Optimisation Algorithms for Planning and Scheduling Workplace Training

by Olivér Gábor Czibula

Supervisors: Hanyu Gu, Feng-Jang Hwang, Mikhail Y. Kovalyov

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

University of Technology, Sydney – 2017

CERTIFICATE OF ORIGINAL AUTHORSHIP

This research is supported by an Australian Government Research Training Program Scholarship.

Signature of Student:

Date:

Abstract

This thesis is concerned with a number of related mathematical optimisation problems in, or closely related to, the fields of scheduling, timetabling, and rostering. The studies are motivated by real world problems routinely faced at an Australian electricity distributor. Due to the computational complexity of the problems considered and typical real-world problem sizes, solution by direct application of mathematical programming is not possible in practically acceptable time. Therefore, we propose a variety of heuristic, metaheuristic, and matheuristic approaches to obtain good quality solutions in acceptable time.

The first study is concerned with a large-scale class timetabling and trainer rostering problem. The problem is formulated as two Integer Programs: one for class timetabling and one for trainer rostering. A three-stage approach is presented, consisting of a timetable construction stage, a timetable improvement stage, and a trainer rostering stage.

The second study investigates a variation of the timetabling problem considered in the first study, but from an analytical perspective. The problem is presented in the context of batch scheduling. Conditions that lead to NP-hardness are shown, and a previously-known NP-hardness result is strengthened. A polynomial time algorithm is presented for a particular case. Simulated Annealing and Genetic Algorithm based metaheuristics are compared by means of extensive computational experimentation.

The third study is of a partitioning problem concerned with optimising the composition of study groups, which is related to, but distinct from, several well-known problems in the literature. The problem is shown to be NP-hard in the strong sense. Four approaches are proposed: the first is based on Lagrangian Relaxation, the second is based on Column Generation, the third encapsulates Column Generation within a fix-and-optimise Large Neighbourhood Search framework, and the fourth is a Genetic Algorithm amalgamated with Integer Programming.

The fourth study is concerned with the timetabling of practice placements. The problem is shown to be NP-hard in the strong sense. Two approaches are presented: the first approach improves an initial timetable by means of a Simulated Annealing metaheuristic, and the second approach constructs and improves a timetable by means of a fix-and-optimise Large Neighbourhood Search procedure.

The aim of this research is to study these related optimisation problems encountered at the electricity distributor from both analytical and practical perspectives, and to design successful solution approaches to them.

The research presented in this thesis has been published in several refereed publications [39, 37, 40, 41, 42, 38, 43].

Acknowledgements

I thank my wife, Anastasia, whose constant support and patience throughout my research made this work possible. I also extend my profound gratitude to my supervisors: Hanyu Gu, Feng-Jang Hwang, and Mikhail Y. Kovalyov, and to my advisor Yakov Zinder. All their encouragement, advice, experience, attention to detail, and expertise helped keep my work to a high standard. Finally, I thank Aaron J. Russell and Tom Emeleus for giving me this opportunity to work on these fascinating real-world problems.

To my son, Maxwell.

Contents

List of Figures

List of Tables

List of Refereed Publications

[39] O. Czibula, H. Gu, Y. Zinder, and A. Russell. A multi-stage IP-based heuristic for class timetabling and trainer rostering. In *International Conference of the Practice and Theory of Automated Timetabling*, 2014

[37] O. Czibula, H. Gu, A. Russell, and Y. Zinder. A multi-stage IP-based heuristic for class timetabling and trainer rostering. *Annals of Operations Research*, pages 1–29, 2014

[40] O. G. Czibula, H. Gu, F.-J. Hwang, M. Y. Kovalyov, and Y. Zinder. Bi-criteria sequencing of courses and formation of classes for a bottleneck classroom. *Computers & Operations Research*, 65:53–63, 2016

[41] O. G. Czibula, H. Gu, and Y. Zinder. A Lagrangian relaxation-based heuristic to solve large extended graph partitioning problems. In *WALCOM: Algorithms and Computation*, pages 327–338. Springer, 2016

[42] O. G. Czibula, H. Gu, and Y. Zinder. Scheduling personnel retraining: Column generation heuristics. In *International Symposium on Combinatorial Optimization*, pages 213–224. Springer, 2016

[38] O. Czibula, H. Gu, and Y. Zinder. Timetabling of workplace training: A combination of mathematical programming and simulated annealing. In *International Conference of the Practice and Theory of Automated Timetabling*, 2016

[43] O. G. Czibula, H. Gu, and Y. Zinder. Lagrangian relaxation versus genetic algorithm based matheuristic for a large partitioning problem. *Theoretical Computer Science*, 2017

Chapter 1

Introduction

This thesis is concerned with a number of related mathematical optimisation problems in, or closely linked to, the fields of scheduling, timetabling, and rostering. The studied problems are all real-world, and routinely faced in industry. Each of the problems discussed in this thesis originate at an Australian electricity distributor. Care was always taken to present each of the problems, background, solution approaches, results, and conclusions, in such a way that they are applicable to many other scenarios encountered in industry, or otherwise such that they are of academic interest.

The aim of the research was twofold. The primary goal of the research was to investigate the specific problems routinely faced at the electricity distributor, and to develop efficient solution approaches to them. The secondary goal was to analyse the investigated problems from an analytical viewpoint, in order to derive computational complexity results, identify particular cases that are especially easy or especially hard, and to develop algorithms that solve the problems, or special cases thereof, efficiently. In all cases, in order to facilitate investigation by analytical means, simplification was made to the underlying problem. Sometimes a great deal of simplification was made. In practice this is not unusual, however the consequence is that any algorithms developed for these simplified problems and the corresponding results may not be directly applicable to the underlying problems. Nevertheless, these results and algorithms do give us a feel for some of the behaviour of these underlying problems, and can often guide us towards developing efficient solution approaches to them.

Below is a short outline of each of the problems considered in this thesis along with a brief overview of the main results presented. This chapter concludes with a summary of the contributions of this thesis.

1.1 Demand-Based Course Timetabling

The first of the studied problems, discussed in Chapter 3, is an in-depth investigation into the optimisation of a large-scale periodic training and re-training problem at an Australian electricity distributor. This problem shares many common features with academic timetabling problems, such as high school and university course and examination timetabling, however our problem contains a number of features—discussed in the Chapters 2 and 3—that distinguish it from the timetabling problems most commonly studied in the literature. These distinguishing features also make existing solution approaches not directly applicable to our problem, and therefore represent a gap in knowledge that Chapter 3 aims to address.

Due to the extreme hazards involved when working with high voltages, at heights or underground, in confined spaces, or in the presence of highly toxic or carcinogenic substances, the Australian government requires—via the Work Health and Safety Act 2011: Part 2, Division 19; and enforced by the Energy Networks Association (ENA)—that all such workers must successfully complete mandatory safety and technical training prior to undertaking such work. This training has a limited period of validity and must be retrained, or 'refreshed', otherwise it is considered no longer valid. Most courses have validity periods of 12 months, but a small few last 3 or even 5 years.

Many large electricity distributors find it practical to deliver some or all of this training themselves, and those that do will often deliver to the hundreds or perhaps thousands of field workers they employ. Many such distributors often also offer safety and technical training to the employees and contractors of other distributors that do not provide their own training, as well as to any other people who work on or near the electricity network.

Due to the highly periodic nature of the training volume, which is induced by the strict qualification validity periods, the distributors are able to produce a timetable several months in advance, deciding how many times each course should be run, when and where, and taught by whom. This problem can be formulated as a large-scale IP model, however, due to the size of the problem, it is not possible to solve by direct application of mathematical programming in practically acceptable time, even for very small problem instances.

Chapter 3 presents the problem in the form of two separated components: class timetabling, and trainer rostering. IP models for each problem are presented. The proposed solution approach is a three-stage heuristic consisting of: ***(i)*** sequential construction of the initial timetable, ***(ii)*** timetable improvement by means of Large Neighbourhood Search, and ***(iii)*** the rostering of trainers.

In the first stage, an initial feasible timetable is produced by means of a constructive heuristic. At each iteration of the constructive heuristic, the proposed IP model is solved for a small set of classes, and the resulting solution is appended to a partial solution that becomes increasingly complete at each iteration with respect to the underlying problem. The second stage attempts to improve the timetable using a fix-and-optimise Large Neighbourhood Search heuristic. At each iteration of the second stage, the proposed IP model is solved for only a small selected subset of the variables, with the remaining variables substituted by constants that correspond to their values given by the incumbent solution. In the third and final stage, specific trainers are assigned to the timetable using direct application of Integer Programming.

While this specific timetabling problem has not received much research attention thus far, the problem formulation and solution approaches are likely to be relevant to any organisation that delivers training based on forecast demand, i.e. where numbers of students are approximately known, and a training timetable must be created to accommodate this future demand. These organisations may be those that deliver training to their own employees, such as those operating in dangerous industries (e.g. utilities, natural resource extraction, construction, demolition), those operating in technical industries (e.g. telecommunications, engineering, manufacturing), or those in general that deliver professional development training (e.g. acceptable conduct, communication, legal compliance) to their employees.

The application of this automated timetabling system at the electricity distributor was reported to save the organisation two senior trainer full time equivalent (FTE) salaries each year. At the time of implementation, this corresponded to AUD$270,000 savings in total per year compared with the manual approach used previously.

The research in Chapter 3 was presented at the 10th International Conference on the Practice and Theory of Automated Timetabling (PATAT) 2014 [125, 39] and included in the refereed conference proceeding as "O. Czibula, H. Gu, Y. Zinder, and A. Russell. A multi-stage IP-based heuristic for class timetabling and trainer rostering. In *International Conference of the Practice and Theory of Automated Timetabling*, 2014", and was subsequently extended and published in the PATAT 2014 Special Issue by the Annals of Operations Research [37], which was cited by El-Sakka [53].

1.2 Formation and Sequencing of Classes for a Bottleneck Classroom

The second of the studied problems, discussed in Chapter 4, considers a variation of the timetabling problem discussed in the previous chapter, but from a different perspective. Here, we instead consider a single bottleneck classroom with a limited capacity, together with the allocation of specific students to classes. On the one hand, this problem is a simplification of the problem discussed in Chapter 3 due to the single bottleneck classroom. On the other hand, this problem considers the assignment of specific students to classes, whereas individual student assignments were not considered in the previous chapter. This problem is presented as a batch scheduling problem with incompatible job families [16].

While primarily an analytical study, this research has practical application also. While individual student assignments are not present in the research in Chapter 3, this is primarily necessitated due to difficulty in scheduling people in many different corporate divisions across a large geographical area amid their existing work schedules. Overcoming this obstacle would allow for significantly greater control in class timetabling, which would likely result in greater training efficiency and further cost savings. Therefore, it is reasonable to say that an organisation of this type would aim towards such a goal. Moreover, due to the distribution of rooms in each training location, it is not rarely the case that only one room is available for a particular course in a given location.

The considered problem has an ordered bicriteria objective function, i.e. a primary objective and a secondary objective; several primary and secondary objectives are considered. A feasible timetable is one where all students can be assigned to their one or more required courses, with the number of students in each class not exceeding some upper bound, which is a property of each course. The objective is to determine an optimal formation and sequence of classes for courses and students so as to minimise the considered performance metrics. An optimal schedule is one that minimises the secondary objective over the set of schedules that minimise the primary objective.

In the context of batch scheduling, we represent the bottleneck classroom by a batching machine, and each of the students' course requirements by a job, with a due date being the date at which their corresponding training expires. The batching machine has a maximum capacity (number of simultaneous jobs being processed) equal to the maximum capacity of the room, in students. Each class for a given course is a batch of jobs, where all the jobs correspond to students undertaking that course. This single machine batch scheduling with an ordered bicriteria objective and incompatible job families has not been discussed in the literature thus far, and the material in Chapter 4 aims to address this gap in knowledge.

We present a proof of NP-hardness in the strong sense for the objective of minimising the number of tardy jobs by reduction in pseudopolynomial time from the single machine total weighted tardiness problem. We show that for the objective of minimising a maximal regular function of job completion times, which is NP-hard, students can be optimally assigned to classes in polynomial time so long as the sequence of courses remains fixed. We present a polynomial time algorithm that is based on Lawler's algorithm [97] for the objective of minimising total cost over any regular function of job completion time.

The bicriteria problem with a maximal primary objective and total secondary objective is formulated by means of Integer Programming. Direct application of mathematical programming is compared to two proposed metaheuristic solution approaches—one based on Simulated Annealing and the other on

Genetic Algorithms—by means of extensive computational experimentation.

The research in Chapter 4 was presented at the Australian Society for Operations Research (ASOR) Recent Advances 2015 conference [150], where it was awarded the "Best Student Presentation" prize and AU$250. The research was subsequently published as "O. G. Czibula, H. Gu, F.-J. Hwang, M. Y. Kovalyov, and Y. Zinder. Bi-criteria sequencing of courses and formation of classes for a bottleneck classroom. *Computers & Operations Research*, 65:53–63, 2016" [40].

1.3 Partitioning of Students into Classes

The third of the studied problems, discussed in Chapter 5, is of how best to assign a set students of different types to an existing timetable of classes, such that the total incompatibility between all pairs of students in each given class, and simultaneously that the total penalty of assigning particular students to particular classes, is minimised.

In this context, since different students are employed in different working roles, have different learning outcomes, have different educational and professional backgrounds, have different skills and proficiency with regard to certain tasks, etc., it follows that the composition of the study groups can influence the effectiveness of the delivered training. By assigning similar types of students together into the same class, the trainer can more effectively tailor the delivery of the core material to the specific needs of the group. However, it is not simply a case of grouping students into classes such that the students within each class are as similar to one another as possible. Other considerations exist, such as the desire for different types of students, but those who must interact with one another professionally, to occasionally undertake training together in order to facilitate the flow of knowledge, experience, and understanding between the different groups. Other considerations of the problem must also be acknowledged, such the permissible ranges of class sizes as well as the costs associated with delivering each training class.

Using more general terminology, we consider the problem of assigning each of a set of objects, each of which belongs to one of a set of types, to exactly one of a set of non-identical bins. Each bin has a permissible cardinality of both contained types and contained objects. The objective is to minimise the cost of pairing differing types within a single bin, and the cost of assigning particular objects to particular bins. This problem closely related to, but distinct from, several classical, well-studied problems in combinatorial optimisation, such as the Graph Partitioning Problem (GPP) and the related Edge Partitioning Problem (EPP), the Quadratic Multiple Knapsack Problem (QMKP), and the Quadratic Assignment Problem (QAP). This particular problem does not appear to have been studied in the past, and Chapter 5 aims to address this gap in knowledge.

The problem is formulated as a linearly constrained Quadratic Program, and a linearisation is presented. Four solution approaches to the problem are presented. The first approach is a heuristic based on the Lagrangian Relaxation of the Quadratic Program, and involves identifying trends in the many intermediate solutions of the subgradient algorithm to significantly reduce the decision space of the problem by imposing additional restrictions. The second approach is based on Column Generation, and three strategies are proposed to produce integer feasible solutions. The third approach incorporates Column Generation within a fix-and-optimise Large Neighbourhood Search framework. The fourth approach is an amalgamation of Genetic Algorithms and Integer Programming.

The research in Chapter 5 is an amalgamation of what was presented at the 10th International Workshop on Algorithms and Computation (WALCOM) 2016 [157, 41] and included in the refereed conference proceeding as "O. G. Czibula, H. Gu, and Y. Zinder. A Lagrangian relaxation-based heuristic to solve large extended graph partitioning problems. In *WALCOM: Algorithms and Computation*, pages 327–338. Springer, 2016", and what was presented at the 4th International Symposium on Combinatorial Optimization (ISCO) 2016 [80, 42] and included in the refereed conference proceeding as "O. G. Czibula, H. Gu, and Y. Zinder. Scheduling personnel retraining: Column generation heuristics. In *International Symposium on Combinatorial Optimization*, pages 213–224. Springer, 2016". The research was extended and submitted to the WALCOM 2016 Special Issue by Theoretical Computer Science, ans was accepted for publication [43], and submitted to the ISCO 2016 Special Issue by the Journal of Combinatorial Optimisation.

1.4 Timetabling of Practice Placements

The last of the studied problems, discussed in Chapter 6, is concerned with the creation of a practice placement timetable for apprentices. This problem—which is not well studied from the perspective of optimisation—is very closely related to the problem of timetabling practice placements in clinical education, for example nursing students. Since this wide-spread optimisation problem has received very little research attention thus far, Chapter 6 aims at addressing this gap in knowledge.

Following their theoretical studies, apprentices at the electricity distributor must successfully complete a number of practice placements, also known as rotations, during which they are exposed to the skills they will need in their profession. Each apprentice must complete exactly one placement from each of a set of placement groups, where the set of placement groups is given for the type of apprenticeship. Each placement within a placement group is of the same type, however they are different insofar as they may be in different locations or may otherwise have different suitability for each apprentice. The assignment of a particular apprentice to a particular placement therefore carries a cost, or penalty, that is accumulated per unit of time the apprentice spends at the placement. Each placement requires the apprentice to spend a minimum amount of time there. Each placement can have either zero apprentices, or between a lower and upper bound of apprentices that is given for the placement. Each apprentice can do at most one placement at a time, and apprentices cannot stop and then resume placements at a later time. The objective is to produce a placement timetable where all apprentices are assigned to exactly one of the placements in each of their required groups, and the total weighted penalty is minimised.

The problem is shown to be NP-hard by reduction from the partition problem. Two solution approaches are presented for the problem. The first approach initially constructs a timetable by means of solving a proposed IP model one apprentice at a time, and then attempts to improve the timetable using a Simulated Annealing metaheuristic. The second approach uses a fix-and-optimise Large Neighbourhood Search procedure to produce a timetable.

The problem of timetabling practice placements for students is encountered in many places around the world in many different disciplines, but perhaps most notably in clinical education. In 2012 alone, around 300 nursing students in Sydney were unable to graduate on time as they had not completed all their required practice placements. This was because a timetable where all nursing students would complete their practice placements in the allotted time could not be constructed. A successful solution approach to this timetabling problem greatly reduces the chance of this issue, and can also automate much of the routine and manual task of practice placement timetabling.

The research in Chapter 6 was presented at the 11th International Conference on the Practice and Theory of Automated Timetabling (PATAT) 2016 [126, 38] and included in the refereed conference proceeding as "O. Czibula, H. Gu, and Y. Zinder. Timetabling of workplace training: A combination of mathematical programming and simulated annealing. In *International Conference of the Practice and Theory of Automated Timetabling*, 2016", and was subsequently extended and submitted to the PATAT 2016 Special Issue by the Annals of Operations Research.

