scheduled in them. We then attempt to systematically schedule each event in the timeslots in order. The event is scheduled in the first one that maintains feasibility of the timetable. If there are no timeslots that maintain feasibility then the event would remain unscheduled. Also, in this case, for the timetable to remain feasible, SC1 and SC3 cannot be violated.

Next, we attempt to schedule the events on the unplaced list and remove the smallest number of events as possible from the timetable to maintain feasibility. When events have to be removed from the timetable they are added to the end of the unplaced list and the process continues until either; a (complete) feasible solution is found or 1,000,000 iterations have been performed.

Table 4.2 shows the average number of events that had to remain unscheduled, the average distance to feasibility (defined in Chapter 3) and the average number of SC2 violations that were present in the incomplete timetable. Unfortunately, including SC1 and SC3 with the hard constraints made the problem impractical to solve and we did not manage to find a feasible solution in any of the 24 instances with, on average, 30% of the events remaining unscheduled. The same experiment could be repeated with only classifying SC2 or SC3 as hard constraints for example. Classifying SC1 as a hard constraint means that no event can be scheduled in the last timeslot of each day which could be too restrictive. In another experiment we could count the number of students who have an event in the last timeslot of the day instead of forbidding this.

Table 4.2: Results from disallowing violations of SC1 and SC3 from the beginning of Stage one.

Instance	Average number of events left unplaced	Average DTF	Average SC2 violations
Early1	138	3980	456
Early2	147	4078	579
Early3	41	3414	1597
Early4	52	3676	1729
Early5	130	2192	196
Early6	130	2226	254
Early7	54	2007	801
Early8	57	2021	901
Late1	142	4201	553
Late2	154	4426	628
Late3	49	3916	1646
Late4	56	4024	1720
Late5	142	2395	282
Late6	136	2302	269
Late7	52	1810	940
Late8	47	1622	777
Hidden1	16	2052	318
Hidden2	67	3719	473
Hidden3	90	4415	1814
Hidden4	89	3217	1568
Hidden5	155	2044	223
Hidden6	231	4313	561
Hidden7	146	9014	985
Hidden8	126	4337	1557
Average	102	3392	868

4.2 Improving the feasible solution

Different neighbourhood operators produce different landscapes. Therefore, we investigated how the solution quality changed with each of the neighbourhood operators described in Section 3.2, namely; moves, swaps, timeslot swaps, the Hungarian method and Kempe chains. In this section we will discuss the operators that were the most successful for optimising the feasible solutions in terms of the SCP for stage two.

4.2.1 Move and swap operators

As concluded in Chapter 3, the move and swap operators can make quick improvements to the cost function. All possible moves and swaps were considered and the move or swap that makes the biggest improvement to the cost function was performed, provided it maintained feasibility. Tabu search (as described in Section 3.2.2.1) was used, so we also ensured that the move or swap was not considered tabu. Table 4.3 shows the average SCP of each instance once moves and swaps have been implemented, the percentage of improvement from the initial SCP and the average computational time. The move and swap operators were repeated until either; 1,000,000 iterations have passed, the SCP has not improved in 1,000 iterations or the SCP reaches zero. The SCP on average decreased by 36% in 0.85 seconds over the 24 instances. This reinforces the point that the simple move and swap operators are easy to implement and quick to improve the cost function.

4.2.2 Kempe chains and non-clashing swaps

Initial experiments showed that Kempe chains are an effective way of successfully searching the search space and decreasing the SCP. The Kempe chains neighbourhood takes two sets of mutually conflicting events from two different timeslots. These sets are then swapped without causing any further conflicts. All Kempe chains are considered and tabu search is used to ensure that we are capable of escaping from a local optimum. Tabu search accepts worsening moves when there are no further moves that can improve the solution quality. The reverse of any moves in the chain that are performed are saved on a tabu list for a set number of iterations so that any event in the chain cannot move back to its previous position. The number of iterations that the move remains on the tabu list is called the tabu tenure. This is usually unique for each problem. We performed some tests on different tabu tenures such as; constant, random and variable. These tests showed that a random tabu tenure performed well on this problem. Specifically, a random number between 150 and 200 iterations, so that a new random number

Table 4.3: SCP after move and swap operator has been used for optimisation.

Instance	Initial SCP	Average SCP	% improvement	Average computational time (sec)
Early1	2901	1923	34%	0.94
Early2	2774	1916	31%	0.93
Early3	3403	1823	46%	0.51
Early4	2776	1893	32%	0.50
Early5	1750	1154	34%	0.88
Early6	1697	1096	35%	0.90
Early7	1579	1116	29%	0.33
Early8	1593	1101	31%	0.34
Late1	3063	1986	35%	1.08
Late2	2760	1871	32%	0.95
Late3	2961	1899	36%	0.48
Late4	3177	1977	38%	0.47
Late5	1830	1247	32%	0.93
Late6	1808	1209	33%	0.92
Late7	1624	1087	33%	0.33
Late8	1605	923	42%	0.33
Hidden1	2388	843	65%	0.13
Hidden2	2721	1697	38%	0.33
Hidden3	3132	2290	27%	0.91
Hidden4	2871	1696	41%	1.37
Hidden5	1840	921	50%	1.41
Hidden6	2740	2508	8%	2.32
Hidden7	6442	3190	50%	1.5
Hidden8	3063	1920	37%	1.52
Average	2604	1637	36%	0.85

in this range is chosen on every iteration. The maximum matching algorithm, as described in Chapter 3, is used to allocate events to suitable rooms in order to maintain feasibility.

Clearly, sets of mutually conflicting events exclude pairs of events that do not clash with each other. Therefore, once all Kempe chains have been considered, all pairs of events that do not clash are then examined to see if they can improve the cost function further. This is repeated until; no soft constraints are violated (SCP = 0), no improvement has been made for 500 iterations, or the number of iterations reaches 100,000. These stopping conditions were chosen by observation of the program.

The results from these experiments are shown in Table 4.4, and Figure 4.2, for the eight early and eight late instances. Also shown are the results from Cambazard et al. [2012] who came first in track two of the ITC 2007, along with Chiarandini et al. [2008] who came third and Lewis [2012] who was a competition organiser but did publish his own algorithm. Cambazard et al. present a hybridisation of local search with constraint programming techniques. They found an optimal solution in four of the 16 instances tested. Chiarandini et al. submitted a two phase algorithm using local search to initially schedule events and PartialCol for any unscheduled events followed by a second phase which uses local search methods to minimise the soft constraint violations. This algorithm produced very competitive results. Lewis' algorithm consisted of three phases. The initial phase relaxes HC5 and inserts as many events as possible, simulated annealing is then used to satisfy HC5 and minimise the soft constraint violations in the third phase. In many of the instances it is clear that Cambazard et al. have the most successful method. This could be due to a number of reasons such as parameter settings, the way it was programmed, the connectivity of the search space due to the neighbourhood operators or the choice of search algorithm (simulated annealing versus our tabu search). More research would be required here to provide a definitive reason.

The technique presented in this thesis can clearly compete with the winning algorithms, outperforming them all in three instances (highlighted bold in the table) and coming second or third in many others. However, it is still the worst performing algorithm in seven of the 24 instances so there is still room for improvement.