1.5 Contributions

A summary of the contributions of the work presented in this thesis is as follows:

- for the course timetabling problem, discussed in Chapter 3:
  - a novel, industry-inspired timetabling problem, similar to, but distinguished from, traditional academic timetabling problems, is presented,
  - a large-scale Integer Programming formulation of the considered problem is presented as:
    1. an Integer Programming model to create a class timetable, subject to the unique operational constraints imposed by the industry, and

    2. an Integer Programming model, presented as several network flow models together with non-network coupling constraints, to create a trainer roster for an existing class timetable, subject to the unique operational constraints imposed by the industry,
  - a three stage solution framework is proposed, consisting of:
    1. an initial timetable construction stage, employing an Integer Programming-based constructive heuristic,
    2. a timetable improvement stage, employing a Integer Programming-based Large Neighbourhood Search fix-and-optimise heuristic,
    3. a trainer rostering stage, employing direct application of Integer Programming,
  - the proposed heuristic framework is shown, by means of extensive computational experimentation, to produce good quality solutions in practically acceptable time, whereas direct application of mathematical programming could not be shown to perform successfully,

- for the bi-criteria single-machine batch scheduling problem with incompatible job families, discussed in Chapter 4:
  - a novel, industry-inspired problem was presented as a single-machine batch scheduling problem with incompatible job families and an ordered bi-criteria objective,
  - for the objective of minimising the number of tardy jobs, NP-hardness is proven by pseudopolynomial reduction from the single machine total weighted tardiness problem,
  - for the objective of minimising maximum cost, an $O(n^2)$ algorithm is presented, where n is the number of jobs, for both the single- and bicriteria cases,
  - an Integer Programming model for the considered model is presented,
  - for the case with a maximal primary objective and a total secondary objective, a Simulated Annealing approach incorporating a Tabu list is presented, requiring $O(n)$ operations to establish feasibility,
  - to compare with the Simulated Annealing approach, a Genetic Algorithm is also presented for the considered problem,
  - it is shown, by means of extensive computational experimentation, that both of the proposed metaheuristics outperformed direct application of Integer Programming,

- for the partitioning problem, discussed in Chapter 5:
  - a novel, industry-inspired problem was presented, which is distinct from but closely related to the graph partitioning problem, the edge partitioning problem, the quadratic multiple knapsack problem, and the quadratic assignment problem,

  - the NP-hardness of the problem is proved by reduction from the clique problem,
  - a novel Quadratic Programming formulation, as well as its linearisation, is presented for the problem,
  - four solution approaches are presented:
    1. an approach based on solving the Lagrangian Dual problem by means of the subgradient algorithm, and then identifying trends in the intermediate solutions and imposing restrictions that significantly reduce the decision space of the underlying problem,
    2. an approach based on Column Generation, where three strategies utilising the generated columns are proposed to produce integer feasible solutions to the primal problem,
    3. a fix-and-optimise Large Neighbourhood Search approach incorporating the Column Generation procedure,
    4. an amalgamation of Genetic Algorithms and Integer Programming,
  - it is shown, by means of extensive computational experimentation, that the fix-and-optimise LNS approach performed the best, followed by the Column Generation-based approaches, followed by the Lagrangian Relaxation-based approach, followed by the Genetic Algorithm-based matheuristic. Each of the proposed heuristics are shown to outperform direct application of mathematical programming,
- for the practice placement timetabling problem, discussed in Chapter 6:
  - a industry-inspired timetabling problem was presented that does not appear to have been previously studied from the perspective of optimisation,
  - the NP-hardness is proved, even in the case of only a single time interval, by reduction from the partition problem,
  - a novel Integer Programming formulation of the problem is presented,
  - two solution approaches are presented:
    1. an approach that constructs an initial timetable one student at a time, and then improves the initial timetable using Simulated Annealing, and
    2. an approach that employs a fix-and-optimise Large Neighbourhood Search procedure to produce a timetable,
  - it is shown, by means of extensive computational experimentation, that the proposed heuristics outperform direct application of Integer Programming.

Chapter 2

Literature Review

Contents

This chapter discusses both the historical and the state-of-the-art research that is most relevant to this thesis. First, the concept of timetabling from the perspective of combinatorial optimisation is briefly introduced. Next, timetabling is discussed in general, mainly from the perspective of complexity analysis. Finally, the literature relevant to the research in Chapters 3, 4, 5, and 6 is discussed separately and in order.

A number of books on combinatorial optimisation, decomposition, and complexity analysis ([64], [117], [160]), and metaheuristics ([51], [68], [147], [123]) are recommended in order to gain an understanding of material presented in this thesis.

2.1 A Very Brief Introduction to Timetabling

A timetable can be described as the allocation of a set of meetings to a set of times, where a meeting is a combination of resources, such as people, rooms, and equipment. Timetabling is a combinatorial optimisation problem concerned with allocating resources in space and time, subject to the specific problem's constraints, in such a way that a given set of objectives are optimised. Automated timetabling was first mentioned in the scientific literature by Gotlieb [71] in the 1960's, and since then the field of automated timetabling has attracted a great deal of research attention.

There are several variations on the timetabling problem in general, and it has a wide range of applications, such as the timetabling of buses, trains, and aeroplanes, home deliveries, work shifts, in project management, etc. A large proportion of practical timetabling research in the literature is concerned with high school and university course and examination timetabling. This may be, in part, due to researchers' increased exposure to these optimisation problems, and perhaps also due to the relative uniformity between timetabling problems across educational institutions throughout the world, making it more amenable to research, collaboration, and competition. Indeed, the International Timetabling Competition, focussing on high school and university timetabling, attracts researchers from all over the world who compete against one another for prizes, prestige, and to advance knowledge in the field [131]. High school and university timetabling problems are special cases of educational timetabling problems, which are special cases of timetabling problems in general, which are special cases of scheduling problems in general [6].

In many cases, authors choose to describe timetabling problems in terms of "hard" and "soft" constraints. The so-called hard constraints are those that must not be violated for a solution to be considered feasible, whereas the so-called soft constraints are those whose violations are considered undesirable, and should be minimised wherever possible. For example, hard constraints may include those forbidding trainers or students from being scheduled to different activities at the same time, while soft constraints may include those that require each lecturer to be allocated a total number of teaching hours between a nominated upper and lower bound. In some cases, like where feasibility is particularly difficult to achieve, some constraints that may otherwise be considered hard, such as the requirement that students should not be required to attend two lectures simultaneously, are instead classified as soft [138].

2.2 Complexity of Timetabling

Carter and Tovey [27] discuss the computational complexity of a small subproblem contained within perhaps all university and high school timetabling problems: the Classroom Assignment Problem (CAP).

Define $X_{i,j}$ to be 1 if class $i \in [1, n]$ is assigned to room $j \in [1, m]$, or 0 otherwise. Time is partitioned into a number of discrete time periods, and classes are already understood to run at a particular time. The constraints of the problem can be formulated as follows:

$$\sum_{j=1}^{m} X_{i,j} = 1 \quad i = 1, \ldots, n \tag{2.1}$$

$$\sum_{i \in P_k} X_{i,j} \leq 1 \quad j = 1, \ldots, m; \forall k \tag{2.2}$$

where P_k is the set of classes that run during period k. Constraints (2.1) ensure that each class is assigned to exactly one room, and constraints (2.2) ensure that classes running at simultaneous time periods cannot be assigned to the same room.

The authors define three objectives for the above-mentioned problem in increasing order of difficulty: *Feasibility*, which asks the question of whether a feasible solution exists at all; *Satisfice*, which asks the question whether a feasible solution exists in which each class i can only be assigned to a subset of rooms $S_i \subseteq [1, m]$ deemed "satisfactory"; and *Optimize*, where the objective is to minimise $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} X_{i,j}$ for the cost $c_{i,j}$ of assigning class i to room j. The authors define the case where several related but non-overlapping classes must be run in the same room as the noninterval classroom assignment problem, and the case with no such restriction as the interval classroom assignment problem.

The authors provide a proof of NP-completeness, by reduction from the graph K-colourability problem, for the *Feasibility* objective of the noninterval case, while the *Feasibility* problem for the interval case is solvable in $O(n)$ time. Also provided is a proof of NP-completeness, by reduction from 3-SAT, of the *Satisfice* objective of the interval case, even when the problem is restricted to just two periods. The results in [27] show that optimisation of essentially all high school and university course and examination timetabling, including the class timetabling problem we consider in Chapter 3, is NP-hard.

Cooper and Kingston [34] discuss five sources of NP-completeness for the timetable construction problem frequently encountered in practice:

- Intractability owing to giving students choice of subjects from a wider pool is shown by reduction from graph K-colourability. The authors note that this problem can be avoided by publishing the timetable in advance, and requiring students to choose their subjects such that they have no clashes.

- Intractability owing to varied class sizes resulting in difficulty in assigning the target number of meetings to each teacher—even when disregarding restrictions imposed by teacher qualifications and the requirement that there be no clashes—is shown by reduction from the bin packing problem. The authors note that some high schools permit split assignment, where meetings can be split into two, allowing two teachers to share the meeting, to create small fragments to fill the bins. Universities are not subject to this problem, as workloads are generally lighter and more flexible.

- Intractability owing to time-incoherence, where as a result of varying meeting sizes some meetings must share at least one time period, is shown by reduction from the exact cover by 3-sets problem. The authors demonstrate that the timetable construction problem is NP-complete as a result of time incoherence, even in the absence of the above-mentioned bin packing substructures. The authors also show that enforcement of time-coherence can be shown to be NP-complete by reduction from the bin packing problem.

- Intractability owing to restrictions on class times—such as double classes or classes spread throughout the week—is shown by reduction from the exact cover by 3-sets problem. The authors note that the special case where the exact cover by 3-sets problem can be solved in polynomial time if no element occurs in more than two subsets [64], and also the special case where times are to be assigned to any number of time-disjoint meetings, can be solved in polynomial time by means of a bipartite matching.

- Intractability owing to the problem of assigning multiple forms simultaneously—a form being where all meetings having a nominated student group in common—is shown by reduction from the three-dimensional matching problem.

In general, the multiple unique sources of intractability inherent to the timetable construction problem imply that it is unlikely that one can construct an efficiently solvable subproblem from which meaningful information about the greater problem can be recovered.

Welsh [158] introduces the relationship between the NP-hard graph colouring problem and the timetabling problem, discussed further by Lewis [101]. A simple timetabling problem can be converted to a graph colouring problem, and vice-versa, by representing events as vertices, and connecting together with edges those vertices that correspond to events that cannot be scheduled at the same time [90]. Each timeslot can be represented by a colour, and a feasible timetable is one where the corresponding graph colouring uses no more unique colours than available time slots. The author discusses that although real-world timetabling problems often contain constraints that make graph colouring insufficient as a modelling tool on its own, some timetabling heuristics extract information derived from this subproblem to aid in the search for solutions to the underlying problem. Marx [107] discusses the relationship between graph colouring problems and their extensions to several scheduling problems, such as aircraft scheduling and the scheduling of biprocessor tasks.

2.3 Demand-Based Course Timetabling

This section gives an overview of literature in the field of academic timetabling, as it best relates to the research presented in Chapter 3. Comparisons with our considered class timetabling problem are made, and highlights of important similarities and differences are highlighted.

Academic timetabling has been a very active and growing research area [127]. Early solution approaches to these problems attempted to imitate a person's decision processes, and were known as 'Direct Heuristics' [140]. These worked on the principle of successive augmentation, whereby the 'most constrained' courses were usually scheduled first, and variations to these approaches typically explored different ways of defining 'most constrained'. The approaches that followed were more general, such as those based on mathematical programming, network flow, or by reducing the problem to the graph colouring problem [138].

Researchers in large-scale timetabling typically do not attempt to find optimal solutions to these problems as such solutions often cannot be found in practically acceptable time. Much of the recent research attention has instead been focused on heuristics, including metaheuristics [101, 22], and decomposition methods such as Lagrangian Relaxation [58], Column Generation [133, 124], and Row Generation [21]. There has been some research activity in using graph colouring heuristics to solve timetabling problems that can be expressed as graphs [119, 109, 23, 153].

In an example of Row Generation, Burke et al. [21] propose a mixed integer model with fewer variables than a classical approach, but with an exponential number of constraints. The constraints are only applied if they are needed. The authors also introduce additional cuts that are not required for optimality, but do improve the computational performance of the procedure. In an example of applying graph colouring to timetabling, Burke et al. [23] solve an examination timetabling problem by means of a heuristic based on graph colouring to split exams into groups that can be scheduled together, combined with another algorithm to fit the exams into rooms.

A recent trend has been to develop 'hybrid heuristics' [22], which combine certain features of one heuristic with those of another, typically with the aim of improving computational performance by overcoming the heuristics' weaknesses. In Gunawan et al. [73], attempts were made to improve the convergence rate of Simulated Annealing (SA)—whose introduction is attributed to Kirkpatrick et al. [94] and Černỳ [28]—by implementing the memory property of Tabu Search (TS) to solve a university course timetabling problem. The annealing rate in SA can have a dramatic influence over the performance of a SA implementation [161]. It is not uncommon for people to experiment with a variety of simple and complex annealing schedules paired with reheating rules, typically with the aim of avoiding the heuristic becoming trapped in local minima [1]. A novel hybrid heuristic was presented in Bölte and Thonemann [15], where the authors propose a Genetic Program (GP) to optimise the annealing schedule for SA. They presented the dedicated cooling schedules found by GP that converge fastest for particular problems, and they also provided the cooling schedule found by GP that converges fastest across all problems they tested.

Despite the difficulty in solving large-scale timetabling models using straight-forward application of IP, several researchers have had some success using this approach. Perhaps the earliest examples are by Lawrie [98] from 1969 and by Akkoyunlu [2] from 1973. A more recent example of IP-based university course and examination timetabling is presented by Dimopoulou and Miliotis [48], where the authors augmented an IP with a heuristic improvement stage. A high school timetabling problem is presented in [14], where the authors were able to solve the IP directly. An ILP was presented by Schimmelpfeng and Helber [139], which had, amongst others, 99 teachers, 156 courses, and 181 teaching groups. The model had 35,611 rows, 91,767 integer variables, and 662,824 non-zeroes in the co-efficient matrix. An optimal solution was obtained in just 10 seconds using IBM ILOG CPLEX 9.1.2, or about 2 minutes with Coin-OR Branch-and-Cut.

The vast literature produced on university and high school class timetabling does not address our considered class timetabling problem in Chapter 3. The reasons are outlined below:

- Our considered class timetabling problem has different objectives to that of traditional university and high school class timetabling problems. Indeed, in our problem the focus is on satisfying demand, which fluctuates over the planning horizon, whereas in all publications of university and high school timetabling problems there is no fluctuation in demand, and the main focus is typically on the minimisation of teacher and student preference violations.

- Another objective of our class timetabling problem is to strengthen the robustness of the timetable over long periods of time by levelling the utilisation of resources, whereas in university and high school timetabling problems, the goal is typically to construct a timetable for one or two weeks that repeats throughout the semester or year.

- Our considered class timetabling problem is intended for a commercial organisation, where the ability to rent out available rooms is an important concern. The consideration of such opportunity for renting out rooms is not found in university and high school timetabling.

- University and high school timetabling problems require the allocation of a given set of classes to locations and times, whereas in our class timetabling problem, the number of classes and their sizes is part of the decisions that are to be made.

- In our class timetabling problem, the timetable should satisfy rigorous restrictions on how courses—which are composed of several modules—are scheduled. This is in contrast to university timetabling, where there are no comparable restrictions on the time and location for the assignment of different activities, such as lectures, computer laboratories, tutorials, etc.

- Our considered class timetabling problem requires decisions on the movement of scarce mobile resources between different locations, which is very unusual in university and high school timetabling.

The above-mentioned differences make the mathematical models commonly used in university and high school timetabling not applicable to our problem, which causes the necessity to develop new mathematical models together with new optimisation procedures that address the specific features and objectives of the problem on hand.

Not only new and interesting from a mathematical point of view, the considered class timetabling problem is also important from a practical perspective. Indeed, training and retraining of employees is an important and costly component of the business for many organisations. For example, according to a 2013 report from the Minerals Council of Australia, the vast majority of mining training in 2012 was conducted by the Australian mining industry itself, with over $1.1 billion spent. The report states that the training expenditure is almost 5.5% of total payroll, and that almost 98% of training expenditure is industry-funded with only 2% coming from government subsidies. [111]

> A recent study sponsored by the Minerals Council of Australia (MCA) shows the mining industry in Australia has made big investments in training workers for mining jobs.
>
> Incredibly, it shows the vast majority of mining training in 2012 has been conducted by the mining industry with over $1.1 billion spent compared to government training subsidies of $26 million training subsidies.
>
> The research consultancy report: Training and education activity in the minerals sector reveals contrary to popular opinion, the mining industry spends a huge amount on employee training.

The areas where research on timetabling for training and retraining can be of significant benefit also include other utility companies, forestry companies, oil and gas companies, construction companies, etc., to mention a few.

2.3.1 Rostering

Closely related to timetabling is rostering: the assignment of workers to shifts, satisfying constraints for example related to workers' skills, worker numbers, weekends and breaks, and so on, so as to optimise some performance criteria. Research in rostering has spanned several decades and has steadily been gaining research attention. The workforce rostering problem can be formulated and solved in many different ways [56, 155] including Dynamic Programming [12, 54], Integer Programming [8, 112], metaheuristics [35, 74, 55], and Constraint Programming [32, 120]. A commonly used model for scheduling and rostering comes from Dantzig's set covering formulation [44, 56].

For the rostering subproblem we consider in Chapter 3, we find the network model, based on a shift-time graph, proposed by Balakrishnan and Wong [7] to be effective. The authors discuss the rotating workforce scheduling problem, and propose a model based on network flow. All constraints of the problem are encoded within the network itself, apart from the staff covering constraints, which are enforced using non-network side constraints. Lagrangian Relaxation is used to find a lower bound on the staffing requirements, and a primal solution is obtained by solving the shortest path networks. The proposed approach is able to solve large-scale problems, including those with many complicating conditions on shifts, as is very common in the real world. The authors also note that while all operational constraints could be handled simultaneously by means of Integer Programming, the proposed network flow approach gives a deeper insight into the operation and structure of the problem, and facilitates the design of efficient computational procedures.

2.4 Formation and Sequencing of Classes for a Bottleneck Classroom

This section gives an overview of the literature as it pertains to the material presented in Chapter 4.

Batch scheduling has been studied for over three decades. Among the first to study the problem were Ikura and Gimple [79], who studied the single batch processing machine problem with batch sizes of $1 \dots k$, release times and due dates. They present efficient algorithms for minimising the makespan (commonly referred to as $C_{\max}$) under the assumption that, for a pair of jobs i and j, if the release time for job i is later than for job j, then the due date for job i is no earlier than for job j.

There has been an effort in studying the prototype of the considered batch scheduling problem, i.e. the single-machine batch scheduling problem with incompatible job families, where the batch size is constant rather than family-dependent. For this same prototype problem, but with a different application, Uzsoy [154] showed that the minimisation of the total weighted completion time, or the maximum lateness (commonly referred to as $L_{\max}$) can be solved in polynomial time. The paper presents—for the static case where all job are immediately available—polynomial time algorithms for the objective of minimising $C_{\max}$, $L_{\max}$, and the total weighted completion time. Then $\sum_{j\in J} w_j C_j$, for the case where job arrivals are dynamic, a polynomial time algorithm to minimise $C_{\max}$ and several heuristics to minimise $L_{\max}$ are proposed.