Figures 4.3, 4.4 and 4.5 illustrate the relationship between the event size, the number of times the event was moved during the Kempe chain moves and swaps and how much the event contributes to the final SCP. The relationship was significant between all of the variables. We have used the Spearman's rank correlation to determine the relationship between the variables as the data lends itself to non-parametric statistics.

There was a weak negative correlation between the number of times an event was moved and the final contribution of that event to the SCP, with a Spearman's rank correlation coefficient

Figure 4.2: Comparison of our results with two of the competition entrants and the competition organiser.

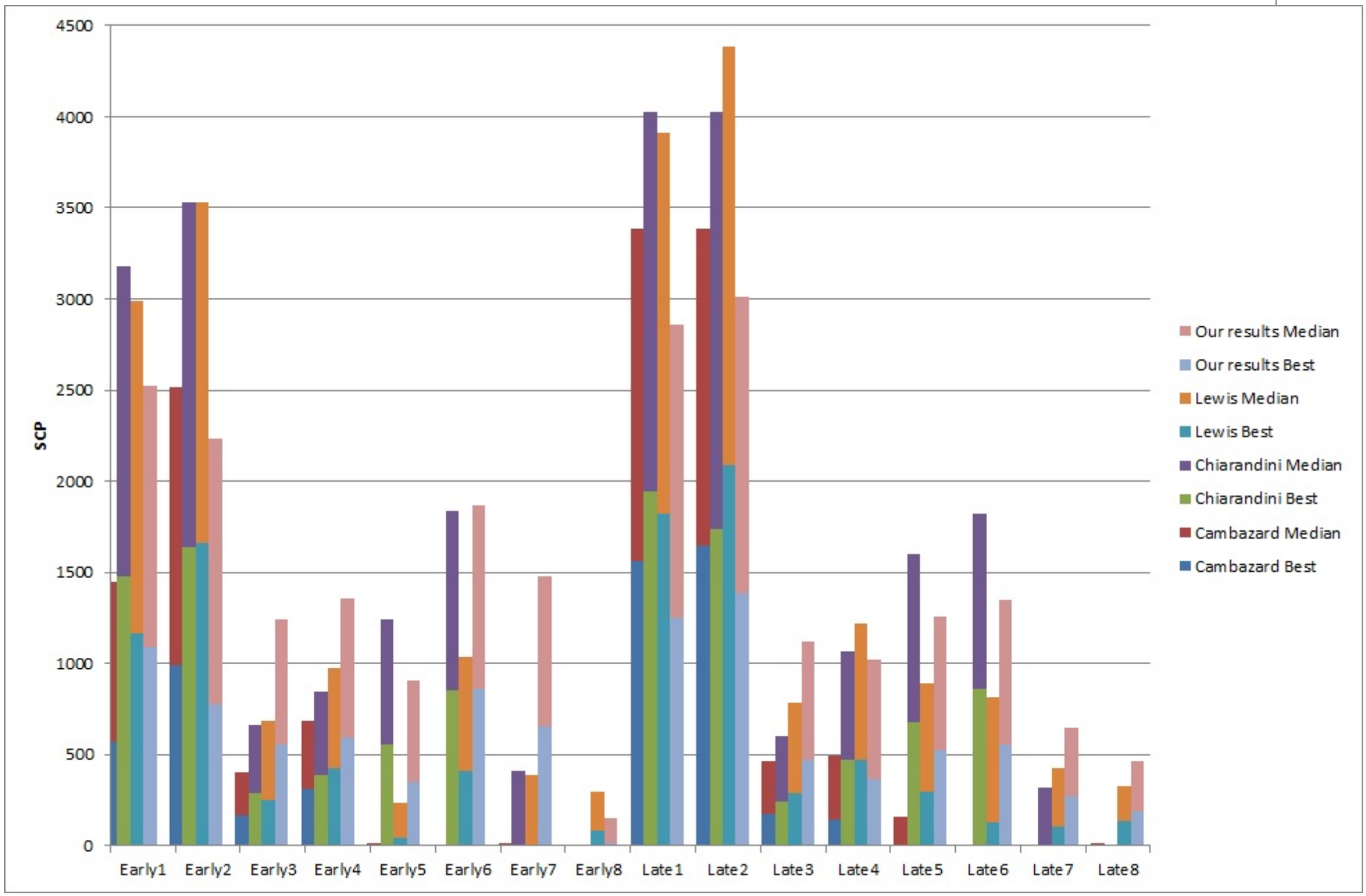


Table 4.4: The best and median results (DTF/SCP) of ten runs obtained by algorithms submitted to the competition. The best scores are shown in bold text.

	Cambazard et al.		Chiarandini et al.		Lewis		Our results	
Instance	Best	Median	Best	Median	Best	Median	Best	Median
Early1	**0/571**	0/877	0/1482	0/1696	0/1166	0/1819	0/1088	0/1434
Early2	0/993	0/1523	0/1635	0/1896	0/1665	42/1866	**0/778**	0/1458
Early3	**0/164**	0/236	0/288	0/374	0/251	0/436	0/556	0/685
Early4	**0/310**	0/372	0/385	0/463	0/424	0/552	0/592	0/768
Early5	**0/5**	0/7	0/559	0/681	0/47	0/190	0/351	0/553
Early6	**0/0**	0/0	0/851	0/985	0/412	0/621	0/859	0/1012
Early7	**0/6**	0/8	0/10	0/403	**0/6**	0/383	0/654	0/823
Early8	**0/0**	0/0	**0/0**	0/1	0/85	0/215	0/11	0/142
Late1	0/1560	0/1823	0/1947	0/2081	0/1819	67/2095	**0/1246**	0/1610
Late2	0/1650	0/1737	0/1741	0/2288	0/2091	110/2293	**0/1390**	0/1621
Late3	**0/178**	0/286	0/240	0/365	0/288	0/496	0/473	0/648
Late4	**0/146**	0/349	0/475	0/593	0/474	0/744	0/368	0/650
Late5	**0/0**	0/160	0/675	0/926	0/298	0/592	0/524	0/731
Late6	**0/1**	0/2	0/864	0/958	0/127	0/690	0/557	0/793
Late7	**0/0**	0/0	**0/0**	0/320	0/108	0/319	0/271	0/375
Late8	0/2	0/11	**0/1**	0/6	0/138	0/192	0/191	0/276

of -0.126.

There was a moderate negative correlation between the size of an event and the number of times it was moved, with a Spearman's rank correlation coefficient of -0.509.

Finally, there was a weak positive correlation between the size of event and the final amount it contributed to the SCP with a Spearman's rank correlation coefficient of 0.12.

Smaller events are moved a large number of times, whereas larger events are not moved many times throughout the run. This is due to smaller events having more available timeslots to move to as the room size is less of a restriction. They are also less likely to clash with other events in a timeslot. Since smaller events have more options of timeslots it is more likely that they can find a timeslot that will incur no hard or soft constraint violations and therefore contribute less to the SCP.

The rest of this chapter will explore ways to further improve these results.

Figure 4.3: A scatter plot of the relationship between the number of times an event was moved and the contribution to the SCP.

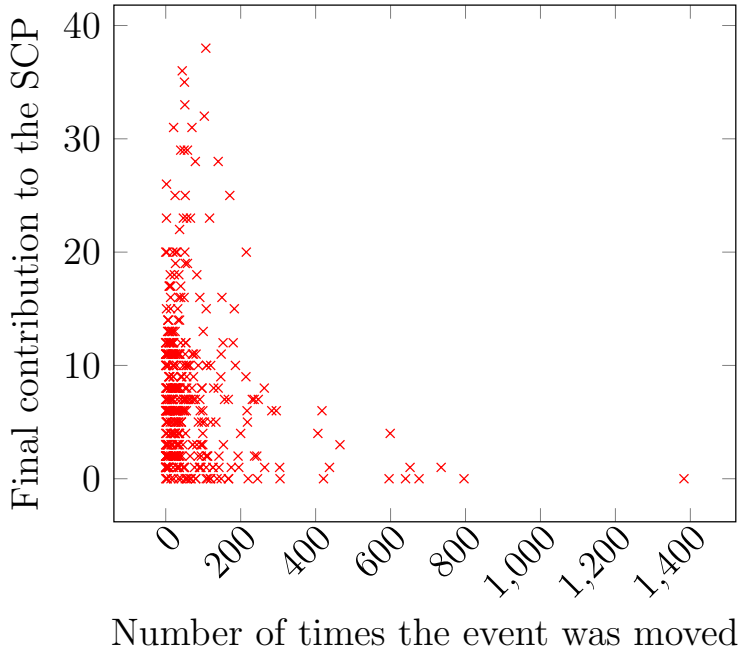


Figure 4.4: A scatter plot of the relationship between the number of times an event was moved and the size of the event.

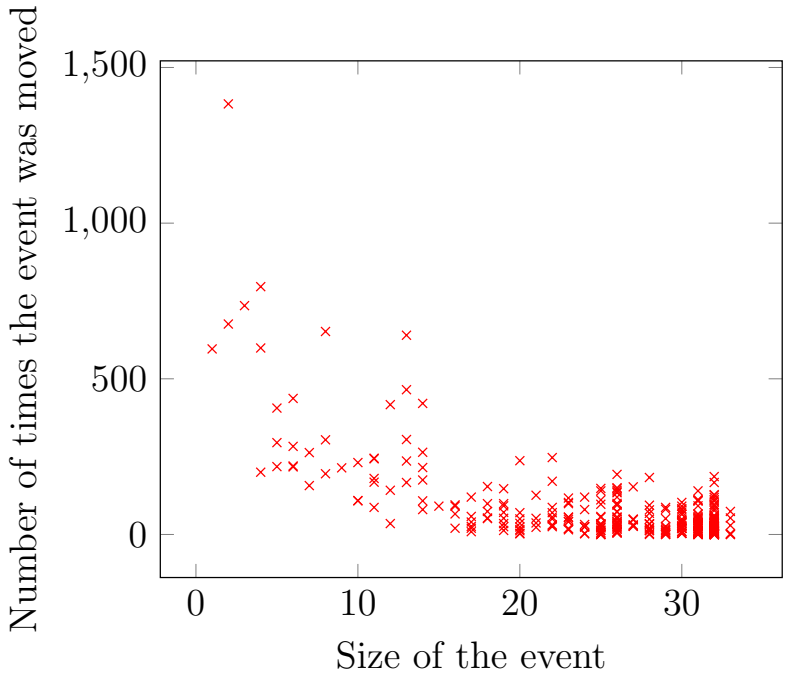
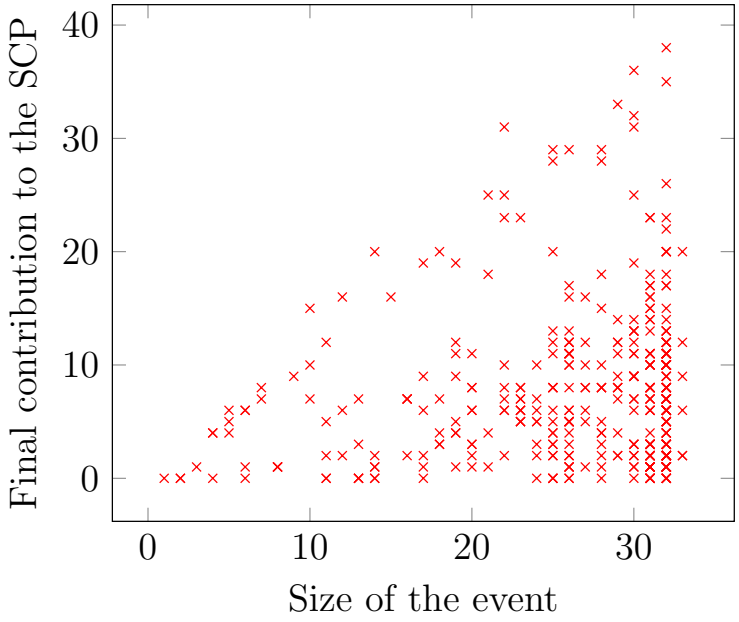


Figure 4.5: A scatter plot of the relationship between the size of the event and the final contribution to the SCP.



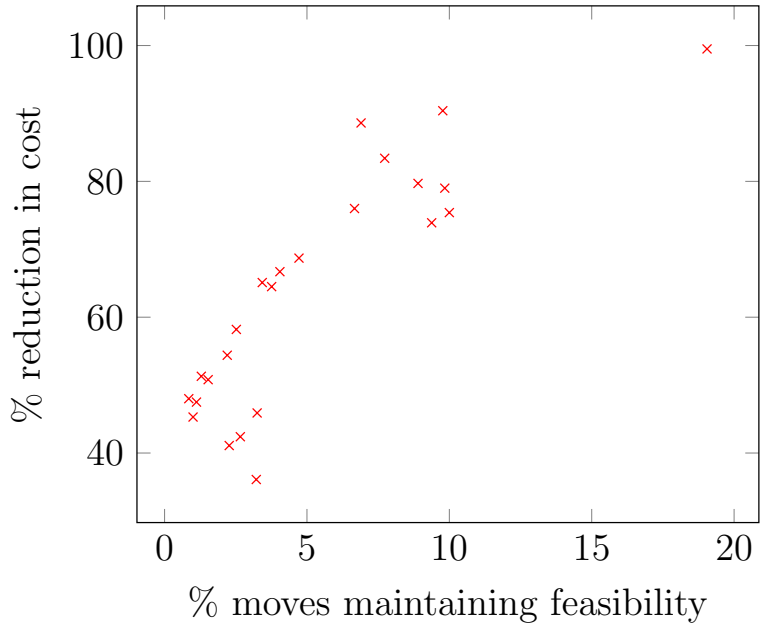
4.3 Feasibility ratio

We gauge the connectivity of the search space by comparing the number of moves considered with the number of moves that maintain feasibility. We call this the 'neighbourhood feasibility ratio'. By testing the relationship between the neighbourhood feasibility ratio and the proportion of the reduction in cost, we found that there was a strong positive correlation between the two variables with a Spearman's rank correlation coefficient of 0.86. Therefore, as the percentage of moves that maintain feasibility increases, so does the percentage of the reduction in cost, see Figure 4.6. This suggests that if we can increase the feasibility ratio then we can decrease the SCP.

The remainder of this section will contain methods we have considered to improve the neighbourhood feasibility ratio (and, by implication, the connectivity).

Improvements can be made via the application of a wide range of more flexible operators or by the temporary relaxation of certain problem constraints.

Figure 4.6: A scatter plot of the relationship between the average neighbourhood feasibility ratio and the average proportion of the reduction in cost over 20 runs for each instance.