Later, Mehta et al. [110] proved that the problem for the minimisation of total tardiness is NP-hard in the strong sense by pseudopolynomial reduction from the single unit capacity machine total weighted tardiness problem. As for the minimisation of the number of tardy jobs, Jolai [85] showed that the problem is NP-hard by pseudopolynomial reduction from the single unit capacity machine total weighted number of tardy jobs problem. Jolai then presented a polynomial-time Dynamic Programming algorithm for the special case where the number of job families and machine capacity are fixed. Jolai [86] studied the problem of minimising earliness-tardiness criteria subject to a common unrestricted job due date, proving the problem is NP-hard.

Studying a more generalised problem where each job requires a different amount of machine capacity, Dobson and Nambimadom [49] considered the problem of minimising the total weighted completion time, presenting a polynomial-time algorithm for special cases of the problem, and three heuristics for the general case: a greedy heuristic, a successive knapsack heuristic, and a generalised assignment heuristic. Azizoglu and Webster [5] designed a branch-and-bound procedure for the same problem that can optimally solve cases up to about 25 jobs in acceptable time.

The problem of scheduling jobs with equal processing time on a single batching machine to minimise a primary and secondary criterion, as well as a linear weighted criterion, is studied by Lie et al. [105]. The authors present polynomial time algorithms for various combinations of the primary and secondary criteria. Janiak et al. [82], Cheng et al. [30] and Janiak et al. [83] developed polynomial time algorithms for *group technology* single machine scheduling problems, including problems with ordered criteria, in which the jobs are processed sequentially in a batch, a single batch is made for each family and each batch is preceded by a set-up time.

2.5 Partitioning of Students into Classes

This section gives an overview of the literature as it pertains to the material presented in Chapter 5.

The considered combinatorial optimisation problem is distinct from, but shares certain similarity with, the Graph Partitioning Problem (GPP) [17], also known as the Min-Cut Clustering Problem

(MCC) [84]. Given an undirected graph $G(V, E)$ with weights $w_v, v \in V$, edge costs $c_e, e \in E$, the GPP, which is the problem to find a partition $\Gamma = \{W_1, \ldots, W_K\}$ of V, can be formulated as follows:

$$\text{min:} \quad \sum_{i=1}^{K} \sum_{e \in E(W_i)} c_e \tag{2.3}$$

$$\text{s.t.} \quad w_{\min} \leq \sum_{u \in W_k} w_i \leq w_{\max} \quad k = 1, \ldots, K \tag{2.4}$$

where $E(W_i) = \{(i, j) \in E : i, j \in W_i\}$. This problem is known to be NP-hard in general [64].

The problem studied in Chapter 5 is distinct from the GPP in several ways. In the language of the GPP, each cluster in our problem has a lower- and upper-bound not only on the included node weight (where we assume each node has unit weight), but also a lower- and upper-bound on the on the node *type*, where the node type is analogous to the student type in our problem. Moreover, our problem permits certain clusters to remain empty. Our problem also penalises particular node pairs for being assigned to particular clusters.

The GPP arises in various practical situations, for example: partitioning subroutines that are compiled into computer code into clusters, such that the communication between clusters is minimised [84]; assigning data or processes evenly to processors, such that inter-process communication is minimised in parallel computing environments [3]; in aircraft control, where the flow of aircraft within blocks is maximised and the flow of aircraft between blocks is minimised [13]; partitioning the electronic subcircuits in very large-scale integration (VLSI) systems, such that the electrical connections between partitions is minimised [89]; partitioning a power network into so-called islands to prevent the propagation of cascading failures [102, 103]; for route planning along road networks [95, 106]; and in many other domains. Although the combinatorial optimisation problem considered in Chapter 5 and the GPP are different, each provides an insight in the design of solution techniques for the other.

Kernighan and Lin [93] studied the GPP, and presented a heuristic that runs in $O(n^3)$ for sparse graphs. It is known as the Kernighan-Lin algorithm, and is among the best-known today for this problem. Carlson and Nemhauser [25], motivated by the problem of scheduling classes at a university, studied the problem of partitioning the graph with no restriction on the size of each subset, however they do consider an interaction cost (penalty) corresponding to any two nodes grouped into the same subset. They present an efficient algorithm for finding feasible solutions that are also local minima, however the nonconvexity of the quadratic objective function prevents them from identifying a global minimum.

Several particular cases of the GPP, including Integer Programming (IP) models for each case, are discussed by Chopra and Rao [31]. The authors take advantage of the graph structure when clustering. They also discuss several valid inequalities and facet-defining inequalities for the GPP. A Column Generation approach is presented in Johnson et al. [84], which the authors tested on graphs with between 30 and 61 nodes, and between 47 and 187 edges. For the 12 test cases the authors considered, their proposed approach provided integer solutions for all but two, and for those, solutions obtained by a Branch-and-Bound scheme were very close to the fractional solutions provided by Column Generation. Chapter 5 also presents IP models for the considered partitioning problem, but is concerned with a Lagrangian Relaxation-, Column Generation- and matheuristic-based approaches, and is focussed on solving large instances with hundreds of classes and thousands of students.

The Edge Partitioning Problem (EPP), which is closely related to the GPP, is the problem of covering the edges of an edge-weighted graph with a set of edge-disjoint subgraphs, where the total edge weight in each subgraph is at most k [69]. The NP-hardness of the EPP is shown by Goldschmidt et al. [69], and also a linear time approximation algorithm with performance guarantee is proposed. Taşkın et al. [148] discuss a combination of integer and constraint programming methods for the stochastic EPP, and present a two-stage cutting plane algorithm.

The problem considered in Chapter 5 is also closely related to the Quadratic Multiple Knapsack Problem (QMKP), which is a combination of the multiple knapsack and quadratic knapsack problems. Given n objects and K knapsacks, the problem can be formulated as follows:

$$\text{max:} \quad \sum_{i=1}^{n}\sum_{k=1}^{K} u_i X_{i,k} + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{K} v_{i,j} X_{i,k} X_{j,k} \tag{2.5}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i X_{i,k} \leq c_k \qquad k = 1, \dots, K \tag{2.6}$$

$$\sum_{k=1}^{K} X_{i,k} \leq 1 \qquad i = 1, \dots, n \tag{2.7}$$

$$X_{i,k} \in \{0,1\} \qquad i = 1, \dots, n; k = 1, \dots, K \tag{2.8}$$

where the variable $X_{i,k}$ corresponds to the assignment of item i to knapsack k, u_i is the reward for assigning item i to any knapsack, $v_{i,j}$ is the reward for assigning items i in the same knapsack as item j, w_i is the weight of item i, and c_k is the maximum capacity of knapsack k.

The problem studied in Chapter 5 is distinct from the QMKP in several ways. In the language of the knapsack problem, our problem assigns a reward for assigning a particular item to a particular knapsack, and a reward for pairing particular item *types* of objects together in knapsacks. Our problem not only introduces a lower bound on the item weight in a knapsack (where we assume each item has unit weight), but also a lower- and upper-bound on the number of item *types* in each knapsack. Moreover, our problem permits certain knapsacks to remain empty. Lastly, our problem requires that each item must be assigned to exactly one knapsack.

The QMKP received little attention in the literature until recently, and most solution approaches are based on meta-heuristics [29, 63, 75, 87]. In particular, greedy and Genetic Algorithms (GA) for the

QMKP are discussed in Julstrom [87]. The author presents two greedy heuristics that build solutions by choosing objects according to their value densities, and two GA heuristics. One GA is a standard implementation, while the other is extended with greedy techniques that probabilistically favour objects of higher value densities. The algorithms are tested on 20 problem instances, and the extended GA is reported to perform best on all but one test case. In Chapter 5 we present a matheuristic that is an amalgamation of Genetic Algorithms and IP.

A Lagrangian Relaxation based approach to solving the QMKP exactly is presented in Caprara et al. [24], whereby a tighter bound is computed using the subgradient method. The proposed approach is able to optimally solve instances with up to 400 binary variables. In Chapter 5 we also consider a Lagrangian Relaxation based approach, but focus on problems with between 17,697 and 267,393 binary variables (and between 42,656 and 367,232 for the linearised models). The size of our problem, together with the complexity results presented in Chapter 5, leads to the aim of obtaining a good approximation solution in practically acceptable time.

Another closely related but distinct problem is the Quadratic Assignment Problem (QAP) [96], which can be formulated as follows:

$$\text{min:} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} c_{i,j,k,l} X_{i,j} X_{k,l} \tag{2.9}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} X_{i,j} = 1 \quad i = 1, \dots, N \tag{2.10}$$

$$\sum_{i=1}^{N} X_{i,j} = 1 \quad j = 1, \dots, N \tag{2.11}$$

$$X_{i,j} \in \{0,1\} \quad i, j = 1, \dots, N \tag{2.12}$$

where $X_{i,j}$ typically represents the assignment of object i to position j. The two sets of constraints require that each object is assigned to exactly one position, and each position is assigned exactly one object. The quadratic objective function penalises the simultaneous assignment of object i to position j, and object k to position l with penalty $c_{i,j,k,l}$.

The problem studied in Chapter 5 is distinct from the QAP in several ways. In the language of the QAP, our problem does not penalise the simultaneous assignment of object i to position j with the assignment of object k to position l where $j \neq l$. Our problem allows multiple objects to be assigned to the same position, and there are lower- and upper-bounds on the number of objects in each position, and lower- and upper-bounds on the number of object *types* in each position. Moreover, our problem permits certain positions remain empty.

Despite being easy to formulate, the QAP is difficult to solve, and was shown to be NP-hard by Sahni and Gonzalez [136]. Only implicit enumeration methods such as branch-and-bound [66, 96, 65, 121] or cutting plane methods [10] are known to solve QAP optimally, however heuristic approaches to solve the QAP, for example based on Simulated Annealing [159, 15], Tabu Search [143, 146, 57], and Genetic Algorithms [149, 60, 52], have been proposed.

2.6 Timetabling of Practice Placements

This section gives an overview of the literature as it pertains to the material presented in Chapter 6.

Much of the literature on educational timetabling focuses on high school and university course and examination timetabling [19, 22]. To the authors' knowledge, the considered optimisation problem of practice placement timetabling has not received the attention in the literature that it deserves. Among few publications relevant to the problem considered in this Chapter 6, Franz and Miller [61] consider the resident scheduling problem—a special case of the multiperiod staff assignment problem [4]—in which medical residents are assigned to rotations at hospitals. The authors present a decision support system in which feasible integer solutions were successfully obtained by means of a rounding heuristic applied to the linear relaxation of the binary Integer Programming model.

Beliën and Demeulemeester [11] discuss the problem of scheduling a number of trainees (students) at hospital ophthalmology department. The authors study the problem of scheduling a set of nursing trainees to specific activities along the planning horizon. The students are partitioned into a number of experience groups, and each task requires the allocation of a number of students from one or more experience groups. The authors detail several practical considerations, such as the requirement that students performing an activity for the first time will require more time until the student masters the relevant skills. The authors develop an exact procedure based on Column Generation, making use of a zero-one multi-commodity flow formulation with side constraints, where the pricing problem was formulated as a constrained shortest path problem. The authors were able to solve instances with 52 time periods.

The problem studied in this chapter can be viewed as a generalisation of the classical Open Shop Scheduling Problem (OSSP) [128]. In the OSSP, there is a set of machines and a set of jobs. Each job must be processed on each of the machines in any order, and the processing time of each job on each machine is specified. Each machine can process at most one job at a time, and each job can be processed on at most one machine at a time.

A Simulated Annealing (SA) solution approach was applied to the OSSP for the objective of minimising makespan by Liaw [104], where the author defines the neighbourhood structure based on blocks of operations on critical paths. The authors tested their approach on a number of randomly generated problems, as well as benchmark problems from the literature. Their approach was able to produce many optimal and near-optimal solutions, and some of the benchmark problems were solved to optimality for the first time.

In contrast to the OSSP, in the considered problem, each placement (which can be interpreted as a machine) can accommodate several apprentices (which can be viewed as jobs) simultaneously. The duration of the placement (which can be interpreted as the processing time of a job on a machine) in our problem is variable, and must be at least the minimum required time for this placement. The requirement of OSSP that each job must be processed on each machine is replaced by the partition of all placements (machines) into groups; the association of each apprentice (job) to a subset of these groups; and the requirement that each apprentice must be assigned to exactly one placement in each associated group. Furthermore, the objective function in our problem (to minimise total weighted penalty) differs from the objective functions commonly used in publications on OSSP. The typical objectives to minimise for the OSSP include, for example, makespan, maximum lateness, total completion time or total weighted completion time, total number of late jobs, etc.

It is well-known that the classical OSSP is a difficult problem. In particular, it is NP-hard for the criterion of makespan [70]. Since the planning horizon is given in the problem considered in Chapter 6, which imposes a restriction on the makespan, even the problem of answering whether or not there exists a feasible solution for the workplace training timetabling problem is NP-complete. The complexity of the feasibility problem is caused also by the restrictions on the number of apprentices allowed by each placement. In the formulation in Chapter 6, the planning horizon is partitioned into intervals of equal length. It will be shown in Section 6.3 that even if the planning horizon is comprised of just a single time interval, the feasibility problem remains NP-complete.

Chapter 3

Demand-Based Course Timetabling

Contents

The research in this chapter was presented at the 10th International Conference on the Practice and Theory of Automated Timetabling (PATAT) 2014 [125, 39] as "O. Czibula, H. Gu, Y. Zinder, and A. Russell. A multi-stage IP-based heuristic for class timetabling and trainer rostering. In *International Conference of the Practice and Theory of Automated Timetabling*, 2014", and was subsequently extended and published in the PATAT 2014 Special Issue by the Annals of Operations Research [37].

Abstract

We consider a timetabling and rostering problem involving periodic retraining of large numbers of employees at an Australian electricity distributor. This problem is different from traditional high school and university timetabling problems studied in the literature in several aspects. We propose a three-stage heuristic consisting of timetable generation, timetable improvement, and trainer rostering. Large-scale Integer Linear Programming (ILP) models for both the timetabling and the rostering components are proposed, and several unique operational constraints are discussed. We show that this solution approach is able to produce good solutions in practically acceptable time.

3.1 Introduction

Australia's largest electricity distributor (by number of connections and number of employees) is responsible for building, repairing, and maintaining all electricity distribution equipment that service their operational area of over 20,000 square kilometres. The voltages on the electricity network range from 230V to 500kV, and there is an extreme risk of electrocution if works are not performed carefully and with strict safeguards in place. Other hazards electrical workers face include falling from heights, having objects dropped on them from above, working in confined spaces, and working in the presence of hazardous materials such as toxic gas, asbestos, or other harmful substances. Having such a hazardous working environment and supplying such an essential service to the population, it is among the distributor's highest priorities to deliver safety and technical training promptly and efficiently to all people where required, including employees, contractors, and to third parties, working on or near the electricity network as required by Australian industry law. Indeed, it is a legal requirement in Australia—via the Work Health and Safety Act 2011: Part 2, Division 19; and enforced by the Energy Networks Association (ENA)—that such workers maintain their safety and technical training.

Most of the training in question has a limited validity period after which it is considered no longer valid unless "refreshed". Most courses have a validity period of 1 year, while some others can last 3 or 5 years, and a scarce few have unlimited validity periods. Validity periods are subject to change as industry legislation is occasionally revised. If a worker does not successfully refresh training for a job role before it expires, they are not permitted to work on or near the electricity network in that role until they have successfully completed the required training. Sometimes the initial training is longer and more comprehensive than subsequent training for that course. In some cases, if the subsequent training has not been successfully refreshed before the expiry date, the initial training must be completed again.

Many different training courses are delivered, each of which is composed of one or more modules. Each student enrolled in a course must successfully complete all the contained modules in order to successfully complete the course. Each module has a duration and a maximum number of students that it can accommodate. The modules of a course can be run in any order, however they must be run back-to-back, with no gaps between the modules. These are often referred to as 'no-wait constraints'. If a course runs longer than a working day, the modules must be sequenced such that none of the training days exceed the length of a single working day. Courses run for a maximum of five days, and must complete before the weekend. If a course has a total duration of half a day or less, it may either start first thing in the morning or immediately after lunch (which is fixed to be at noon for half an hour). Otherwise, if a course's duration is longer than half a day, it may only start first thing in the morning. Each course may be run an arbitrary number of times, perhaps several times in a single day and in several locations, and we refer to each individual instance of a course as a class.

The distributor's operational area can be partitioned into a number of regions, each containing one or more training facilities. We refer to each training facility as a location, and each location contains one or more rooms in various sizes. Some rooms have a removable divider that allows them to be split into two separate, smaller rooms. Each module of each class is run in a room. Each room has a list of compatible modules and a maximum number of students it can accommodate. Since both modules and rooms have a limitation on the number of students, each module-room pair has a maximum number of students given by the minimum of these two values. Different modules of a class can be assigned to different rooms, but all the modules of a class must be assigned to a single location.

Modules are taught by trainers, each of whom has a location to which they are regularly assigned, however trainers may travel to other locations when required. All trainers work a standard 8 hour work day with a half hour lunch break. There are different types of trainers, such as junior and senior trainers. All trainers have other responsibilities aside from teaching, therefore the proportion of working time each trainer is assigned to face-to-face teaching should be as close as possible to a value determined by the trainer's type.

Certain modules require shared, mobile resources which can be moved from location to location. The total number of these modules that can run at any given time in any given location is limited by the quantity of the required equipment locally present. Each mobile resource requires a certain amount of time to pack and unpack. The amount of time required to move a mobile resource from one location to another is given, and approximately proportional to the distance between them. Mobile resources can be moved at any time, including outside working hours. Mobile resources cannot be used while they are being moved.

When constructing a timetable, the distributor does not make decisions about which students will attend which classes, as this is out of the distributor's control. The reason is that training is offered not only to the distributor's own employees, but also to contractors and to third parties—anyone in any industry working on or near the electricity network. Due to the highly periodic nature of the training volume induced by the course validity periods, and taking into account other factors such as planned capital works, the distributor can estimate to a good approximation the expected training demand profile for each course in each region over the next year or so. Rather than creating a timetable incorporating which students will attend which classes, the distributor instead creates a timetable for classes and trainers only, based on the expected demand profile for each course in each region. Workers can then view the timetable using an online system, and book themselves into classes at times and places that are suitable and convenient for them. The distributor aims to schedule at least enough capacity for each course in each region across the planning horizon to meet the expected demand. If by two weeks before the course start date it still has not accrued the minimum class size in students, the distributor can then decide whether to cancel the class and notify the attendees, or permit the class continue with fewer students than normally required.

The robustness of the training plan is of great importance. A robust timetable will not need to be significantly changed in the event of unexpected circumstances such as room or trainer unavailability. Rather than approaching the issue of robustness in an explicit way, e.g. by incorporating extra variables or constraints, or running sensitivity analyses, we approach it in an implicit way by considering what timetable features undermine robustness, and avoiding them where possible.

One timetable feature that detracts from its robustness is a room swap, which is where the class must move from one room to another. In addition to being time consuming and inconvenient for the students and the trainer, if one of the rooms assigned to a class become unexpectedly unavailable and no substitute can be found, the class may need to be cancelled. Likewise, and for largely the same reasons, having trainer swaps—where the trainer teaching one module is different from the trainer teaching the previous module—also detracts from the robustness of the solution. Additionally, it is desirable for trainers not to travel excessively, as the further a trainer has to travel the more cost is incurred and the greater the likelihood the trainer will be unexpectedly delayed, for example due to heavy traffic, breakdowns, accidents, etc.

Since it is expected that people will be booked into classes at a time and place suitable and convenient to them, this should be considered during the timetabling process. Courses should be uniformly distributed throughout the planning horizon to maximise the likelihood that people will be able to find a class at a suitable time.

Certain rooms can be rented out to third parties if not in use for an extended period of time, generating some revenue for the distributor. Currently, due to administration costs, only a few rooms have the option to be rented out to third parties, however if the potential revenue generated can be shown to be substantial, the distributor may expand their room rental program.

Until recently, training timetables were designed by hand, requiring an experienced person at least three uninterrupted days to produce the timetable for one month. Currently, a software tool is used to generate their training plans on a month-by-month basis. This software is able to rapidly generate feasible timetables and rosters, however it does not contain any optimisation functionality. Due to changing industry regulations related to safety and technical training, as well as long-term fluctuations of demand, a more sophisticated tool is needed to manage and optimise their training plan that is capable of handling these changes in a flexible way.

The presented timetabling and rostering problem is a large-scale, multi-objective optimisation problem. It has many similarities to typical high school and university timetabling problems, but is distinguished from them in a number of important ways. The goal of this research is to investigate a

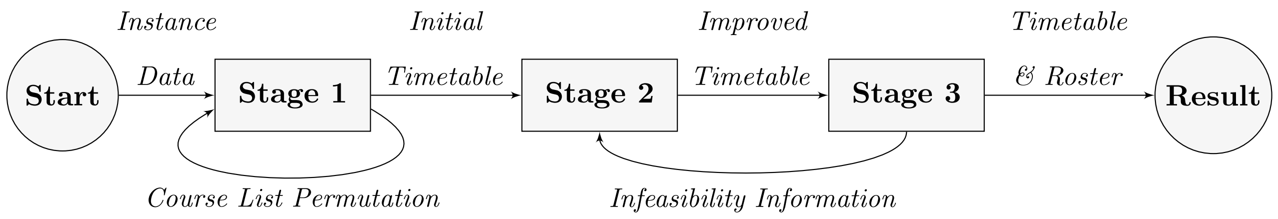


Figure 3.1: A high-level view of the three-stage approach.

flexible software tool incorporating mathematical optimisation techniques to help solve this difficult timetabling and rostering problem. Due to the large-scale nature of the considered problem, we have so far been unable to produce an optimal solution in practically acceptable time by means of direct application of mathematical programming. We propose a three-stage heuristic procedure consisting of an initial timetable generation stage, an iterative timetable improvement stage, and finally a trainer rostering stage. Integer Programming (IP) models are developed for each stage, which address all the current practical requirements, and are also flexible to changes in requirements.

The remainder of the chapter is organised as follows: Section 3.2 describes the three-stage heuristic in detail. Section 3.3 describes the class timetabling ILP model, and Section 3.4 describes the trainer rostering ILP model. Section 3.5 discusses some important details about the implementation of the approach. Section 3.6 describes our computational experimentation and results. Finally, our conclusions are given in Section 3.7.

3.2 Optimisation Procedure

In order to improve tractability and simplify the modelling of the considered problem, we have divided it into two related subproblems: a class timetabling subproblem and a trainer rostering subproblem. The former is the problem of allocating modules to rooms forming classes, where the objectives are that the unsatisfied student demand is minimised, room swaps are minimised, and the potential to rent out rooms is maximised. The latter is the problem of—subject to an existing timetable of classes—allocating trainers to modules, where the objective is to minimise trainer travel and minimise the number of trainer swaps. Aggregated trainer numbers, as opposed to individual trainers, are included in the class timetabling subproblem for capacity purposes only, in order to improve the likelihood that the timetable will allow for a feasible trainer roster. The combined solution to both subproblems results in a complete timetable and roster.

To have a flexible tool that is able to solve these complex subproblems, we have developed two IP models: one for the class timetabling subproblem and one for the trainer rostering subproblem. IP is flexible in the sense that one can simply add, remove, or substitute constraints to modify the model in various ways. Decomposing a larger, integrated problem into multiple subproblems, such as this training problem into the timetabling and rostering subproblems, often results in substantially faster optimisation, although usually at the cost of solution quality. In our case, even after decomposing the problem into two subproblems, the timetabling IP model was still too large to be solved in practically acceptable time given real world data for organisations such as the electricity distributor.

In order to produce a complete, usable timetable and roster in practically acceptable time, we propose a three stage heuristic approach (see Figure 3.1). The first two stages tackle the class timetabling subproblem, and the third stage tackles the trainer rostering subproblem. The first stage produces an initial feasible class timetable, the second stage attempts to improve the class timetable, and the third stage allocates individual trainers—subject to the timetable produced in the second stage—to form a roster.

3.2.1 Stage 1: Initial timetable construction

The considered class timetabling problem is NP-hard [45, 27]. We choose to construct the initial class timetable using the class timetabling IP model rather than using metaheuristics, as each generated solution to the IP is guaranteed to be optimal with respect to the model being solved, even though the constructed initial timetable as a whole is unlikely to be optimal.

The first stage constructs an initial feasible timetable incrementally. This is accomplished by successively solving the class timetabling IP model for one class at a time. Each time a class is scheduled, the rooms and other resources it occupies become unavailable for subsequent classes for its duration. As more and more classes are present in the partial schedule, more rooms become unavailable at certain times, and more resources and trainers are consumed in particular locations at particular times. In each subsequent iteration, variables corresponding to the occupied rooms, resources, and trainers in the partial solution are fixed when creating the IP model. As less rooms, resources, and trainers become available in each subsequent iteration, the resulting models tend to have, in general, fewer decision variables. This constructive approach is conceptually similar to so-called Direct Heuristics [138].

The order of the list that determines the sequence in which classes are to be scheduled can be arbitrary. Scheduling courses one by one using different sequences likely results in different timetables with different objective values. It is reasonable to assume that if the most demanding courses—those which have longer durations, with many modules, and very specific room and resource requirements—are scheduled first, and the least demanding courses are scheduled last, then it is more likely that a feasible timetable can be constructed. Based on similar ideas in [26] and [88], a scheduling complexity value should be estimated for each course, and classes should be scheduled in decreasing order of these

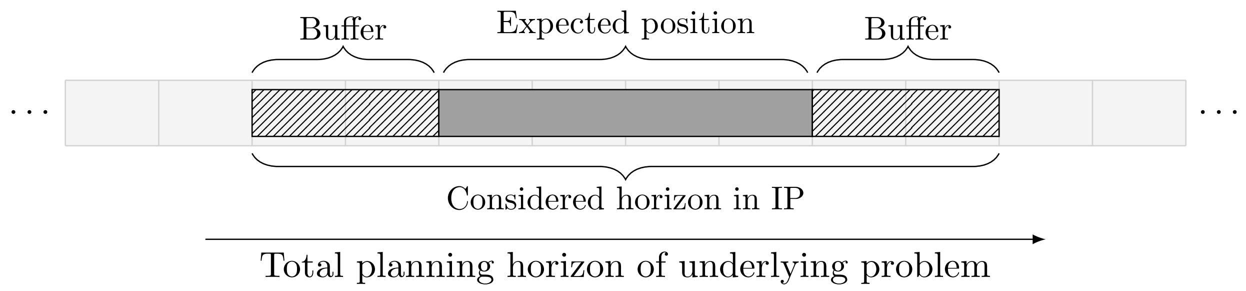


Figure 3.2: A buffer added to either end of the reduced planning horizon.

values. If the courses are scheduled in descending order of their estimated scheduling complexity, we can reasonably expect there to be a greater likelihood that an initial timetable can be constructed in the first stage that minimises unsatisfied demand than if the classes were scheduled in a very different order. The course scheduling complexity value can be estimated by considering:

- the total duration of the course,
- the number of unique mobile resources required by the modules of the course, and the duration for which they are needed, and
- the room specificity of the modules in the course, i.e. how few rooms the modules are compatible with, and how many unique, specific rooms are required, and how many other locally in-demand courses also compete for those rooms.

Since the problem being solved at each iteration of the first stage is relatively small, several initial timetables can be constructed by considering the above-mentioned order with minor perturbations, such as swapping two classes in the list. The timetable with the best objective value can be used in subsequent stages.

To further reduce the size of the IP model, we can add additional restrictions based on the class being scheduled. We can determine when each class should ideally run in order for the classes of that course to be uniformly distributed along the timeline. Then, when allocating a class, we can consider a much shorter planning horizon centred around this time as opposed to considering the entire planning horizon. However, since we cannot guarantee that the course can be scheduled at exactly the times we want, especially in the later iterations, this restricted planning horizon may need to be extended to allow some freedom in scheduling (see Figure 3.2). The shorter the considered planning window, the more control we have over the position of the course and the smaller the IP model will be, however a feasible solution is less likely exist, particularly in later iterations. The buffer in the planning horizon can start out small, and be incrementally increased if no solution can be found.

When scheduling classes one by one, it is also possible, without loss of generality, to consider only a single region in the IP model. Looking at the demand of a course, together with the state of the current partial solution, we decide in which region the next class should be placed, and include only that region in the IP model. Moreover, any room incompatible with all of the course's modules should also be excluded from the IP model.

The final IP for each iteration of the first stage becomes substantially smaller and can be typically solved in a few seconds on a modern desktop computer. Given real-world data from the electricity distributor, the model being solved at each iteration contains one class with 1 to 5 modules; a planning horizon of between 24 and 144 time periods; and one region with a 1 to 4 locations, each with 1 to 15 rooms.

3.2.2 Stage 2: Timetable improvement

The first stage constructs an initial feasible timetable that it is unlikely to be optimal, since poor decisions made at the early stages of the process can have a compounding effect on the remaining allocations. The second stage attempts to improve the timetable using a Large Neighbourhood Search (LNS) fix-and-optimise heuristic. As with the first stage, the second stage makes use of the timetabling IP model to produce new solutions.

At each iteration, the components of the incumbent solution whose change may improve the objective value are identified. For example, suppose one class contributes a penalty to the objective value because it contains some room swaps. Perhaps the room swaps were necessary at the time the initial timetable was constructed as there was no other way to fit it in subject to the surrounding classes, but it may be possible to eliminate the room swaps. To attempt to improve the timetable in this regard, the identified class, together with at least one other class, should be removed from the timetable and the class timetabling IP model should be re-solved for the removed classes, subject to the remaining timetable. The more classes are removed and re-scheduled simultaneously, the better the outcome is likely to be, however the computational effort required grows rapidly in the number of selected classes.

See Figure 3.3 for an illustration of a poorly constructed timetable, where three classes are scheduled one by one, in order. Due to the limited compatibility between modules and rooms, and poor decisions early in the process, the modules of Class 3 have been assigned to three separate rooms. During Stage 2, Class 3 is identified as contributing a penalty to the objective value due to the room swaps. Solving the timetabling IP model for only Class 3 alone cannot produce a better solution as it is already in the optimal position subject to the surrounding classes. If the timetabling IP model is solved for all three classes simultaneously, then the timetable shown in Figure 3.4 will be produced, eliminating the room swap cost of Class 3 entirely.

At each iteration, the only variables present in the IP are those pertaining to the selected classes; all

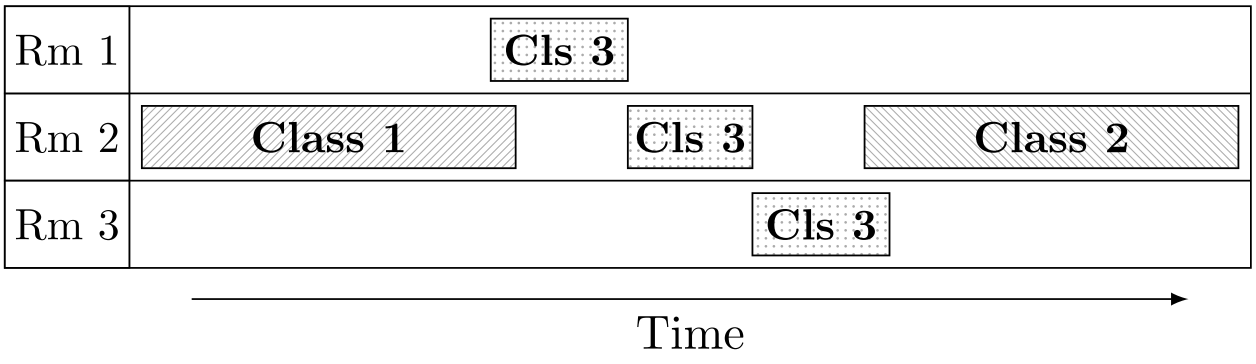


Figure 3.3: An example of a timetable with unnecessary room swaps.

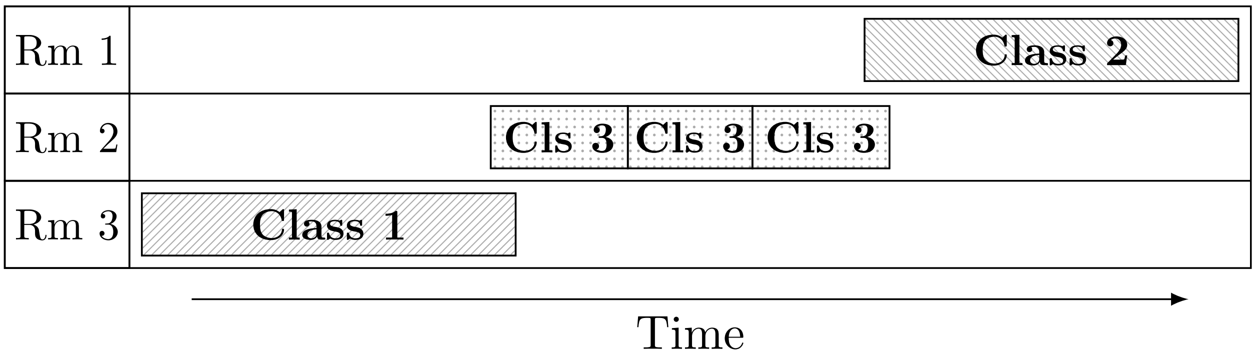


Figure 3.4: Unnecessary room swaps from the example in Figure 3.3 avoided.

other variables are fixed to a constant value determined by the state of the remainder of the timetable. The "large neighbourhood" in the LNS heuristic is the set of all feasible solutions to the IP model. Stage 1 is a "fix-and-optimise" heuristic because, at each iteration, a set of variables are fixed, and the remainder of the variables or optimised. Stage 2 continues until a stopping criterion has been reached, such as a time limit or failure to improve the timetable after a certain number of iterations.

Stage 2 attempts to improve situations where there are:

1: courses where not all demand has been satisfied,

2: classes with many room swaps, and

3: classes occupying rooms that could otherwise be rented out.

We identify several classes to be rescheduled that are relevant to one another with respect to opportunity to make improvements. In the case of 1, the classes of a selected course whose demand has not been satisfied in a given region and time period should be paired with other classes that have an overlapping set of compatible rooms with the selected classes. If necessary, additional classes can be created for the courses with insufficient capacity. In the case of 2, a class which contains room swaps should be paired with one or more other classes in the same region and time period, and that have an overlapping set of compatible rooms with the selected class. In the case of 3, a set of selected classes that occupy rooms that could otherwise be rented out should be paired with one or more other classes in the same region and time period, and that have an overlapping set of compatible rooms with the selected classes, and that are not currently occupying the rentable rooms. It is important to balance the desire to make more substantial improvements by selecting as many classes as possible, with the need to keep size of the IP model manageable by selecting as few classes as possible. The procedure in Stage 2 is heuristic, and therefore cannot guarantee an optimal solution. The algorithm is also vulnerable to becoming stuck in a local minimum if the selected neighbourhood is not sufficiently large.

Dorneles et al. [50] solve a Brazilian high school timetabling problem with a similar IP-based fix-and-optimize heuristic that is conceptually very similar to the Stage 2 (improvement) approach discussed here. In their implementation, they select one of three possible neighbourhood types: classes, teachers, or days. They explore the set of neighbourhoods by means of a Variable Neighbourhood Descent (VND) method, in which they limit the size of the neighbourhood by some predefined parameter, and the neighbourhoods are selected randomly. The sizes of their subproblems are sufficiently small to enable repeated solution by a general-purpose MIP solver in acceptable time. Our Stage 2 approach is similar in that we also solve subproblems, defined by a subset of all variables of the whole timetabling problem, with respect to the remaining class timetable. In contrast, our neighbourhoods are selected by analysing the incumbent timetable and identifying which scheduled classes negatively affect the objective value. Of the neighbourhood types discussed in [50], the 'classes' neighbourhood type is the only one applicable in our case, as our timetabling subproblem only includes aggregate trainers, and the length of courses relative to time granularity make the 'days' neighbourhood computationally impractical. The size of our neighbourhoods is adaptive, based on analysis of which other classes are most relevant with respect to opportunity to make improvements, to the classes being improved in a particular iteration.

3.2.3 Stage 3: Rostering

In the third stage, the rostering IP model is used to allocate specific trainers to each module on the timetable. While the IP for the timetabling subproblem cannot be solved by direct application of Integer Programming for real-world data due to the size of the model, the IP model for the rostering subproblem—subject to the produced timetable—is considerably smaller and can be solved by a general-purpose MIP solver on a modern desktop computer typically in under a minute.

We found that the network model in [7] provides an efficient solution approach for our rostering problem, which is represented as a one minimum cost flow networks for each trainer, together with some non-network side constraints. The objective of the rostering IP model is to minimise the total flow cost. The structures of the networks are given by the timetable for which the roster is being generated. Flow along a particular arc on a network corresponds to a particular trainer being allocated to a module. Since the locations of trainers and the locations of scheduled modules are known, it is simple to determine the distance each trainer will need to travel in order to teach a given set of modules. Moreover, determining the number of trainer swaps is as simple as counting the incidences where the ith trainer differs from the $(i+1)$th trainer for a given class. Flow along each arc on each network therefore represents some amount of travel distance and the possibility of a trainer swap. The flow cost of each arc is given by a linear combination of the travel cost and trainer swap cost. While worker pay is often a high priority in many other rostering problems, we do not consider it in our rostering subproblem, as the distributors trainers are paid a fixed salary, regardless of how many hours they teach. The case of considering workforce pay may be a consideration in future research.