4.3.1 Remove events

To improve the feasibility ratio we considered temporarily removing some events from the problem. This will free up more timeslots and rooms for the remaining events to move to and therefore could improve the ratio between the number of possible moves and the number of them that maintain feasibility. This will enable the search to explore more regions of the search space and potentially decrease the SCP for us to then reinsert the removed events for a complete solution.

Our process of choosing which events should be removed involves multiple steps. Firstly, we calculate the proportion of the SCP that each event contributes and sort the events in ascending order by this proportion. We have retained a list of the number of times each event was moved in the optimisation process of Kempe chains and swaps described in Section 4.2.2. This list is arranged in descending order so the event first on the list was moved the most amount of times. Both lists are assigned a rank. These ranks are then combined so that we have an overall ranking of events, the lowest rank being the events that contributes a low percentage

of the SCP and were moved many times. The removed event should, in theory, be one of the easiest to reschedule. It has been moved between timeslots many times, and thus it must have a large number of timeslots where it can be scheduled. The low cost could mean that it does not conflict with many other events and therefore will be easier to schedule.

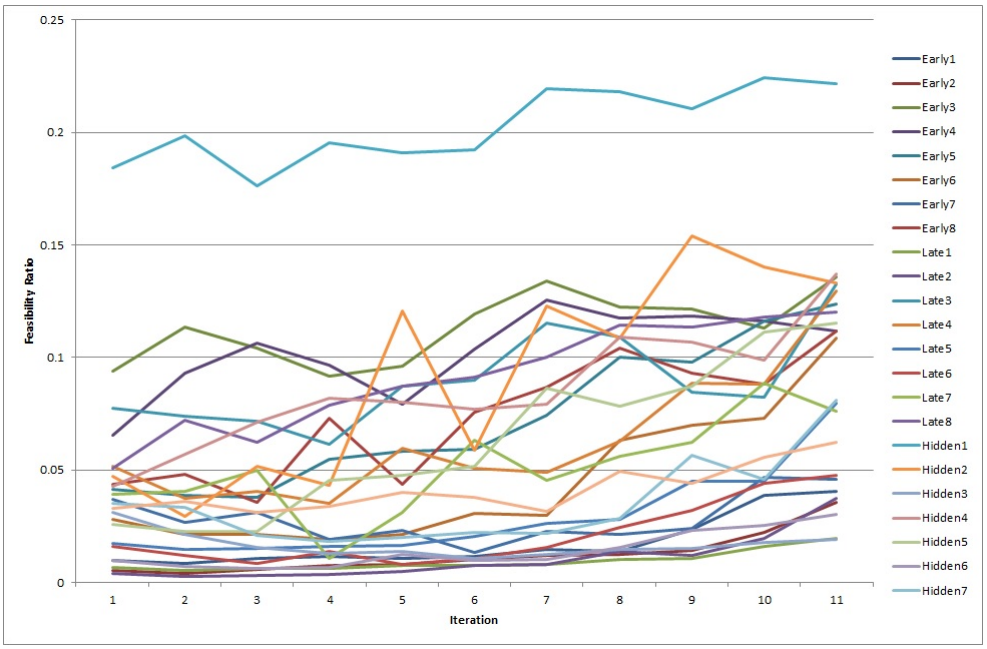
We remove 2.5% of the events and then optimise the solution using the optimisation process of Kempe chains and non-clashing swaps described in Section 4.2.2. We then attempt to feasibly reschedule the events and optimise again. The decision of removing 2.5% of the events was based on observation of the program and found a good balance between not disrupting the timetable too much and removing enough to make a difference to the search.

The maximum matching algorithm is used when we are trying to schedule an event in a timeslot. The algorithm finds in which rooms the event can be scheduled. If one is empty in the timetable then it schedules the event and finishes. If the feasible rooms have events scheduled in them, then these events are added to a queue. For all events in the queue, we find other rooms that they could be moved to. If any of these are empty the events are moved and the algorithm finishes, otherwise the events in these timeslots are again added to the queue. This process is repeated until there is a feasible allocation of events to rooms within the timeslot that does not violate any hard constraints.

It was clear from our experiments that reinserting the events once optimisation has taken place with fewer events only makes small improvements to the feasibility ratio or the SCP. The search will have moved to another area of the search space and therefore be a different solution. However, there were only small changes to the feasibility ratio due to the highly constrained nature of this problem. Following this conclusion we tested this method without reinserting the events therefore continuing the search with an incomplete solution. This obviously makes an easier problem so is not comparable to previous results, but is an indicator of how the method performs on simpler problems. See Figure 4.7 and Figure 4.8 for effects on the feasibility ratio and SCP when the removed events are not reinserted. The figures show all instances for one random seed as an example. It is clear there is a trend of the feasibility ratio increasing and

the SCP decreasing as was expected. On average, the feasibility ratio increased by 201% and the SCP reduced by 51% after the ten iterations each removing 2.5%.

Figure 4.7: A line chart showing the change in feasibility ratio of the problem at each point after events have been removed and the timetable has been optimised.

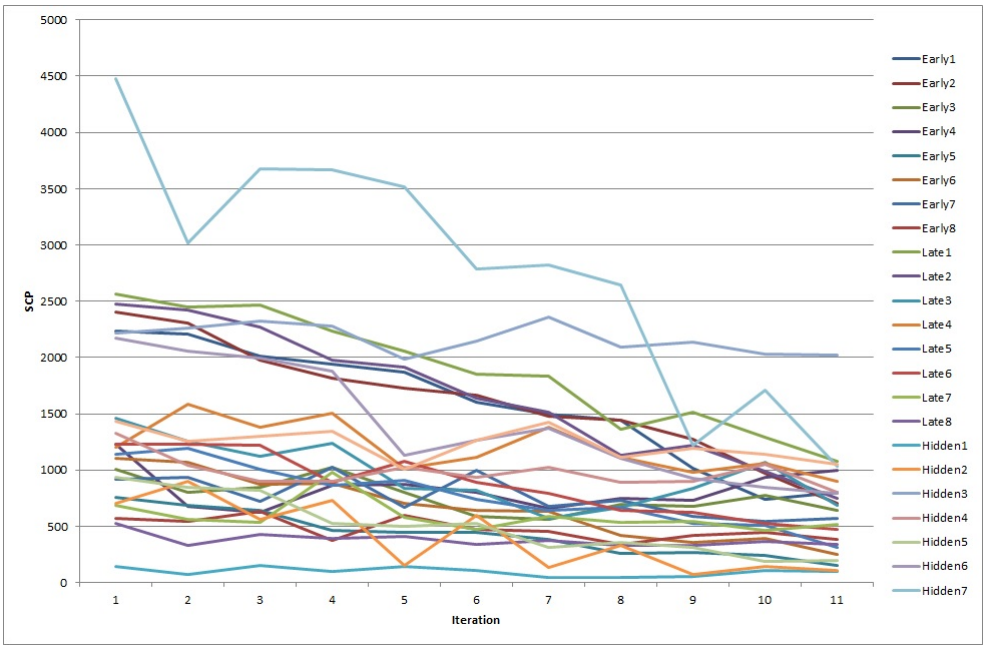


4.3.2 Remove constraints

We also considered removing constraints to test our techniques on easier problems with fewer constraints. This will make more timeslots available to the events and allow them to move more easily within the timetable during the optimisation process and enable us to explore more of the search space. Keeping track of the feasibility ratio will enable us to see how it changes as the problem gets easier.

We begin with the feasible solutions constructed by the partial solution method presented in Section 3.3. We then randomly remove a total of 0.5% from the sets of room, timeslot and clash constraints. This is done proportionally according to the size of these sets. The number of constraints to remove was based upon observation of the program; if too many were removed the problem would become too simple and conversely, if too small a number were removed then the

Figure 4.8: A line chart showing the change in SCP of the problem at each point after events have been removed and the timetable has been optimised.



timetable would not differ enough from the initial solution. The solution is optimised with the smaller set of constraints using the Kempe chains and swap method described in Section 4.2.2. This process is repeated ten times to get an idea of how these neighbourhood operators work.

The results are shown in Table 4.5. Removing constraints had the same effect on the feasibility ratio and SCP as removing events did however, the changes were not quite as large as when events were removed. If we have removed a larger proportion of events on each iteration or performed more iterations we may have seen similar results. Over ten iterations of removing 0.5% of the constraints the feasibility ratio increased, on average, by 38% and the SCP decreased by 24%. This was to be expected as the problem would had a higher ratio of number of moves to the amount of those that maintain feasibility when the problem is less constrained.

Table 4.5: The change in feasibility ratio (FR) and SCP after 0.5% of the constraints are removed on each of the ten iterations.

Instance	Average increase in FR	Average improvement to the SCP
Early1	56%	33%
Early2	53%	26%
Early3	15%	9%
Early4	43%	36%
Early5	20%	4%
Early6	19%	15%
Early7	6%	4%
Early8	31%	6%
Late1	66%	34%
Late2	43%	13%
Late3	41%	54%
Late4	49%	30%
Late5	43%	29%
Late6	43%	16%
Late7	32%	24%
Late8	58%	35%
Hidden1	11%	17%
Hidden2	55%	31%
Hidden3	21%	21%
Hidden4	63%	50%
Hidden5	44%	27%
Hidden6	32%	11%
Hidden7	35%	31%
Hidden8	33%	17%
Average	38%	24%

4.3.3 Adding constraints

This section describes a similar process to Section 4.3.2 where constraints are removed. In this section, we will add constraints to test our techniques on more difficult highly constrained problems.

The process begins with the feasible solutions produced in Section 3.3. The solutions are optimised focussing on the SCP using the Kempe chains and non-clashing swaps described in Section 4.2.2. We then randomly add a total of 0.5% of the constraints to the sets of room,

timeslot and clash constraints. This is done proportionally according to the size of these sets. We add the constraints in such a way that the solution remains feasible but also an optimal solution is still achievable. We were provided with four optimal solutions so that we can ensure it is still possible to eliminate all soft constraint violations even after the constraints have been added. These four optimal solutions were for the instances; Early1, Early8, Late6 and Hidden1. This process is repeated ten times to get an idea of how these neighbourhood operators work. We kept track of the feasibility ratio to see how it changed as the problem gets more difficult.

The results are shown in Table 4.6. On average, the feasibility ratio decreased by 43% and the run ended with a feasibility ratio of 0.06, representing the ratio between the number of possible moves and the number of those that maintain feasibility. With a feasibility ratio of 0.06 it is easy to understand how constrained the problem has become. The solution quality deteriorates when constraints are added at approximately the same rate as it improves when constraints are removed.

Table 4.6: The change in feasibility ratio and SCP when 0.5% of the total constraints are added on each of the ten iterations.

Instance	Average decrease in FR	Average increase to the SCP
Early5	54%	26%
Early8	62%	71%
Late6	45%	14%
Hidden1	10%	11%
Average	43%	31%

4.4 Conclusions

In Chapter 3 we found a feasible solution in each instance. This chapter has focussed on minimising the number of soft constraint violations to improve the solution quality. We examined the effects of some of the more successful neighbourhood operators that were presented in

Chapter 3.

The move and swap operators were effective in making a quick and significant improvement to the SCP. On average, they managed to reduce the SCP by 36% from the initial cost.

Employing Kempe chains and non-clashing swaps after the move and swap operators decreased the SCP on average by 74% overall. Although these operators have proven again to be very effective, further work still needs to be done to achieve optimal solutions.

The feasibility ratio was explored, which is the relationship between the number of possible moves and the number of moves that maintain feasibility. This was done by removing events from the timetable. Once removed the timetable was optimised again to see how the feasibility ratio changed. The events were reinserted after each iteration to gauge if the feasibility ratio had improved for the complete solution. This did not have much effect on the feasibility ratio for the complete problem.

Following these results we tested the effects on the feasibility ratio on less dense problems by removing the events and not rescheduling them as before but leaving them unscheduled. As would be expected, the feasibility ratio significantly increased by, on average, 201% and the SCP decreased by 51% from the initial solution with less events included in the problem.

The performance of these methods was also tested on simpler and more difficult problems by removing and adding constraints. We removed constraints to make the problem simpler and, as expected, the feasibility ratio increased and the SCP decreased. We found that when adding constraints the method quickly deteriorated in finding improvements to the cost function as the problem was already very highly constrained.

Chapter 5

Conclusions and future research

The research presented in this thesis has investigated ways of solving the post-enrolment based course timetabling problem. In particular, it has explored ways of applying local search. This chapter will present a summary of the main conclusions that can be drawn from this research, highlighting the contributions made to the field. Areas are then identified for future work.

5.1 Conclusions

Chapter 2 summarised the existing research into post enrolment-based course timetabling and related topics. It also highlighted the importance of automating the process of timetabling in educational institutions. We have suggested that the problem size and the complexity of real-world problems are not appropriate for exact method approaches, and that local search techniques and metaheuristics are more suited to the problem. The chapter also outlined the methods that have been previously used and due to the level of their success we have chosen to utilise these techniques in this thesis. This chapter can be used as a handy resource, as it provides many important references for anyone who wishes to study this topic.

Chapter 3 described two methods for solving the post enrolment-based course timetabling.

The method described in Section 3.2 searches for a feasible solution whilst relaxing two hard constraints, HC1 and HC5. Neighbourhood operators and tabu search are used to attempt to satisfy the relaxed constraints. The smaller operators, such as the move and swap operators (Sections 3.2.2.1, 3.2.2.2), were very effective methods for making quick improvements to the cost function. However, the timeslot swap operator (Section 3.2.2.3) was not effective in making any improvements to the cost function. This was due to the events having predefined timeslots into which they can be placed (HC4) and it was likely that they could not all be scheduled in the timeslot that they were moving to. The Hungarian method (Section 3.2.3.1) made small improvements to the cost function but with a high computational time. Since this is a highly constrained problem, the events are regularly returned to their original positions and no improvement is made to the cost function. Kempe chains (Section 3.9) were much more successful in improving the value of the cost function. Feasible solutions were found in three of the 24 instances when Kempe chains were used in conjunction with the move and swap operators.

The method described in Section 3.3 searches for a feasible solution without relaxing any constraints. This method uses colour degree and saturation degree to schedule events to form an initial solution. If feasible timeslots are not available for an event then that event is temporarily left unplaced. To insert the unscheduled events, some events that would cause hard constraint violations would need to be removed. If events are removed then they are added on the end of the list of unscheduled events to be inserted later. The unscheduled events are inserted in the least disruptive timeslot. By the least disruptive, we mean the timeslot which would require the least amount of events to be removed. Using this method we obtained feasible solutions in all instances. This method was much more efficient than the neighbourhood operators method and produced feasible solutions in all of the 24 instances.