If the produced timetable has no feasible solution with respect to trainer rostering, the reason must be determined, and the problem should be corrected in the timetable. Due to the nature of the rostering subproblem, infeasibility can only arise if there are insufficient compatible trainers to teach the modules on the timetable. If there are n trainers capable of teaching a particular set modules, however there are $m > n$ of those modules running at a particular time period in a particular location, then there can be no feasible roster. In this case, the algorithm returns to stage two and solves the timetabling model with the additional constraint that, at all times, the total number of times modules requiring this type of trainer running at any given time must not exceed n. If this approach continues to fail to admit a feasible roster, some classes may need to be removed from the timetable resulting in some unmet course demand.

3.3 Timetabling Model

This section describes an Integer Programming model to create a timetable of classes subject to demand, following the problem description above. The first two of the three stages in the heuristic procedure discussed in Section 3.2 make use of this model.

3.3.1 Time Discretisation

In the timetabling model, time is discretised into half-hour periods. Times that are not available for training, such as lunch times and any time outside standard working hours, are not present in the set of time periods. Periods are grouped into coarser intervals called days, each of which contain exactly 15 periods. We group multiple days into longer units that we denote time windows. Each time window corresponds to a month, and contains each working day from that month. Finally, periods are grouped together into rental windows. A rental window represents a set of consecutive periods in which rooms may be rented out to a third party. Rental windows range from being half a day to several days long.

Rental windows may overlap with one another, such as a morning, an afternoon, and a whole day rental window for a particular day. If a rentable room is unoccupied for one or more rental windows, it may be rented out to a third party, potentially bringing in additional revenue.

3.3.2 Input Data Set-up

Some pre-processing on the input data can reduce the size of the resulting IP models. For the purpose of the class timetabling model, we fix the number of classes of each course to a practical estimation. While in practice each individual module can exist in many different courses, for the purposes of this IP model, we require each module to be part of exactly one class - we refer to these as module instances. Each module in each course must therefore be duplicated such that each class has a set of unique modules.

Each location may have several rooms, however some rooms may be regarded as identical, i.e. they have the same list of compatible modules, the same capacity, the same rental income, etc. In this case, we can group identical rooms into 'room types', where each room type represents a set of rooms in a given location that are functionally identical. Rooms in different locations are never grouped together, and each compound room—rooms with removable dividers—has its own, unique room type.

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

3.3.3 List of Symbols

Notation:

$S^{(i)}$	The i^{th} element of any indexed set S.

Sets of primary objects:

P	The indexed set of time periods.
D	The indexed set of days.
Ω	The indexed set of time windows.
Λ	The indexed set of rental windows.
L	The set of locations.
Ξ	The set of regions.
$\hat{M}$	The indexed set of module instances.
C	The set of courses.
R	The set of room types.

T	The set of resource types.

Sets of derived objects:

I_c	The indexed set of classes for course $c \in C$.
$B_{c,i}$	The indexed set of module instances for course $c \in C$ class $i \in I_c$.
P_d	The indexed set of periods in day $d \in D$.
$\bar{P}_\omega$	The indexed set of periods in time window $\omega \in \Omega$.
$\tilde{P}_\lambda$	The indexed set of periods in rental window $\lambda \in \Lambda$.
$\hat{P}_c$	The set of periods in which course $c \in C$ is permitted to start.
$\bar{L}_\xi$	The set of locations in region $\xi \in \Xi$.
$\tilde{R}$	The set of mutually exclusive room pairs, where $\{r_1, r_2\} \in \tilde{R}$ for $r_1, r_2 \in R, r_1 \neq r_2$.
$\tilde{R}_l$	The set of room types in location $l \in L$.
R_m	The set of room types suitable for module $m \in \hat{M}$.

Primary decision variables:

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

Auxiliary variables:

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

Constants:

σ_t	$\in \mathbb{Z}^{++}$	The quantity available of resource $t \in T$.
$\delta_{t,l,k}$	$\in \mathbb{Z}^{+}$	The time required (in periods) for a unit of resource $t \in T$ to move from location $l \in L$ to location $k \in L$.
$\theta_{l,d}$	$\in \mathbb{Z}^{+}$	The number of trainers normally allocated to location $l \in L$ on day $d \in D$.
$\theta^{\max}$	$\in \mathbb{Z}^{+}$	The maximum number of additional trainers permitted to any location on any given day.
$\theta^{\min}$	$\in \mathbb{Z}^{+}$	The maximum number of subtracted trainers permitted from any location on any given day.
$Q_{r,p}$	$\in \mathbb{Z}^{+}$	The quantity of room type $r \in R$ available at period $p \in P$, or 0 otherwise.
$d_{r,\lambda}$	$\in \mathbb{R}^{+}$	The expected revenue from renting out a unit of room type $r \in R$ during rental window $\lambda \in \Lambda$.
l_c	$\in \mathbb{Z}^{++}$	The length (in periods) of course $c \in C$.
π_c	$\in \mathbb{Z}^{++}$	The length (in periods) of the rolling time window used to compute the minimum and maximum number of times a course $c \in C$ should be run.

π_c^+	$\in \mathbb{Z}^{++}$	The maximum number of times a course $c \in C$ should be run in any given time window of length π_c.
π_c^-	$\in \mathbb{Z}^{+}$	The minimum number of times a course $c \in C$ should be run in any given time window of length π_c.
$s_{c,\omega,\xi}$	$\in \mathbb{Z}^{+}$	The demand (in students) for course $c \in C$ during window $\omega \in \Omega$ in region $\xi \in \Xi$.
α_i	$\in \mathbb{R}^{+}$	The coefficient of the ith goal in the objective function.
u_m	$\in \mathbb{Z}^{++}$	The maximum number of students module $m \in \hat{M}$ can hold.
v_r	$\in \mathbb{Z}^{++}$	The maximum number of students room type $r \in R$ can hold.

3.3.4 Core Timetabling Constraints

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

$$\sum_{q=0}^{d_m-1} X_{m,r,(p-q)} = \hat{X}_{m,r,p} \qquad \forall m \in \hat{M}, r \in \hat{R}_m, p \in P \tag{3.1}$$

$$\sum_{m\in\hat{M}} \hat{X}_{m,r,p} \leq Q_{r,p} \qquad \forall r \in R, p \in P \tag{3.2}$$

$$\sum_{m\in\hat{M}} \hat{X}_{m,\tilde{r}_1,p} + \sum_{m\in\hat{M}} \hat{X}_{m,\tilde{r}_2,p} \leq 1 \qquad \forall \{\tilde{r}_1, \tilde{r}_2\} \in \tilde{R}, p \in P \tag{3.3}$$

$$\sum_{r\in R}\sum_{p\in P} X_{m,r,p} = \bar{Y}_{c,i} \qquad \forall c \in C, i \in I_c, m \in B_{c,i} \tag{3.4}$$

$$\sum_{p\in\hat{P}_c} Y_{c,i,p} = \bar{Y}_{c,i} \qquad \forall c \in C, i \in I_c \tag{3.5}$$

The auxiliary variables $\hat{X}_{m,r,p}$ are set up from $X_{m,r,p}$ according to (3.1). The constraints (3.2) express the requirement that rooms should not be double-booked, however since identical rooms within a single location are aggregated together, the right-hand-side is given by the quantities of the aggregated rooms. The constraints (3.3) also express the requirement that splittable rooms should not be double-booked, however since splittable rooms are never aggregated together, the right-hand-side remains 1. The constraints (3.4) ensure that each module of a course is run exactly once if the course is run, or not at all. The expressions (3.5) set up the $\bar{Y}_{c,i}$ variables, which is a sum over all periods of $Y_{c,i,p}$, and also imply $\sum_{p\in\hat{P}_c} Y_{c,i,p} \leq 1$ for all classes.

3.3.5 Characteristic Constraints

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

Module Positioning Constraints

$$\sum_{m \in B_{c,i}} \sum_{r \in R_m} \hat{X}_{m,r,p} = \sum_{q=0}^{l_c - 1} Y_{c,i,(p-q)} \qquad \forall c \in C, i \in I_c, p \in P \tag{3.6}$$

$$\sum_{m \in B_{c,i}} \sum_{r \in \tilde{R}_l} \sum_{p \in P} X_{m,r,p} = |B_{c,i}| \times \hat{Y}_{c,i,l} \quad \forall c \in C, i \in I_c, l \in L \tag{3.7}$$

The constraints (3.6) expresses the requirement that the modules in a class run back-to-back, and (3.7) expresses the requirement that all the modules in a class must be run in exactly one location.

Capacity Constraints

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

$$\tilde{Y}_{c,i,\omega,\xi} \leq \mathscr{M}_{c,\xi} \sum_{p \in \bar{P}_\omega} Y_{c,i,p} \qquad \forall c \in C, i \in I_c, \omega \in \Omega, \xi \in \Xi \tag{3.8}$$

$$\tilde{Y}_{c,i,\omega,\xi} \leq \mathscr{M}_{c,\xi} \sum_{l \in \bar{L}_\xi} \hat{Y}_{c,i,l} \qquad \forall c \in C, i \in I_c, \omega \in \Omega, \xi \in \Xi \tag{3.9}$$

$$\tilde{Y}_{c,i,\omega,\xi} \leq \mathscr{M}_{c,\xi} \left(1 - \sum_{p \in \bar{P}_\omega} X_{m,r,p} \right) + \min\{u_m, v_r\} \quad \forall c \in C, i \in I_c, \omega \in \Omega, \xi \in \Xi, \; m \in B_{c,i}, l \in \bar{L}_\xi, r \in \tilde{R}_l \tag{3.10}$$

The constraints (3.8)-(3.10) set up the $\tilde{Y}_{c,i,\omega,\xi}$ variables given some sufficiently large constant $\mathscr{M}_{c,\xi}$. For best performance, the value of $\mathscr{M}_{c,\xi}$ should be chosen to be as small as possible, which is the largest course capacity for the course c in the rooms available in region ξ.

3.3.6 Trainer Movement Constraints

In this model, trainers are considered in a generalised, aggregated way for capacity purposes only. Nevertheless, we permit the quantity of these generalised trainers to change per location per day to give a coarse representation of trainer movements. Each trainer has a location where they are normally based, however they may be required to travel to other locations.

$$\phi_{l,d} \leq \theta_{l,d} + \theta^{\max} \qquad \forall l \in L, d \in D \tag{3.11}$$

$$\phi_{l,d} \geq \theta_{l,d} - \theta^{\min} \qquad \forall l \in L, d \in D \tag{3.12}$$

$$\sum_{l\in L}\phi_{l,d} = \sum_{l\in L}\theta_{l,d} \qquad \forall d \in D \tag{3.13}$$

$$\sum_{m\in\hat{M}}\sum_{r\in\tilde{R}_l}\hat{X}_{m,r,p} \leq \phi_{l,d} \quad \forall l \in L, d \in D, p \in P_d \tag{3.14}$$

The constraints (3.11) and (3.12) establish the minimum and maximum number of trainers permitted to be at a given location on a given day, and the constraints (3.13) ensure that the total number of trainers allocated to each location is equal to the total number of trainers expected to be working company-wide on that day. The constraints (3.14) express the requirement that, at any given time, the total number of modules run in a location concurrently must not exceed the number of generalised trainers we have chosen to allocate there.

3.3.7 Resource Movement Constraints

A flow network can be help to visualise the flow of resources between locations across time. Resources, in this context, refer to shared, mobile pieces of equipment that are required for teaching particular modules. For each type of resource and for each day, we can construct a flow network with the nodes arranged in a rectangular lattice (See Figure 3.5). The horizontal axis represents time, and the vertical axis represents the various locations. Each node represents the end points of a time period at a given location. Adjacent nodes are connected by directed arcs horizontally and pointing forward in time, with the flow along those arcs representing the quantity of the resource available at a particular location at a particular time. Nodes are also connected between different locations by directed arcs in such a way that the time interval from the source node to the destination node is given by the time required to move the resource from the source location to the destination. This flow network is merely a modelling aid, and not solved directly by flow network algorithms. However the flow balance equations from the network can be transcribed directly into the resource movement constraints in the timetabling IP. One can say that the timetabling IP model contains a network flow subproblem.

We permit resources to move from any location to any other location overnight at no cost, therefore the initial condition for each resource network for each day is simply that the sum across all locations must equal the available quantity of that resource.

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

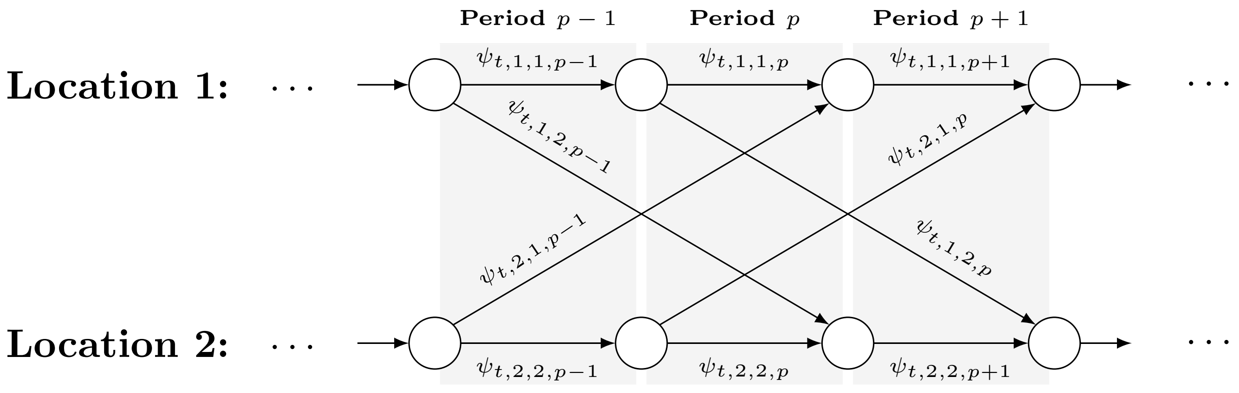


Figure 3.5: A sample flow network for some resource t about period p with 2 locations.

The flow balance equations for the networks are expressed as follows:

$$\sum_{l\in L}\sum_{k\in L}\psi_{t,l,k,P_d^{(1)}} = \sigma_t \qquad \forall t\in T, d\in D \tag{3.15}$$

$$\sum_{k\in L}\psi_{t,k,l,(p-\delta_{t,k,l})} = \sum_{k\in L}\psi_{t,l,k,p} \quad \forall t\in T, l\in L, d\in D, p\in (P_d\setminus P_d^{(1)}) \tag{3.16}$$

We ensure that the number of times resources are used is limited by the number available at the time:

$$\sum_{m\in\hat{M}_t}\sum_{r\in\tilde{R}_l}\hat{X}_{m,r,p} \le \psi_{t,l,l,p} \quad \forall l\in L, t\in T, p\in P \tag{3.17}$$

3.3.8 Spreading Constraints

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

$$\pi_c^- \le \sum_{i\in I_c}\sum_{q=0}^{\pi_c} Y_{c,i,p+q} \le \pi_c^+ \quad \forall c\in C, p\in\hat{P}_c \tag{3.18}$$

The constraints (3.18) establish the minimum and maximum number of classes, respectively, that must be run across all regions for each course. The value of the π_c and the π_c^- and π_c^+ constants is determined given the problem data.

3.3.9 Objective Function

Being a large-scale industrial problem, there are many potential objectives we can consider. In this research we consider three objectives in a weighted linear function:

- minimise the number of expected students not accommodated,
- maximise the rental revenue, and

- minimise the number of room swaps in the timetable.

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

$$\sum_{i\in I_c} \tilde{Y}_{c,i,\omega,\xi} + u_{c,\omega,\xi} = s_{c,\omega,\xi} \quad \forall c \in C, \omega \in \Omega, \xi \in \Xi \tag{3.19}$$

$$u_{c,\omega,\xi} \geq 0 \quad \forall c \in C, \omega \in \Omega, \xi \in \Xi \tag{3.20}$$

$$\sum_{c\in C}\sum_{\omega\in\Omega}\sum_{\xi\in\Xi} u_{c,\omega,\xi} = Z_1 \tag{3.21}$$

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

$$\sum_{m\in\hat{M}} \hat{X}_{m,r,p} \leq \rho_{r,\lambda} \quad \forall r \in R, \lambda \in \Lambda, p \in \tilde{P}_\lambda \tag{3.22}$$

$$Z_2 = \sum_{r\in R}\sum_{\lambda\in\Lambda} \left[-d_{r,\lambda} \times (\hat{Q}_{r,\lambda} - \rho_{r,\lambda})\right] \tag{3.23}$$

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

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

$$X_{m,r,p} - \sum_{n\in B_{c,i}, m\neq n} \hat{X}_{n,r,(p-1)} \leq t_m \quad \forall c \in C, i \in I_c, m \in B_{c,i}, r \in R_m, p \in P \tag{3.24}$$

$$Z_3 = \sum_{m\in\hat{M}} t_m \tag{3.25}$$

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

$$\text{minimise:} \quad Z = \alpha_1 Z_1 + \alpha_2 Z_2 + \alpha_3 Z_3 \tag{3.26}$$

with nonnegative weights α_1, α_2, and α_3.

In reality, the timetabling problem has an ordered bi-criteria objective: the goal is to maximise the potential room rental revenue and minimise the number of room swaps (Z_2 and Z_3) over the set of timetables that minimise the unmet demand (Z_1). This can be achieved by solving the timetabling subproblem for the primary objective of minimising Z_1, and then solving the timetabling subproblem again for the secondary objectives of minimising over Z_2 and Z_3, but with the additional constraint that Z_1 must assume the optimal value. A simpler way of achieving the same result is to give a sufficiently large coefficient α_1 with respect to the coefficients α_2 and α_3. A similar subject of study is Lexicographic Optimisation, in which multiple ordered objectives are present, where lower priority objectives are optimised so long as they don't interfere with higher priority objectives [81, 163]. Lexicographic Optimisation is an application of Goal Programming [99], a branch of Multiobjective Optimisation.

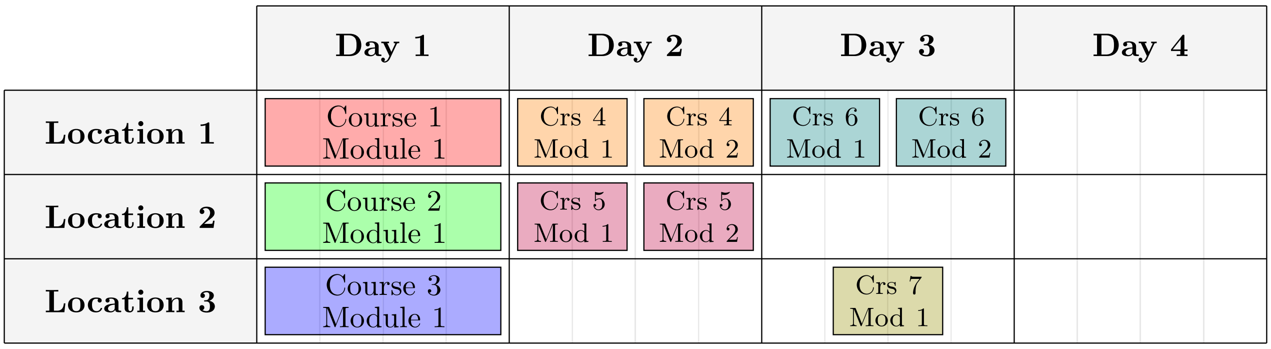


Figure 3.6: A sample timetable, simplified for viewing in this format, showing 4 days, 3 locations, and 7 courses each with 1 or 2 modules.

3.4 Rostering Model

Given a solution to the class timetabling problem from Section 3.3, the rostering model describes the problem of allocating specific trainers to modules, resulting in a complete timetable and roster. A minimum cost network flow approach, together with some non-network side constraints, can be utilised to give a simple and convenient representation of the rostering problem.

Given a timetable, a min-cost flow network is constructed for each trainer. There are two different types of nodes, and four different types of directed arcs in the networks: home nodes, activity nodes, commencement arcs, transition arcs, return arcs, and bypass arcs. Home nodes represent the location where the trainer starts and ends their work day. Activity nodes represent specific modules that can be taught by the trainer. Commencement arcs—representing the trainer commencing training for a particular day—originate from home nodes and end at activity nodes. Transition arcs—representing trainer finishing teaching one module and then teaching another (with or without a break)—originate from activity nodes and end at activity nodes. Return arcs—representing the trainer completing their training for a particular day—originate from activity nodes and end at home nodes. Bypass arcs—representing a particular day when a trainer will not deliver any training—originate at home nodes and end at home nodes.

As an example, Figure 3.6 shows a simplified view of a timetable with 7 courses, and the corresponding min-cost flow network is shown in Figure 3.7. In Figure 3.7, flow along any of the arcs from (4:1)→(5:2), (H2)→(4:2), (H2)→(5:2), (5:1)→(4:2), or (H3)→(6:2) indicate the presence of a trainer swap. Care should be taken not to double-count trainer swaps. The convention we use is to count only classes where the incoming trainer differs from the preceding trainer, not when the outgoing trainer is different from the proceeding trainer. For our incoming-only convention, we do not count flow along the arcs (4:1)→(H3), (5:1)→(H3), or (6:1)→(H4) when determining the number of trainer swaps.

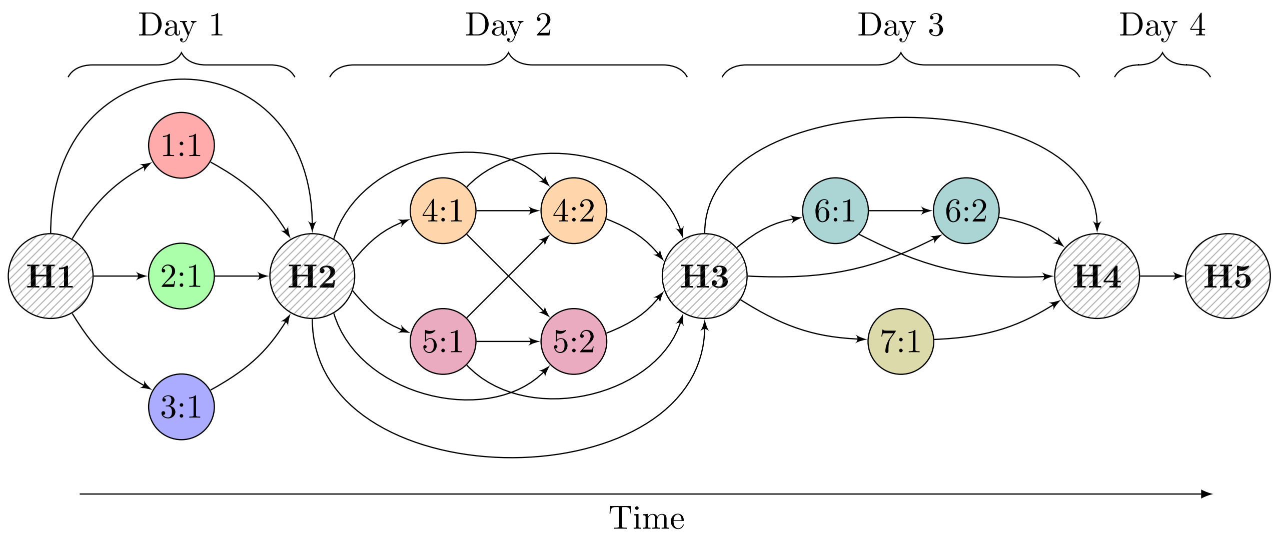


Figure 3.7: The min-cost flow network corresponding to the sample timetable shown in Figure 3.6. (Home nodes are hatched, and activity nodes are solid)

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

We introduce the following symbols for the rostering model:

Notation:

$pred(m)$	The set of predecessors of module $m \in \hat{M}$.
$succ(m)$	The set of successors of module $m \in \hat{M}$.

Sets of objects:

D	The indexed set of days.
$\hat{M}$	The indexed set of module instances.
M_d	The set of modules that run within day $d \in D$
T	The set of trainers.
T_m	The set of trainers capable of teach module $m \in \hat{M}$.

Variables:

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

The flow balance equations for the network are as follows:

$$\hat{\psi}_{\tau,d} + \sum_{m \in M_d} \bar{\psi}_{\tau,m} = 1 \qquad \forall \tau \in \mathrm{T}, d \in D \tag{3.27}$$

$$\bar{\psi}_{\tau,m} + \sum_{n \in pred(m)} \psi_{\tau,n,m} = \sum_{n \in succ(m)} \psi_{\tau,m,n} + \tilde{\psi}_{\tau,m} \quad \forall \tau \in \mathrm{T}, m \in \hat{M} \tag{3.28}$$

where (3.27) ensures that, at the start of each day, the trainer either teaches one or more modules or does not teach any modules; and (3.28) conserves flow throughout the day. Since the flow for each day is implicitly conserved by (3.27) and (3.28), we do not require any additional equations to balance the flow from day to day.

Any integral flow is always a feasible roster for a single trainer, since the network is constructed in such a way the trainer can never be required to be in two places at once, nor can they be required to teach a module they are not capable of teaching.

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

$$\bar{\psi}_{\tau,m} + \sum_{n \in pred(m)} \psi_{\tau,n,m} = X_{m,\tau} \qquad \forall m \in \hat{M}, \tau \in \mathrm{T} \tag{3.29}$$

$$\sum_{\tau \in \mathrm{T}_m} X_{m,\tau} = 1 \qquad \forall m \in \hat{M} \tag{3.30}$$

where (3.29) sets up the auxiliary variable $X_{m,t}$, which is 1 if trainer t teaches module m or 0 otherwise, and (3.30) ensures that every scheduled module is taught by exactly one trainer.

In order to ensure fairness, we wish to avoid, wherever possible, the situation where one trainer is scheduled to train much more or less than their peers:

$$U_\tau^- \leq \sum_{m \in \hat{M}} (w_m \times X_{m,\tau}) \leq U_\tau^+ \quad \forall \tau \in \mathrm{T} \tag{3.31}$$

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

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

$$\begin{aligned}\min \sum_{\tau \in \mathrm{T}} \sum_{m \in \hat{M}} \left[c_1(\tau, m) \times \tilde{\psi}_{\tau,m}\right] + \sum_{\tau \in \mathrm{T}} \sum_{m \in \hat{M}} \sum_{n \in \hat{M}} \left[c_2(\tau, m, n) \times \psi_{\tau,m,n}\right] + \\ \sum_{\tau \in \mathrm{T}} \sum_{m \in \hat{M}} \left[c_3(\tau, m) \times \bar{\psi}_{\tau,m}\right] + \sum_{\tau \in \mathrm{T}} \sum_{d \in D} \left[c_4(\tau, d) \times \hat{\psi}_{\tau,d}\right]\end{aligned} \tag{3.32}$$

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

3.5 Implementation

As a result of the back-to-back restriction when scheduling the modules of a class, there is an implied set of feasible start times for each module instance. These can be determined by permuting the order of the modules of the course, and determining the start time of each module relative to the course start time. Clearly, if no module permutation can allow for certain modules to start at certain times, then the variables corresponding to those module start times can be safely eliminated from the model.

The solution space of the timetabling model exhibits some symmetry, as, for a given timetable, the indices of the classes of any course can be permuted without affecting the objective value. There are $n!$ ways of indexing the n classes of a single course across an existing timetable. We can eliminate many symmetric solutions by introducing the following constraints:

$$\sum_{q=0}^{p} Y_{c,i,q} \geq Y_{c,(i+1),p} \quad \forall c \in C, i \in I_c, p \in P_c \tag{3.33}$$

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

In Stage 2 (improvement), since the solution at iteration i is feasible at iteration $i+1$, it can be provided to the IP solver as a "warm start" or "advanced start" to begin the search procedure [77].

3.6 Computational Results

The electricity distributor supplied both current and historical data related to class timetabling and trainer rostering. The overwhelming majority of their training volume comes from the eight most frequently run safety courses, which have between one and four modules and needed up to 26 classes each for a monthly timetable. There were five regions and 15 locations, and room counts ranged from

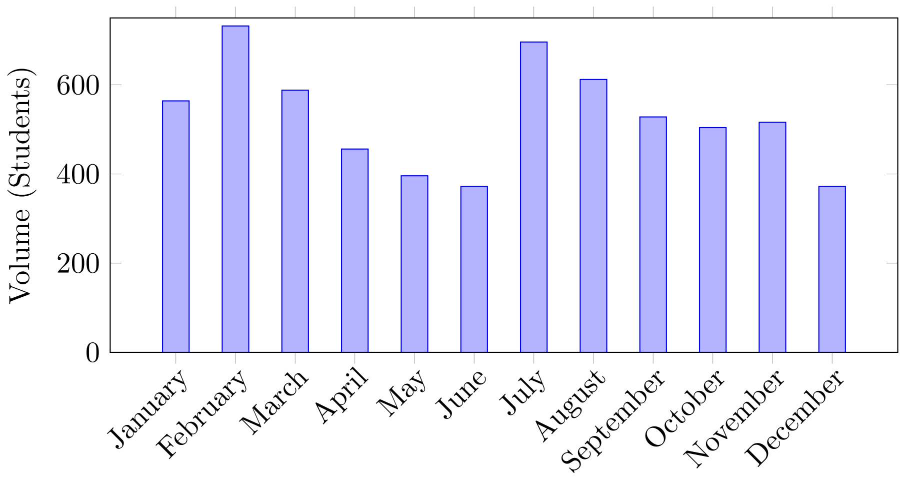


Figure 3.8: Typical yearly training volume at the Australian electricity distributor.

one to eight per location. There were 21 trainers in 11 of the 15 locations, and each trainer could teach between eight and 14 modules. There were eight working hours per day, split into half-hour periods. There were three rental windows per day: one in the morning, one in the afternoon, and one for the whole day. There was one demand window per month, spanning the entire duration of the month.

As a result of several factors, including new- and mid-year apprentice intakes, and from the summer and winter holiday seasons, the training volume fluctuates from month to month following a yearly pattern. Since most courses are valid for 12 months, students who complete a course in a particular month are likely to request the same course again around the same time the following year. Figure 3.8 shows typical training volume by month. February—the busiest month—is usually about twice as busy as June or December—the least busy months.

We generated a series of 48 test cases with similar properties to the supplied data. All test cases and solution files can be downloaded from `https://goo.gl/zUWYV2`. The test cases had planning horizons ranging between two and five weeks, between two and five regions each with two locations per region, and between two and four rooms per location. The courses were generated to be between one and five days long, with up to 15 modules per course. Each of the test cases target one of three timetable densities: Low (`L`), Medium (`M`), and High (`H`). The course demand values in the `L`, `M`, and `H` test cases were selected such that, if all classes are run at 80% of their student capacity, the resulting timetable will be about $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{2}{3}$ full, respectively, where a full timetable is one where every room is occupied at every time slot. For brevity, we refer to each test case by their target timetable density, number of days, and number of regions. For example, (`L`|10|2) is the smallest test case with low target timetable density, 10 days and 2 regions, whereas (`H`|25|5) is the largest test case with high target timetable density, 25 days and 5 regions.

The smallest test case, (`L`|10|2), had 13,261 variables and 18,565 constraints for the complete

Test Case	Cols	Rows	Test Case	Cols	Rows	Test Case	Cols	Rows
(L|10|2)	13261	18565	(M|10|2)	14427	20730	(H|10|2)	16267	23869
(L|10|3)	29215	31393	(M|10|3)	33099	37108	(H|10|3)	36605	42592
(L|10|4)	56253	49907	(M|10|4)	61749	57005	(H|10|4)	69639	66068
(L|10|5)	83383	64823	(M|10|5)	93693	77022	(H|10|5)	106155	91076
(L|15|2)	23246	36118	(M|15|2)	26284	41816	(H|15|2)	29976	48737
(L|15|3)	52356	58944	(M|15|3)	60013	70093	(H|15|3)	70050	84574
(L|15|4)	103785	93173	(M|15|4)	121753	113339	(H|15|4)	134357	127404
(L|15|5)	147920	124199	(M|15|5)	178637	159168	(H|15|5)	200594	185017
(L|20|2)	35802	56074	(M|20|2)	41623	66586	(H|20|2)	47713	78115
(L|20|3)	83827	100481	(M|20|3)	96509	120324	(H|20|3)	115221	148142
(L|20|4)	153137	150021	(M|20|4)	178934	180809	(H|20|4)	206623	214455
(L|20|5)	253259	223389	(M|20|5)	299697	275786	(H|20|5)	347315	327884
(L|25|2)	52191	82204	(M|25|2)	60888	97115	(H|25|2)	72637	117327
(L|25|3)	121575	151682	(M|25|3)	149025	193513	(H|25|3)	174001	229970
(L|25|4)	225070	226831	(M|25|4)	273907	282633	(H|25|4)	321000	338276
(L|25|5)	352849	322377	(M|25|5)	421138	399011	(H|25|5)	507011	493472

Table 3.1: The number of variables (Cols) and constraints (Rows) for the timetabling IP model for each test case.

timetabling subproblem, and the largest test case (H|25|5) had $507,011$ variables and $493,472$ constraints. The number of variables (columns) and constraints (rows) for each of the test cases can be seen in Table 3.1.

We used IBM ILOG CPLEX 12.5.0.0 [78] 64-bit on an Intel i7-4790K quad-core 4.00Ghz system with 16GB of RAM, running Windows 7 Professional. Our code was written in C#, and interacted with CPLEX using the IBM ILOG Concert API. We used default CPLEX settings, except we increased the maximum allowed memory usage to the total amount of free physical memory.

We chose the weights in the class timetabling objective function to be $\alpha_1 = 10^6$, $\alpha_2 = 5$, and $\alpha_3 = 1$, based on empirical testing and the distributor's inspection of the produced timetables. For the rostering objective, we gave each km travelled a weighting of 1, and each trainer swap an objective weighting of 5 in the objective function.

The time the algorithm spent in Stage 1 (construction) and Stage 2 (improvement) is shown in Table 3.2. During timetable construction in the first stage, no time limit is imposed. During Stage 2, there are two termination criteria: the LNS algorithm terminates if it is unable to improve the solution

Test Case	S1	S2	Test Case	S1	S2	Test Case	S1	S2
(L|10|2)	13	12	(M|10|2)	22	12	(H|10|2)	43	13
(L|10|3)	44	28	(M|10|3)	103	27	(H|10|3)	168	9550
(L|10|4)	195	11634	(M|10|4)	287	12227	(H|10|4)	481	13075
(L|10|5)	420	14549	(M|10|5)	571	15660	(H|10|5)	1015	16999
(L|15|2)	45	25	(M|15|2)	102	31	(H|15|2)	167	34
(L|15|3)	146	62	(M|15|3)	239	86	(H|15|3)	554	83
(L|15|4)	565	19226	(M|15|4)	1116	18734	(H|15|4)	1714	20107
(L|15|5)	1146	21492	(M|15|5)	2347	24786	(H|15|5)	3986	27158
(L|20|2)	103	49	(M|20|2)	158	282	(H|20|2)	339	66
(L|20|3)	443	129	(M|20|3)	589	165	(H|20|3)	3779	18129
(L|20|4)	1413	820	(M|20|4)	3964	28955	(H|20|4)	6377	28961
(L|20|5)	6084	36161	(M|20|5)	8558	42273	(H|20|5)	19297	71159
(L|25|2)	232	94	(M|25|2)	331	109	(H|25|2)	805	107
(L|25|3)	888	273	(M|25|3)	1916	907	(H|25|3)	7410	24196
(L|25|4)	3006	1492	(M|25|4)	7962	38773	(H|25|4)	17354	43881
(L|25|5)	10768	48409	(M|25|5)	12577	54548	(H|25|5)	43749	60956

Table 3.2: The amount of time, in seconds, the algorithm spent in Stage 1 (S1) and Stage 2 (S2) for each test case.

Test Case	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$
(L|10|2)	0	-8131	42	0	-9455	42
(L|10|3)	0	-11773	54	0	-13991	56
(L|10|4)	9	-14398	105	9	-16184	92
(L|10|5)	3	-15568	112	3	-21403	111
(L|15|2)	0	-11327	65	0	-13632	66
(L|15|3)	0	-16389	98	0	-22053	87
(L|15|4)	0	-21551	146	0	-25996	156
(L|15|5)	34	-27153	156	12	-24446	166
(L|20|2)	0	-15804	91	0	-20507	89
(L|20|3)	0	-22585	139	0	-28911	125
(L|20|4)	0	-34912	160	0	-40988	158
(L|20|5)	0	-43984	257	0	-45091	251
(L|25|2)	0	-16621	115	0	-20060	106
(L|25|3)	0	-33200	135	0	-39603	143
(L|25|4)	0	-38811	250	0	-54119	207
(L|25|5)	0	-50316	274	0	-62890	277

Table 3.3: The objective value components for stages 1 and 2 for the test cases with low target timetable density.

after a number of iterations or if a time limit is reached. The time limit was chosen to be $1000 \cdot |D| \cdot |\Xi| \cdot \kappa$ seconds and the number of iterations at which to terminate if no improvement can be made was chosen to be $10 \cdot |D| \cdot |\Xi| \cdot \kappa$, where, for a given test case, D is the set of included days (as defined in Section 3.3), Ξ is the set of included regions (as defined in Section 3.3), and κ is $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{2}{3}$ for test cases with low, medium, and high target density, respectively. These values were determined experimentally. Due to the time required to solve each individual IP model, the total time sometimes exceeds the time limit slightly.