Chapter 4 focussed on minimising the number of soft constraint violations. To do this, we used the move and swap operators and Kempe chains with non-clashing swaps. These were again successful in making improvements to the cost function, with an average reduction of 74% over

the 24 instances.

In Section 4.3 we investigated ways of increasing the feasibility ratio, which is a comparison between the number of moves considered with the number of moves that maintain feasibility. We considered temporarily removing events and optimising the reduced timetable to then re-schedule the events in feasible positions. It was hoped that this may improve the feasibility ratio by having the ability to explore more of the solution space whilst the events are removed. Small improvements were made to the feasibility ratio and SCP, however, it was encouraging that the search found a different feasible solution. Further investigation may be required to understand the landscape of the search space and how this method can be improved.

We considered this approach without re-inserting the events to determine how it could perform on less dense timetables. Significant improvements were made to the feasibility ratio and the SCP. The feasibility ratio increased, on average, by 201% and the SCP decreased by 51% after ten iterations each removing 2.5% of the events. When the events were removed the feasibility ratio quickly increased. This suggests that how constrained the problem is and the feasibility ratio are inversely proportional, in that, the less constrained the problem, the easier it can be to solve.

We tested our optimisation method on simpler and more complex problems by removing and adding constraints. When constraints had been removed the feasibility ratio increased which implies there is more freedom to explore the search space. Conversely, when constraints were added the feasibility ratio decreased and the SCP increased at approximately the same rate that they improved when constraints were removed.

5.2 Future research

In this section we will discuss ways in which the research presented in this thesis can be continued.

It would be interesting to continue exploring neighbourhood operators to further the research carried out in Section 3.2 to expand on the number of feasible solutions found using this method. We produced feasible solutions in three of the 24 instances and it may be possible to find more feasible solutions in this way had more neighbourhood operators been added.

The operators described in Section 3.2 were performed until the stopping criteria was reached and the next neighbourhood operator would begin. Each neighbourhood operator was only initiated once. A version of Variable Neighbourhood Search (Mladenovic and Hansen [1997]) could be introduced, so at the end of the process all the operators are repeated. This could continue until no further improvement could be made to the cost function. This final solution would then be a local optimum. Refer to the flow diagram shown in Figure 5.1 for an example of how this could be implemented.

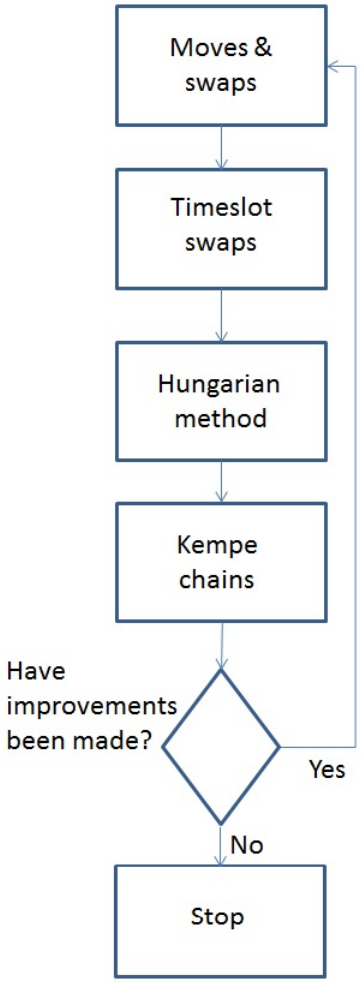
In Section 4.2 we included SC1 and SC3 in the set of hard constraints which made the problem even more highly constrained. We could further investigate how the algorithm performs with larger or more highly constrained problem instances. The research conducted for this thesis was carried out on the track two of the ITC 2007 which has 24 instances and, of which, the largest includes 600 events. There exists many other problem instances that are available to test techniques on. For example, the partial solution method could be applied to any of the instances described in Section 2.3.

The partial solution method presented in Chapter 3 and optimisation techniques presented in Chapter 4 could be applied to other scheduling problems, including transport scheduling or sports scheduling. In addition, they could also be employed to solve other optimisation problems, such as the bin-packing problem or vehicle routing.

To achieve further improvements to the SCP in Chapter 4 further neighbourhood operators could be added to the process. However, this could have a detrimental effect on the feasibility ratio.

In Chapter 4 we have explored removing events and constraints. We could have extended the

Figure 5.1: A flow chart outlining how a version of a variable neighbourhood search could be implemented using the neighbourhood operators described in Section 3.2.



experiments on removing constraints to see how many constraints would need to be removed before an optimal solution was found. We could also relax the problem by adding rooms and timeslots to give more freedom for search methods.

In Section 4.2 we disallowed violations of SC1 and SC3 because they had fewer violations than SC2. This could be investigated further by only classing SC1, SC2 or SC3 as hard constraints and disallowing them from the beginning of stage one. If only one soft constraint is classed as a hard constraint the problem would not be as constrained as when we attempted to find a solution with two classed as hard constraints. If a feasible solution was achieved it would leave

then only two soft constraints to be minimised in further stages.

When we remove constraints in Section 4.3.2, we remove constraints at random. Alternatively, we could use heuristics to choose which constraints to remove and in which order.

It would be interesting to explore how the methods presented in this thesis would perform on real-world variants of the problem. It is vital for many industries to have a method for efficient scheduling in place. For instance, in healthcare, the National Health Service (NHS) are frequently under great scrutiny to achieve targets in terms of the time in which patients are seen or arrival times of ambulances. They could benefit from some of our methods being applied to minimise these times. As an indication of how large the problem is the NHS employed 369,868 qualified nursing staff in 2012[1], which had increased by 34,006 over the previous ten years. Hospitals either try to produce adequate solutions by hand or use a number of programs available to purchase such as *SMART Workforce Management*[2]. The techniques used in this thesis could be adapted to find feasible solutions to a problem such as nurse rostering.

[1] http://www.nhsconfed.org/priorities/political-engagement/Pages/NHS-statistics.aspx#staff

[2] http://www.smart-rostering.co.uk/resource/uploads/documents/Product-Sheets/SMART-RPC-eBrochure.pdf

Bibliography

S. Abdullah and H. Turabieh. On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems. *Information Sciences*, 191(0): 146 – 168, 2012.

S. Abdullah, E. K. Burke, and B. McCollum. A hybrid evolutionary approach to the university course timetabling problem. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1764 –1768, 2007.

S. Abdullah, H. Turabieh, B. McCollum, and P. McMullan. A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, 18(1):1–23, 2012.

E. Angel and V. Zissimopoulos. On the landscape ruggedness of the quadratic assignment problem. *Theoretical computer science*, 263(1):159–172, 2001.

H. Arntzen and A. Løkketangen. A tabu search heuristic for a university timetabling problem. In *Metaheuristics: Progress as Real Problem Solvers*, Operations Research/Computer Science Interfaces Series, pages 65–86. 2005.

H. Asmuni, E. K. Burke, J. Garibaldi, and B. McCollum. Fuzzy multiple heuristic orderings for examination timetabling. In *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 334–353. 2005.

A. S. Asratian and D. de Werra. A generalized class-teacher model for some timetabling problems. *European Journal of Operational Research*, 143(3):531 – 542, 2002.

J. A. D. Atkin, E. K. Burke, J. Greenwood, and D. Reeson. Hybrid metaheuristics to aid runway scheduling at London Heathrow airport. *Transportation Science*, 41(1):90–106, 2007.

M. Atsuta, K. Nonobe, and T. Ibaraki. ITC-2007 track 2: An approach using general CSP solver. *http://www.cs.qub.ac.uk/itc2007/winner/bestcoursesolutions/Atsuta_et_al.pdf*, 2007.

V. Bardadym. Computer-aided school and university timetabling: The new wave. In *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 22–45. 1996.

D. P. Bertsekas and D. A. Castañon. Parallel asynchronous hungarian methods for the assignment problem. *ORSA Journal on Computing*, 5(3):261–274, 1993.

G. D. Birkhoff. The reducibility of maps. *American Journal of Mathematics*, 35(2):115–128, 1913.

I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, 2008.

O. C. Bosquez, P. P. Parra, and F. Lengyel. Towards a deterministic algorithm for the international timetabling competition. *Proceedings of the 17th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2010.

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

S. Broder. Final examination scheduling. *Commun. ACM*, 7(8):494–498, 1964.

M. Bufé, T. Fischer, H. Gubbels, C. Häcker, O. Hasprich, C. Scheibel, K. Weicker, N. Weicker, M. Wenig, and C. Wolfangel. Automated solution of a highly constrained school timetabling problem - preliminary results. In *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 431–440. 2001.

B. Bullnheimer. An examination scheduling model to maximize students' study time. In *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pages 78–91. 1998.

E. K. Burke and J. Landa Silva. The design of memetic algorithms for scheduling and timetabling problems. In *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 289–311. 2005.

E. K. Burke and J. P. Newall. Solving examination timetabling problems through adaption of heuristic orderings. *Annals of Operations Research*, 129:107–134, 2004.

E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266 – 280, 2002.

E. K. Burke, D. Elliman, and R. Weare. A genetic algorithm based university timetabling system. *2nd East-West International Conference on Computer Technologies in Education*, pages 35 – 40, 1994.

E. K. Burke, D. G. Elliman, and R. F. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 605–610, 1995.

E. K. Burke, D. Elliman, P Ford, and R. Weare. Examination timetabling in British universities: A survey. In *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 76–90. 1996a.

E. K. Burke, J. Newall, and R. Weare. A memetic algorithm for university exam timetabling. In *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 241–250. 1996b.

E. K. Burke, K. Jackson, J. H. Kingston, and R. Weare. Automated university timetabling: The state of the art. *The Computer Journal*, 40(9):565–571, 1997.

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

E. K. Burke, J. Li, and R. Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484–493, 2010.

H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos. Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals of Operations Research*, 194(1):111–135, 2012.

M. W. Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34(2):193 – 202, 1986.

M. W. Carter and M. Gendreau. A practical algorithm for finding the largest clique in a graph. Technical Report 84-08, University of Toronto, Canada, 1984.

M. W. Carter and G. Laporte. Recent developments in practical examination timetabling. In *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 1–21. 1996.

M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *The Journal of the Operational Research Society*, 47(3):373–383, 1996.

S. Casey and J. M. Thompson. Grasping the examination scheduling problem. In *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 232–244. 2003a.

S. Casey and J. M. Thompson. Grasping the examination scheduling problem. In *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 232–244. 2003b.

S. Ceschia, L. Di Gaspero, and A. Schaerf. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research*, 39(7):1615 – 1624, 2012.

G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein. Register allocation via coloring. *Computer Languages*, 6(1):47 – 57, 1981.

M. Chiarandini, K. Socha, M. Birattari, and O. Rossi-Doria. International timetabling competition. A hybrid approach. Technical Report AIDA-03-049, Intellectics Group, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, 2003.

M. Chiarandini, C. Fawcett, and H. H. Hoos. A multiphase modular heuristic solver for post enrollment course timetabling. *Journal of Scheduling*, pages 2–3, 2008.

A. J. Cole. The preparation of examination time-tables using a small-store computer. *The Computer Journal*, 7(2):117–121, 1964.

A. Colorni, M. Dorigo, and V. Maniezzo. Genetic algorithms and highly constrained problems: The time-table case. In *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 55–59. 1991.

A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for high school timetabling. *Computational Optimization and Applications*, 9:275–298, 1998.

P. Corr, B. McCollum, M. McGreevy, and P. McMullan. A new neural network based construction heuristic for the examination timetabling problem. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 392–401. 2006.

D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48(3):295–305, 1997.

P. Côté, T. Wong, and R. Sabourin. A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. In *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 294–312. 2005.

J. C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge*, 26:245–284, 1996.

D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19 (2):151 – 162, 1985.

J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3:159–168, 1990.

L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. In *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. 2001.

L. Di Gaspero and A. Schaerf. Multi-neighbourhood local search with application to course timetabling. In *Practice and Theory of Automated Timetabling IV*, pages 262–275. 2003.

M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: An autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991a.

M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Dip. Elettronica, Politecnico di Milano, Italy, 1991b.

R. Dorne and J. K. Hao. A new genetic local search algorithm for graph coloring. In *Parallel Problem Solving from Nature - PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 745–754. 1998.

R. Dorne and J. K. Hao. Tabu search for graph coloring, t-coloring and set t-colorings. In *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. 1999.

K. A. Dowsland. Off-the-peg or made-to-measure? timetabling and scheduling with sa and ts. In *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pages 37–52. 1998.

K. A. Dowsland and J. M. Thompson. Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, 56:426–438, 2004.

K. A. Dowsland and J. M. Thompson. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156(3):313 – 324, 2008.

G. Dueck. New optimization heuristic: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104:86–92, 1993.

M. Eley. Ant algorithms for the exam timetabling problem. In *Proceedings of the 6th international conference on Practice and theory of automated timetabling VI*, PATAT'06, pages 364–382, 2007.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153 (1):3 – 27, 2004.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1996.

P. Galinier and J. K. Hao. Tabu search for maximal constraint satisfaction problems. In *Principles and Practice of Constraint Programming-CP97*, pages 196–208. 1997.

P. Galinier and J. K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2):267 – 279, 2008.

M. Gamache, A. Hertz, and J. O. Ouellet. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers & Operations Research*, 34 (8):2384 – 2395, 2007.

L. M. Gambardella, É. Taillard, and G. Agazzi. MACS-VRPTW: A multiple colony system for vehicle routing problems with time windows. In *New Ideas in Optimization*, pages 63–76, 1999.

A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology*, 35(1):8 – 14, 1986.

M. Garey, D. Johnson, and H. So. An application of graph coloring to printed circuit testing. *IEEE Transactions on Circuits and Systems*, 23(10):591 – 599, 1976.

C. A. Glass. Bag rationalisation for a food manufacturer. *Journal of the Operational Research Society*, 53(5):544–551, 2002.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986.

F. Glover. Tabu search methods in artificial intelligence and operations research. *ORSA Artificial Intelligence Newsletter*, 1987.

F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

F. Glover, M. Parker, and J. Ryan. Coloring by tabu branch and bound. In *Cliques, Coloring and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 76–90. 1996.

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*, pages 1–19, 2010.