The solution quality results for Stage 1 (construction) and Stage 2 (improvement) for the test cases with low target timetable density are given in Table 3.3. Stage 1 was able to construct timetables that satisfied all demand for 13 of the 16 cases. Of the three cases where not all demand could be satisfied, Stage 2 was only able to improve one of them with respect to the number of unsatisfied students, bringing the number of unsatisfied students from 34 down to 12 for that test case. Due to the room compatibility, course spreading, resource movement, and trainer capacity constraints, not all test cases have feasible timetables where all student demand can be satisfied. Stage 2 was able to

Test Case	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$
(M|10|2)	0	-6477	54	0	-7682	49
(M|10|3)	0	-6715	83	0	-10070	72
(M|10|4)	14	-9884	122	14	-12894	116
(M|10|5)	5	-11353	140	4	-13897	114
(M|15|2)	0	-6937	84	0	-9989	93
(M|15|3)	0	-16667	89	0	-20669	90
(M|15|4)	35	-17520	168	15	-22039	173
(M|15|5)	18	-21380	182	18	-22456	174
(M|20|2)	0	-15756	89	0	-19439	91
(M|20|3)	0	-16910	163	0	-27363	129
(M|20|4)	6	-23372	218	6	-24548	208
(M|20|5)	8	-33400	312	0	-35073	298
(M|25|2)	0	-15625	121	0	-18673	111
(M|25|3)	0	-21981	183	0	-28373	185
(M|25|4)	2	-37006	295	2	-38862	281
(M|25|5)	4	-53925	318	4	-56626	303

Table 3.4: The objective value components for stages 1 and 2 for the test cases with medium target timetable density.

improve the rental revenue objective by about 18% on average, with a maximum improvement of about 39%. Stage 2 was able to reduce the number of room swaps by about 4 swaps per test case on average with a maximum improvement of 43 eliminated room swaps, or about 3% on average with a maximum improvement of 20%.

The solution quality results for Stage 1 (construction) and Stage 2 (improvement) for the test cases with medium target timetable density are given in Table 3.4. Stage 1 was able to construct timetables that satisfied all demand for 8 of the 16 cases. Of the eight cases where not all demand could be satisfied, Stage 2 was able to improve three of them with respect to the number of unsatisfied students, bringing the total number of unsatisfied students from 48 down to 19 for those three cases. Stage 2 was able to improve the rental revenue objective by about 23% on average, with a maximum improvement of about 62%. Stage 2 was able to reduce the number of room swaps by about 8 swaps per test case on average, with a maximum improvement of 34 eliminated room swaps, or about 6% on average with a maximum improvement of 26%.

The solution quality results for Stage 1 (construction) and Stage 2 (improvement) for the test cases

Test Case	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$
(H|10|2)	0	-4516	64	0	-5939	64
(H|10|3)	8	-5700	94	2	-8110	81
(H|10|4)	29	-8914	124	17	-10856	126
(H|10|5)	34	-9460	155	5	-10587	141
(H|15|2)	0	-5539	93	0	-9338	98
(H|15|3)	0	-7822	131	0	-15269	127
(H|15|4)	49	-20583	191	49	-21617	182
(H|15|5)	40	-20922	233	40	-21976	222
(H|20|2)	0	-8298	136	0	-12972	125
(H|20|3)	0	-9299	209	0	-18822	190
(H|20|4)	23	-19805	254	23	-20818	242
(H|20|5)	0	-21448	314	0	-21448	314
(H|25|2)	0	-8529	161	0	-13369	168
(H|25|3)	0	-17871	210	0	-24004	231
(H|25|4)	0	-29399	350	0	-30873	334
(H|25|5)	26	-36185	393	26	-37999	375

Table 3.5: The objective value components for stages 1 and 2 for the test cases with high target timetable density.

Test Case	S3	Test Case	S3	Test Case	S3
(L|10|2)	0.1	(M|10|2)	0.1	(H|10|2)	0.3
(L|10|3)	0.2	(M|10|3)	0.3	(H|10|3)	0.5
(L|10|4)	6.2	(M|10|4)	8.6	(H|10|4)	10
(L|10|5)	18	(M|10|5)	16	(H|10|5)	21
(L|15|2)	0.1	(M|15|2)	0.2	(H|15|2)	0.4
(L|15|3)	0.4	(M|15|3)	0.5	(H|15|3)	0.5
(L|15|4)	25	(M|15|4)	27	(H|15|4)	27
(L|15|5)	22	(M|15|5)	31	(H|15|5)	38
(L|20|2)	0.3	(M|20|2)	1.6	(H|20|2)	0.3
(L|20|3)	0.4	(M|20|3)	0.5	(H|20|3)	4.7
(L|20|4)	10	(M|20|4)	12	(H|20|4)	15
(L|20|5)	25	(M|20|5)	29	(H|20|5)	41
(L|25|2)	0.4	(M|25|2)	0.5	(H|25|2)	0.6
(L|25|3)	0.4	(M|25|3)	4.3	(H|25|3)	11
(L|25|4)	10	(M|25|4)	17	(H|25|4)	22
(L|25|5)	29	(M|25|5)	33	(H|25|5)	39

Table 3.6: The amount of time, in seconds, the algorithm spent in Stage 3 (S3) for each test case.

with high target timetable density are given in Table 3.5. Stage 1 was able to construct timetables that satisfied all demand for 9 of the 16 cases. Of the seven cases where not all demand could be satisfied, Stage 2 was able to improve three of them with respect to the number of unsatisfied students, bringing the total number of unsatisfied students from 71 down to 24 for those three cases. Stage 2 was able to improve the rental revenue objective by about 34% on average, with a maximum improvement of about 202%. Stage 2 was able to reduce the number of room swaps by about 6 swaps per test case on average, with a maximum improvement of 19 eliminated room swaps, or about 3% on average with a maximum improvement of 16%.

The amount of time, in seconds, the algorithm spent in Stage 3 (rostering) for each test case is shown in Table 3.6. The test case that required the longest time to produce an optimal roster required only 41 seconds, and the largest test case required only 39 seconds to produce an optimal roster.

The number of variables (columns), and constraints (rows) for the rostering IP model for each test case, subject to the final generated timetable is given in Table 3.7. The rostering IP model—subject to the final generated timetable—has, on average, about 13% as many variables and about 9% as many constraints as the complete timetabling IP model for problem described the test case. Moreover, the

Test Case	Cols	Rows	Test Case	Cols	Rows	Test Case	Cols	Rows
(L|10|2)	2533	2085	(M|10|2)	2756	2287	(H|10|2)	3495	2854
(L|10|3)	4154	3238	(M|10|3)	4998	3848	(H|10|3)	5862	4538
(L|10|4)	10086	8015	(M|10|4)	12159	9464	(H|10|4)	13337	10285
(L|10|5)	13284	9945	(M|10|5)	14941	11187	(H|10|5)	17969	13391
(L|15|2)	3385	2706	(M|15|2)	4559	3603	(H|15|2)	4808	3786
(L|15|3)	6513	5107	(M|15|3)	6766	5264	(H|15|3)	9093	7064
(L|15|4)	17939	13850	(M|15|4)	19070	14675	(H|15|4)	19283	14743
(L|15|5)	20955	15562	(M|15|5)	22425	16507	(H|15|5)	27700	20106
(L|20|2)	4874	3902	(M|20|2)	4929	3925	(H|20|2)	9771	5352
(L|20|3)	9164	6996	(M|20|3)	9478	7207	(H|20|3)	13400	10000
(L|20|4)	18323	13831	(M|20|4)	24231	18227	(H|20|4)	27456	20529
(L|20|5)	32205	23638	(M|20|5)	37657	27487	(H|20|5)	40842	29757
(L|25|2)	5983	4811	(M|25|2)	6164	4948	(H|25|2)	9101	7200
(L|25|3)	10178	7619	(M|25|3)	13269	9852	(H|25|3)	16155	11881
(L|25|4)	23550	17618	(M|25|4)	31560	23417	(H|25|4)	37222	27483
(L|25|5)	34816	25222	(M|25|5)	37397	26970	(H|25|5)	46211	33073

Table 3.7: The number of variables (Cols) and constraints (Rows) for the roster IP model, subject to the final generated timetable, for each test case.

Test Case	Travel	Swaps	Test Case	Travel	Swaps	Test Case	Travel	Swaps
(L|10|2)	209.7	1	(M|10|2)	0.0	0	(H|10|2)	419.3	1
(L|10|3)	1045.7	1	(M|10|3)	378.6	0	(H|10|3)	1723.1	1
(L|10|4)	233.0	0	(M|10|4)	490.6	0	(H|10|4)	388.3	1
(L|10|5)	322.3	1	(M|10|5)	177.6	0	(H|10|5)	391.2	2
(L|15|2)	1512.6	1	(M|15|2)	1629.0	1	(H|15|2)	1803.5	1
(L|15|3)	266.1	0	(M|15|3)	105.6	0	(H|15|3)	853.1	1
(L|15|4)	1505.6	0	(M|15|4)	1887.1	1	(H|15|4)	1887.1	1
(L|15|5)	3343.1	0	(M|15|5)	4193.0	0	(H|15|5)	4172.1	0
(L|20|2)	20.1	1	(M|20|2)	20.1	1	(H|20|2)	20.1	1
(L|20|3)	0.0	0	(M|20|3)	0.0	0	(H|20|3)	0.0	0
(L|20|4)	22.3	2	(M|20|4)	0.0	1	(H|20|4)	0.0	1
(L|20|5)	269.2	1	(M|20|5)	669.1	0	(H|20|5)	1056.2	0
(L|25|2)	1569.2	7	(M|25|2)	1024.9	3	(H|25|2)	3546.1	4
(L|25|3)	0.0	1	(M|25|3)	0.0	0	(H|25|3)	0.0	0
(L|25|4)	1399.4	1	(M|25|4)	2896.9	1	(H|25|4)	3665.0	1
(L|25|5)	0.0	1	(M|25|5)	0.0	4	(H|25|5)	50.9	3

Table 3.8: The total trainer travel distance and number of trainer swaps for each roster produced.

complete timetabling IP model has about 16% more constraints than variables, whereas the rostering IP model has about 23% fewer constraints than variables. These substantial differences together with the dominant network flow structure of the rostering model may explain, in part, why these rostering problem instances are computationally tractable even for the largest of the test cases, while the timetabling problem instances remain intractable for all but the smallest test cases.

The total distance travelled by trainers, in kilometres, and the total number of trainer swaps present in each of the produced rosters is shown in Table 3.8. Rosters which had zero trainer travel and swaps were found in 23 of the 48 test cases. There was an average of about 18km travelled and about 0.02 trainer swaps per location per day, which is consistent with the distributor's previous training rosters.

CPLEX was able to find optimal solutions to the full timetabling models described in Section 3.3 and rostering models described in Section 3.4 to 16 of the 48 test cases, and the results are compared with those from the three-stage heuristic in Table 3.9. The columns from left to right are the test case name, the optimal values for Z_1, Z_2, and Z_3 in the timetabling subproblem, the optimal values for trainer travel distance (DT) and trainer swaps (TS) in the rostering subproblem subject to the optimal

TestCase	Z1	Z2	Z3	DT	TS	Time	T-Ratio	TT-Opt	TT-Heur	R-Opt	R-Heur
(L|10|2)	0	-9455	42	209.7	1	48	0.52	-47233	-47233	1049.3	1049.3
(L|10|3)	0	-13991	56	1045.7	1	98	0.73	-69899	-69899	5229.4	5229.4
(L|15|2)	0	-13660	67	1454.5	1	171	0.41	-68233	-68094	7273.3	7564.2
(L|15|3)	0	-22053	87	371.7	0	597	0.35	-110178	-110178	1858.5	1330.4
(L|20|2)	0	-20576	89	20.1	1	1460	0.10	-102791	-102446	101.7	101.7
(L|25|2)	0	-20060	106	1390.3	5	8173	0.04	-100194	-100194	6956.3	7852.8
(M|10|2)	0	-7682	48	0.0	1	862	0.04	-38362	-38361	1.0	0.0
(M|10|3)	0	-10070	72	677.4	0	349	0.37	-50278	-50278	3387.0	1893.1
(M|15|2)	0	-10265	90	1745.3	1	1974	0.07	-51235	-49852	8727.7	8146.0
(M|15|3)	0	-20669	90	0.0	0	2865	0.11	-103255	-103255	0.0	528.1
(M|20|2)	0	-19439	90	20.1	1	1594	0.28	-97105	-97104	101.7	101.7
(H|10|2)	0	-5939	64	629.0	0	177	0.32	-29631	-29631	3145.0	2097.7
(H|10|3)	2	-8416	82	1643.2	1	35168	0.28	1958002	1959531	8217.0	8616.4
(H|15|2)	0	-9373	99	1745.3	1	909	0.22	-46766	-46592	8727.7	9018.6
(H|15|3)	0	-15270	124	743.4	0	81161	0.01	-76226	-76218	3716.9	4266.4
(H|20|2)	0	-13041	127	20.1	1	41349	0.01	-65078	-64735	101.7	101.7

Table 3.9: Results related to the optimal solutions for some of the smaller test cases.

timetable, the time required to find the optimal solution (Time), in seconds, the ratio of total time required by our heuristic to total time required for CPLEX to find the optimal solution (T-Ratio), the optimal objective value for the timetabling problem (TT-Opt), the objective value for the timetabling subproblem our heuristic found (TT-Heur), and the optimal (R-Opt) and heuristic (R-Heur) values for the rostering subproblem, subject to the optimal and heuristic timetables, respectively. Our heuristic was able to find the optimal timetable for 7 of the 16 cases.

Since the rostering subproblem is based on the produced timetable, different timetables yield a different sets of feasible rosters with different optimal objective values. Compared with the optimal roster, subject to the sub-optimal timetable produced by the heuristic approach, the optimal roster, subject to the optimal timetable, had a worse objective value in five cases, had the same objective value in five cases, and a better objective value in six test cases. This suggests that in this case, in the absence of an integrated model—one that unifies the timetabling and rostering subproblems into a single IP model—it may not be a good strategy to invest excessive computational effort in producing the best possible solution for each subproblem.

The electricity distributor does frequently engage in long-term strategic planning, where the purchase and sale of organisational resources must be investigated, specific trainer specialisations must be determined, and the structure of courses must be adjusted to meet the changing needs of the organisation and the requirements of changing safety laws. For long-term strategic planning purposes, having high quality timetables and rosters is of paramount importance to the organisation and the time requirements of the three-stage heuristic are within acceptable limits. It is a matter of ongoing research to further improve the process and reduce total computation time where possible.

An implementation of the Demand-Based Course Timetabling solution approach discussed in this chapter was trialled at the Australian electricity distributor. After a period of monitoring and analysis, it was reported by senior management that the automated timetabling solution, by producing efficient timetables and freeing a number of staff from timetabling-related duties, carries a recurring saving of about AUD$270,000 (2013). Moreover, the once difficult matter of making timetable changes when unexpected events occur has been reported to be far less daunting. At the time of writing, the automated timetabling system remains in use at the organisation.

3.7 Conclusions

In this chapter we studied a timetabling and rostering problem involving periodic retraining of large numbers of employees at an Australian electricity distributor. We developed an Integer Programming model to solve the timetabling of classes, and an IP model to solve the rostering of trainers to an existing class timetable. Both models were developed so that they can deal with all the practical requirements in a flexible manner, given the changing nature of the organisation and industry laws.

Due to the size of the problem given real world data, it was not possible to solve to optimality in practically acceptable time. A three-stage heuristic framework has been presented, which consists of an initial timetable generation stage, an iterative improvement stage, and a trainer rostering stage. All three stages utilise the IP models that were developed. The computational results show that the proposed approach is able to generate solutions for the problem sizes typical for training at the distributor, and that the approach is effective for both operational and strategic planning purposes. The proposed fix-and-optimise Large Neighbourhood Search algorithm is easily generalisable to many other class timetabling problems from other institutions.

In the few cases where we were able to solve the timetabling subproblem to optimality, the optimal rosters occasionally yielded poorer solutions compared with the optimal rosters to timetables produced by our heuristic. Even for different timetables with identical objective values, the optimal rosters can have very different objective values. This reinforces that solving each subproblem to optimality does not always lead to a global optimum, which suggests an integrated model is worth investigating.

Future research can investigate different modelling techniques and related solution approaches to further reduce the required computation time. It can be investigated whether replacement of the first stage with a different heuristic or metaheuristic approach has a significant impact on solution time or quality. Others can adapt the introduced techniques to other problems and determine whether they are successful in other problem domains.

A real-word implementation of the proposed solution approach was very successful, with annual reported savings of AUD$270,000.

Chapter 4

Formation and Sequencing of Classes for a Bottleneck Classroom

Contents

The research in this chapter was presented at the Australian Society for Operations Research (ASOR) Recent Advances 2015 conference [150], where it was awarded the "Best Student Presentation" prize and AU$250. The research was subsequently published as "O. G. Czibula, H. Gu, F.-J. Hwang, M. Y. Kovalyov, and Y. Zinder. Bi-criteria sequencing of courses and formation of classes for a bottleneck classroom. *Computers & Operations Research*, 65:53–63, 2016" [40].

Abstract

In this chapter, the problem of class formation and sequencing for multiple courses subject to a bottleneck classroom with an ordered bi-criteria objective is studied. The problem can be modelled as a single-machine batch scheduling problem with incompatible job families (where jobs are partitioned into families, and only jobs from the same family can processed simultaneously on a single machine), and parallel job processing in batches, where the batch size is family-dependent. For the minimisation of the number of tardy jobs, the strong NP-hardness is proven. For the performance measure of the maximum cost, we consider single criterion and bi-criteria cases. We present an $O(n^2)$ algorithm—n is the number of jobs—for both cases. An Integer Programming model as well as Simulated Annealing and Genetic Algorithm matheuristics to solve a fairly general case of the bi-criteria problem are presented and computationally tested.

4.1 Introduction

This study addresses a scheduling problem that is inspired by the problem of scheduling training courses for trainees working for large organisations with scarce teaching resources. One such example is Australia's largest electricity distributor by number of connections and number of employees, which is required by Australian law under the Work Health and Safety Act 2011: Part 2, Division 19, to deliver regular safety and technical training to all its employees who work on or near the electricity network. The problem of scheduling thousands of workers (students), hundreds of classes of multiple courses with dedicated trainers and classrooms across a small number of training facilities is highly complex in practice. It is useful, however, to develop efficient algorithms for solving smaller or special cases as it provides insight into the underlying problem.

In contrast to the research in Chapter 3, this chapter investigates the case of a single classroom with a limited capacity, primarily from an analytical viewpoint. In the studied problem, there are a set of students and a set of courses. Each student is required to undertake one or more courses by certain due dates. It is assumed that we have a single classroom where multiple students can be taught simultaneously, provided they are all undertaking the same course. The classroom retains a limited capacity, and each course is characterised by a maximum class size in students, in accordance with its distinct training content. Without loss of generality, the maximum class size is no greater than the classroom capacity. The class duration is also course-dependent. The classroom is not allowed to be used for more than one class at any time. The objective is to determine an optimal formation and sequence of classes for courses and students so as to minimise the considered performance metrics. As with many industrial problems, it is often difficult to quantify the needs of the organisation with a single objective function. We investigate the case where two ordered objectives—a primary and a secondary objective—are considered. An optimal schedule is one that minimises the secondary objective over the set of schedules that minimise the primary objective.