A. Gunawan, K. M. Ng, and K. L. Poh. A hybridized lagrangian relaxation and simulated annealing method for the course timetabling problem. *Computers & Operations Research*, 39(12):3074 – 3088, 2012.

P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In *Meta-Heuristics*, pages 433–458. Springer, 1999.

P. J. Heawood. Map-colour theorem. *Quarterly Journal of Mathematics*, 24:332–338, 1890.

A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39: 345–351, 1987.

S. N. Jat and S. Yang. A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. *Journal of Scheduling*, 14(6):617–637, 2011.

R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 43(4):85–103, 1972.

A. B. Kempe. On the geographical problem of the four colours. *American Journal of Mathematics*, 2(3):193–200, 1879.

G. Kendall. Scheduling English football fixtures over holiday periods. *Journal of the Operational Research Society*, 59:743 – 755, 2008.

S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5-6):975–986, 1984.

P. Kostuch. The university course timetabling problem with a three-phase approach. In *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 109–125. 2005.

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30:167–190, 2008.

R. Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers & Operations Research*, 36(7):2295 – 2310, 2009.

R. Lewis. A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. *Annals of Operations Research*, 194(1):273 – 289, 2012.

R. Lewis and B. Paechter. New crossover operators for timetabling with evolutionary algorithms. *5th International Conference on Recent Advances in Soft Computing*, 5:189–195, 2004.

R. Lewis and J. M. Thompson. On the application of graph colouring techniques in round-robin sports scheduling. *Computers & Operations Research*, 38(1):190 – 204, 2011.

R. Lewis, B. Paechter, and B. McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. *Cardiff accounting and finance working papers*, 2007.

R. Lewis, J. M. Thompson, C. Mumford, and J. Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers & Operations Research*, 39(9):1933 – 1950, 2012.

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

Z. Lü and J. K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241 – 250, 2010b.

E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302 – 316, 2008.

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*, 22:120–130, 2010.

P. McMullan and B. McCollum. Dynamic job scheduling on the grid environment using the great deluge algorithm. In *Parallel Computing Technologies*, volume 4671 of *Lecture Notes in Computer Science*, pages 283–292. 2007.

D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333 – 346, 2002.

L. Merlot, N. Boland, B. Hughes, and P. Stuckey. A hybrid algorithm for the examination timetabling problem. In *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 207–231. 2003.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087, 1953.

G. A. Mills-Tettey, A. Stentz, and M. B. Dias. The dynamic hungarian algorithm for the assignment problem with changing costs. Technical Report CMU-RI-TR-07-27, Carnegie Mellon University, 2007.

N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997.

R. Montemanni and D. H. Smith. A tabu search algorithm with a dynamic tabu list for the frequency assignment problem. Technical Report UG-01-01, University of Glamorgan, 2001.

C. Morgenstern and H Shapiro. Chromatic number approximation using simulated annealing. Unpublished, 1986.

P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, Report 826*, 1989.

T. Müller. ITC2007 solver description: a hybrid approach. *Annals of Operations Research*, 172 (1):429–446, October 2009.

T. Müller, H. Rudová, and R. Barták. Minimal perturbation problem in course timetabling. In *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 126–146. 2005.

K. Murray, T. Müller, and H. Rudová. Modeling and solution of a complex university course timetabling problem. In *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 189–209. 2007.

C. Nothegger, A. Mayer, A. Chwatal, and G. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research*, 194:325–339, 2012.

E. Özcan and E. Ersoy. Final exam scheduler - FES. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1356 – 1363 Vol. 2, 2005.

B. Paechter, R. Rankin, A. Cumming, and T. Fogarty. Timetabling the classes of an entire university with an evolutionary algorithm. In *Parallel Problem Solving from Nature PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 865–874. 1998.

L. Paquete and C. Fonseca. A study of examination timetabling with multiobjective evolutionary algorithms. *Proceedings of the 4th Metaheuristics International Conference*, pages 149–154, 2001.

J. E. L. Peck and M. R. Williams. Algorithm 286: Examination scheduling. *Commun. ACM*, 9(6):433–434, 1966.

S. Petrovic and E. K. Burke. *University Timetabling*, chapter 45. CRC Press, 2004.

N. Pillay and W. Banzhaf. An informed genetic algorithm for the examination timetabling problem. *Applied Soft Computing*, 10(2):457 – 467, 2010.

D. Porumbel, J. K. Hao, and P. Kuntz. Diversity control and multi-parent recombination for evolutionary graph coloring algorithms. In *Evolutionary Computation in Combinatorial Optimization*, volume 5482 of *Lecture Notes in Computer Science*, pages 121–132. 2009.

G. Post, L. Gaspero, J. H. Kingston, B. McCollum, and A. Schaerf. The third international timetabling competition. *Annals of Operations Research*, pages 1–7, 2013.

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

T. Ray, T. Kang, and S. K. Chye. An evolutionary algorithm for constrained optimization. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 771–777, 2000.

P. Ross, E. Hart, and D. Corne. Some observations about GA-based exam timetabling. In *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pages 115–129. 1998.

O. Rossi-Doria and B. Paechter. A memetic algorithm for university course timetabling. *Proceedings of Combinatorial Optimization*, page 56, 2004.

H. Rudová and K. Murray. University course timetabling with soft constraints. In *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 310–328. 2003.

H. Rudová, T. Müller, and K. Murray. Complex university course timetabling. *Journal of Scheduling*, 14(2):187–207, 2011.

N. Sabar, M. Ayob, G. Kendall, and R. Qu. Roulette wheel graph colouring for solving examination timetabling problems. In *Combinatorial Optimization and Applications*, volume 5573 of *Lecture Notes in Computer Science*, pages 463–470. 2009.

D. Safaai, O. Sigeru, O. Hiroshi, and S. Puteh. Incorporating constraint propagation in genetic algorithm for university timetable planning. *Engineering Applications of Artificial Intelligence*, 12(3):241 – 253, 1999.

A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13:87–127, 1999.

K. Socha, J. Knowles, and M. Sampels. A max-min ant system for the university course timetabling problem. In *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 63–77. 2002.

K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 334–345. 2003.

H.Y. Tarawneh, M. Ayob, and Z. Ahmad. A hybrid simulated annealing with solutions memory for curriculum-based course timetabling problem. *Journal of Applied Sciences*, 13(2):262–269, 2013.

J. M. Thompson and K. A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25(78):637 – 648, 1998.

H. Turabieh and S. Abdullah. An integrated hybrid approach to the examination timetabling problem. *Omega*, 39(6):598 – 607, 2011.

E. Vizuete Luciano, J. M. Merigó, A. M. Gil-Lafuente, and S. Boria Reverté. OWA operators in the assignment process: The case of the hungarian algorithm. In *Modeling and Simulation in Engineering, Economics and Management*, volume 115 of *Lecture Notes in Business Information Processing*, pages 166–177. 2012.

D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

Y. Xu and R. Qu. An iterative local search approach based on fitness landscapes analysis for the delay-constrained multicast routing problem. *Computer Communications*, 35(3):352 – 365, 2012.

S. Yang and S. N. Jat. Genetic algorithms with guided and local search strategies for university course timetabling. *IEEE TSMC*, 41(1):93 –106, 2011.

N. Zufferey, P. Amstutz, and P. Giaccari. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, 11:263–277, 2008.