The studied course scheduling problem with a bottleneck classroom can be modelled as a single-machine batch scheduling problem with incompatible job families, where the batch size is family-dependent. Before providing the comprehensive mapping between these two problems, we describe the single machine batch scheduling problem with incompatible job families. There is a set of n jobs, which are categorised into m distinct job families. The number of jobs in family f is denoted by n_f, and $\sum_{f=1}^{m} n_f = n$. Here and below, we assume that each summation and maximum is taken over all families or jobs, unless otherwise specified. The job j of family f is denoted by (f, j), $f \in \{1, \dots, m\}$ and $j \in \{1, \dots, n_f\}$. The jobs belonging to the same job family f have a common processing time p_f. Each job (f, j) is also characterised by a due date $d_{f,j}$, a weight $w_{f,j}$ and two non-decreasing cost functions $g_{f,j}(C_{f,j})$ and $h_{f,j}(C_{f,j})$, where $C_{f,j}$ is the completion time of the job. The batching machine, where the parallel-batch mode is applied, can process simultaneously multiple jobs belonging to the same family as a batch [16]. The maximum number of jobs allowed to be processed concurrently, i.e. the batch size, which is clearly no greater than the machine capacity, depends on the family of those jobs inside the batch and is denoted by b_f. The processing time of a batch belonging to family f does not depend on the number of jobs in the batch and is always p_f. There is no machine idle time between the batches, otherwise a schedule is not completely determined. A schedule is defined to be the sequence and composition of all batches. The objective is to find a batch schedule for jobs such that the considered performance measures, dependent on job completion times, are minimised. As with the Class Timetabling subproblem discussed in Chapter 3, this problem is closely related to Lexicographic Optimisation [81, 163], a special case of Goal Programming [99], which is a branch of Multiobjective Optimisation.

The mapping is given as follows. The bottleneck classroom is represented as a single batching machine. Each course maps to a job family, and each course requirement for a student is represented by a job. This means that if a student is required to complete, for example, three different courses, these three requirements will be represented by three jobs which belong to three distinct families. The due date imposed upon some course requirement for some student is exactly the due date of the corresponding job. The course-dependent class duration is represented as the family-dependent job processing time. Then a class of students formed for some course corresponds to a batch of jobs for some family. The maximum class size for some course is represented as the batch size for the corresponding job family. Hereafter, the studied problem will be described as the single-machine batching scheduling problem with incompatible job families subject to family-dependent batch sizes.

The remainder of this chapter is organised as follows. Section 4.2 provides analytical results of the considered problem, including a proof of NP-hardness in the strong sense for the number of tardy jobs criterion, a polynomial time algorithm for the maximal primary and maximal secondary criterion case, and a polynomial algorithm for the case with a given sequence of batches. Section 4.3 discusses computational approaches to solving the considered problem, including Integer Programming (IP) approaches for exact solutions, and Simulated Annealing (SA) and Genetic Algorithm (GA) approaches for heuristic solutions. Section 4.5 contains concluding remarks.

4.2 Analytical Results

In this section we discuss the complexity results for four ordered bi-criteria cases and the polynomial solvability of one of the cases under the assumption that the sequence of batches is fixed a priori. We use the notation (f_1, f_2) to represent the ordered relationship between two objectives in the ordered bi-criteria problem. For example, $(\sum g_{f,j}, \sum h_{f,j})$ is an ordered bi-criteria objective whose optimal solution minimises the secondary criterion $\sum h_{f,j}$ over the set of solutions that minimise the primary criterion $\sum g_{f,j}$. Note that the objective functions $\sum h_{f,j}$, $\sum g_{f,j}$, $\max h_{f,j}$ and $\max g_{f,j}$ are *regular*, that is, they are non-decreasing in each argument.

4.2.1 Complexity of ordered bi-criteria cases

There are four cases of the bi-criteria problem with the sum and maximum objectives depending on their type and order. We will first strengthen the result of Jolai [85] and show that the problem of minimising the number of tardy jobs, $\sum U_{f,j}$, is NP-hard in the strong sense. Given a batch schedule, function $U_{f,j}(t)$ is defined such that $U_{f,j}(t) = 0$ if $t \leq d_{f,j}$ (for $t = C_{f,j}$ job (f, j) is on-time) and $U_{f,j}(t) = 1$ if $t > d_{f,j}$ (for $t = C_{f,j}$ job (f, j) is tardy).

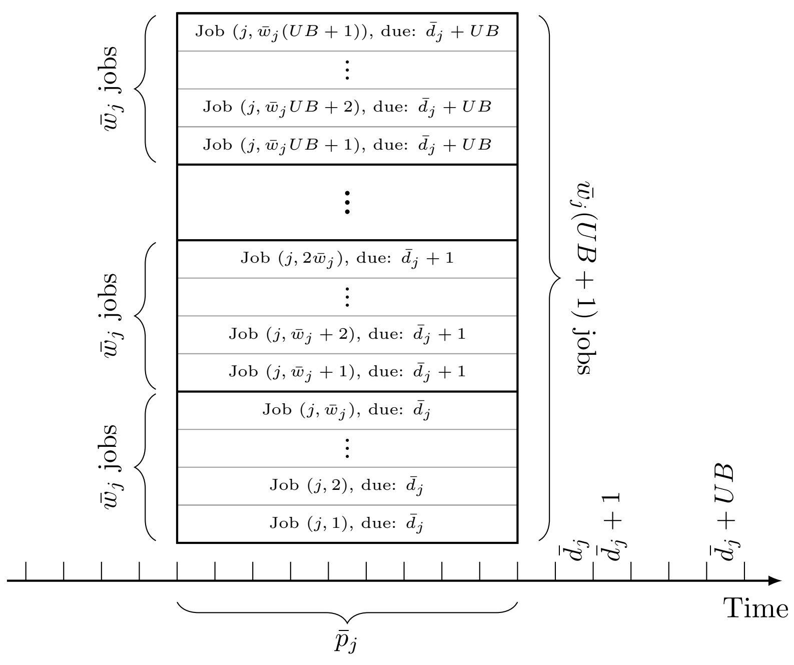
Theorem 1. *The problem of minimising $\sum U_{f,j}$ is NP-hard in the strong sense irrespective of whether the batch sizes are bounded or unbounded.*

Proof. We use a reduction from the decision version of the NP-hard in the strong sense problem of minimising total weighted tardiness on a single machine, discussed in Lawler [97]. Given an instance A of the single-machine total weighted tardiness minimisation problem, we can construct an instance B of the studied problem in pseudo-polynomial time in the following way. For instance A, we let $\bar{n}$ be the number of jobs, and denote by $\bar{p}_j$, $\bar{d}_j$, and $\bar{w}_j$ the processing time, due date, and weight, respectively, for job $j = 1, \ldots, \bar{n}$. For each job j in instance A, construct a corresponding family j consisting of $\bar{w}_j$ jobs with due date $\bar{d}_j$, $\bar{w}_j$ jobs with due date $\bar{d}_j + 1$, and so on until $\bar{w}_j$ jobs with due date $\bar{d}_j + UB$, where the upper bound $UB = \sum_{j=1}^{\bar{n}} \bar{p}_j$, for instance B. This gives instance B a total of $\bar{w}_j(UB + 1)$ jobs for each family j, whose job processing time is $\bar{p}_j$, $j = 1, \ldots, \bar{n}$. The batch size for each family is assumed to be sufficiently large such that all jobs of each family can be processed simultaneously in one batch.

Consider the decision problem of determining whether a schedule with the total weighted tardiness no greater than some positive integer κ exists for instance A. If such a schedule exists for instance A, then a schedule whose number of tardy jobs is no greater than κ for instance B also exists. In the corresponding schedule of instance B, all jobs of the same family are processed in one batch, and the batch sequence, i.e. the family sequence, of instance B is identical to the job sequence of instance A. See Figure 4.1 for an illustration.

Conversely, if a batch schedule with the number of tardy jobs no greater than κ exists for instance B, then it can be converted into a schedule of the same or better quality, in which all jobs of the same family are assigned in the same batch. Then the sequence of jobs for instance A defined by the sequence of families in the latter batch schedule has the total weighted tardiness no greater than κ. □

Composition of Family j in B:



Job $(j, \bar{w}_j(UB+1))$, due: $\bar{d}_j + UB$
Job $(j, \bar{w}_j UB + 2)$, due: $\bar{d}_j + UB$
Job $(j, \bar{w}_j UB + 1)$, due: $\bar{d}_j + UB$
Job $(j, 2\bar{w}_j)$, due: $\bar{d}_j + 1$
Job $(j, \bar{w}_j + 2)$, due: $\bar{d}_j + 1$
Job $(j, \bar{w}_j + 1)$, due: $\bar{d}_j + 1$
Job $(j, \bar{w}_j)$, due: $\bar{d}_j$
Job $(j, 2)$, due: $\bar{d}_j$
Job $(j, 1)$, due: $\bar{d}_j$
$\bar{w}_j$ jobs
$\bar{w}_j$ jobs
$\bar{w}_j$ jobs
$\bar{w}_j(UB+1)$ jobs
$\bar{d}_j$
$\bar{d}_j + 1$
$\bar{d}_j + UB$
Time
$\bar{p}_j$

Figure 4.1: The composition of the family j in B, corresponding to job j in A.

Theorem 1 showed that there exists a regular objective $\sum g_{f,j} = \sum U_{f,j}$ that is NP-hard in the strong sense, therefore, without knowing more about g, the regular objective $\sum g_{f,j}$ in general is also NP-hard in the strong sense. For the ordered bi-criteria cases, the *result* with respect to the primary criterion could affect the *computation* with respect to the secondary criterion (as will be shown in Section 4.2.2), but not the other way around. Therefore, the ordered bi-criteria objective with primary criterion $\sum g_{f,j}$ must have computational complexity at least as hard as the computational complexity of the single criterion $\sum g_{f,j}$ problem. This is essentially because the problem must first be solved for the primary criterion, and then for the secondary criterion with respect to the solution to the primary criterion. We can then conclude that based on the result in Theorem 1, the first two ordered bi-criteria cases: $(\sum g_{f,j}, \sum h_{f,j})$ and $(\sum g_{f,j}, \max h_{f,j})$ are NP-hard in the strong sense in general for regular g and h, though there may exist particular g and h that are solvable in polynomial time. Hoogeveen [76] shows that the case $(\max g_{f,j}, \sum h_{f,j})$ is also NP-hard in the strong sense in general for regular g and h. The discussion in [76] is not specifically for a batch scheduling problem, however the results contained are applicable here also. Again, there may be particular g and f where $(\max g_{f,j}, \sum h_{f,j})$ is solvable in polynomial time.

The final case, $(\max g_{f,j}, \max h_{f,j})$, can be solved in polynomial time assuming that the functions are polynomially computable. We will give some lemmas and an algorithm as a proof. We start with single criterion case of minimising the maximum cost function $\max g_{f,j}$.

Proposition 1 in Uzsoy [154] implies that, for the single criterion problem with a regular performance measure, there exists an optimal schedule in which all batches are full, except possibly the last batch of each family. This last batch of family f includes $s_f = n_f - b_f(\lceil \frac{n_f}{b_f} \rceil - 1)$ jobs and each other batch of family f includes b_f jobs if $n_f > 0$. The makespan value, $C_{\max}$, is therefore given by

$$C_{\max} = \sum_f p_f \left\lceil \frac{n_f}{b_f} \right\rceil .$$

We will only consider schedules that meet these conditions.

Given an arbitrary schedule, let us introduce the family completion times $C^f_{\max} = \max_{j \in 1,\ldots,n_f} C_{f,j}$, $f = 1, \ldots, m$, which represents the latest completion time of any job in family f.

Lemma 1. *There exists an optimal schedule for the problem of minimising* $\max g_{f,j}$, *in which the last batch of each family f contains the jobs that retain the smallest s_f values of $g_{f,j}(C^f_{\max})$.*

Proof. Given a schedule, denote by $\mathtt{S}_f$ the set of s_f jobs which have the smallest s_f values of $g_{f,j}(C^f_{\max})$ for each family f. Consider an optimal schedule σ^* where the last batch of family f is comprised of the set of jobs $\mathtt{U}_f \cup \mathtt{V}_f$ where $\mathtt{U}_f \subset \mathtt{S}_f$ and $\mathtt{V}_f \not\subset \mathtt{S}_f$. Clearly, there exists a set of jobs $\mathtt{V}'_f$ scheduled elsewhere such that $\mathtt{U}_f \cup \mathtt{V}'_f = \mathtt{S}_f$ and $|\mathtt{V}'_f| = |\mathtt{V}_f|$. The positions of the jobs in $\mathtt{V}_f$ and $\mathtt{V}'_f$ can be swapped to produce a new schedule σ', in which the last batch of family f includes the set of all jobs in $\mathtt{S}_f$ and no others. Since the completion time of each job (f,j) in $\mathtt{V}_f$ is earlier in σ' and $g_{f,j}$ is non-decreasing, $g_{f,j}(C_{f,j})$ for each job $(f,j) \in \mathtt{V}_f$ must be no greater in σ' than in σ^*. The completion time of each job (f,j) in $\mathtt{V}'_f$ is later in σ' than in σ^*, however according to the definition of $\mathtt{V}_f$ and $\mathtt{V}'_f$, $g_{f,j}(C^f_{\max}) \geq g_{f,i}(C^f_{\max})$ for $(f,j) \in \mathtt{V}_f$ and $(f,i) \in \mathtt{V}'_f$ and therefore the objective value in σ' is no greater than σ^*, hence σ' is also optimal. The lemma follows. □

Lemma 1 implies that an optimal formation of the last batch of family f can be determined if $C^f_{\max}$ in an optimal schedule is known.

Given an instance, let J^{Last}_f denote the set of s_f jobs of family f retaining the smallest s_f values of $g_{f,j}(C_{\max})$. Define $G_f = \max_{(f,j) \in J^{Last}_f}\{g_{f,j}(C_{\max})\}$, $f = 1, \ldots, m$.

Lemma 2. *There exists an optimal schedule for the problem of minimising* $\max g_{f,j}$, *in which a batch comprising J^{Last}_h is sequenced last if* $G_h = \min_{1 \leq f \leq m}\{G_f\}$.

Proof. Consider an optimal schedule σ^*, in which a batch containing $J^{Last}_{f_1}$ of family f_1 is sequenced last, and for some family f_2 we have $G_{f_1} \geq G_{f_2}$. Denote in schedule σ^* the last batch of family f_2 by B_{f_2}, the partial schedule preceding to B_{f_2} as $\mathtt{A}$, and the partial schedule following B_{f_2} as $\mathtt{B}$. From σ^*, construct a new schedule σ' by removing B_{f_2} from the schedule, shifting $\mathtt{B}$ earlier such that $\mathtt{B}$ starts as soon as $\mathtt{A}$ finishes, and placing B_{f_2} at the end of the schedule. Each job (f,j) in $\mathtt{B}$ has an earlier completion time in σ' than in σ^*, therefore $g_{f,j}(C_{f,j})$ for each job (f,j) in $\mathtt{B}$ must be no greater in

σ' than in σ^*. After shifting B_{f_2} to the end of the schedule σ', the composition of jobs in B_{f_2} might need to be altered to satisfy Lemma 1 for schedule σ', i.e. B_{f_2} shall contain $J_{f_2}^{Last}$. Suppose that B_{f_2} currently comprises the set of jobs $\mathtt{U}_{f_2} \cup \mathtt{V}_{f_2}$ such that the jobs $\mathtt{U}_{f_2} \subset J_{f_2}^{Last}$ and $\mathtt{V}_{f_2} \not\subset J_{f_2}^{Last}$. The set of jobs $\mathtt{V}'_{f_2}$, such that $\mathtt{U}_{f_2} \cup \mathtt{V}'_{f_2} = J_{f_2}^{Last}$, can be found in the partial schedule $\mathtt{A}$. Swapping the position of jobs $\mathtt{V}_{f_2}$ and $\mathtt{V}'_{f_2}$ in σ' ensures B_{f_2} contains $J_{f_2}^{Last}$. Each job $(f,j) \in \mathtt{V}_{f_2}$ now has an earlier completion time and hence $g_{f,j}(C_{f,j})$ for $(f,j) \in V_{f_2}$ is no greater in σ' than in σ^*. Each job $(f,j) \in \mathtt{V}'_{f_2}$ now has a later completion time, however since $G_{f_1} \geq G_{f_2}$, the total cost of σ' must be no greater than σ^*, hence σ' is also optimal. The lemma follows. $\square$

Justified by Lemmas 1 and 2, we suggest the following optimal algorithm for the problem of minimising $\max g_{f,j}$. This algorithm is a modification of Lawler's algorithm [97].

Algorithm 1

Step 1. Initialise job sets $N_f = \{(f,1), \ldots, (f,n_f)\}$ for $f \in \{1, \ldots, m\}$. Determine $C_{\max} = \sum p_f \lceil \frac{n_f}{b_f} \rceil$. Set the objective value $g_{\max} = 0$. Calculate $s_f = n_f - b_f(\lceil \frac{n_f}{b_f} \rceil - 1), f = 1, \ldots, m$.

Step 2. Re-number jobs of each family f with $n_f > 0$ such that $g_{f,1}(C_{\max}) \leq g_{f,2}(C_{\max}) \leq \cdots \leq g_{f,n_f}(C_{\max})$, $f = 1, \ldots, m$. We have $G_f = g_{f,s_f}(C_{\max})$. Determine family f^* of the batch completing at $C_{\max}$ by calculating $f^* = \arg\min_{1 \leq f \leq m, n_f > 0} G_f$. Sequence the batch $B_{f^*} = \{(f^*,1), (f^*,2), \ldots, (f^*, s_{f^*})\}$ such that it completes at $C_{\max}$.

Step 3 Re-set $N_{f^*} := N_{f^*} \setminus B_{f^*}$, $n_{f^*} := n_{f^*} - s_{f^*}$, $s_{f^*} := b_{f^*}$, $g_{\max} := \max\{g_{\max}, G_{f^*}\}$, and $C_{\max} := C_{\max} - p_{f^*}$.

(i) If $C_{\max} = 0$, then terminate with an optimal schedule.

(ii) Otherwise, repeat Step 2.

The run time of Algorithm 1 can be easily evaluated. Step 2 requires $O(n)$ operations since re-sorting can be avoided by making use of the median-based technique for finding the kth smallest value [36], where $k = s_f$. After that, finding f^* needs $O(m)$ time. Step 3 requires $O(1)$ operations. Steps 2 and 3 are repeated at most n times. Thus, the total run time of Algorithm 1 is $O(n^2)$ since $m \leq n$.

The bi-criteria problem of minimising $\max h_{f,j}$ over the set of solutions that minimise $\max g_{f,j}$ can be converted into two single-criterion problems. First, Algorithm 1 should be run for the objective of minimising $\max g_{f,j}$ to find the optimal objective value G^* with respect to the primary criterion. For the secondary criterion of minimising $\max h_{f,j}$, introduce an auxiliary function $\hat{h}_{f,j}(t)$ for each job (f,j) that is equal to $h_{f,j}(t)$ if $g_{f,j}(t) \leq G^*$, or ∞ if $g_{f,j}(t) > G^*$. Next, the solution to the bi-criteria problem $(g_{f,j}, h_{f,j})$ can be found by solving the single-criterion problem of minimising $\max \hat{h}_{f,j}$.

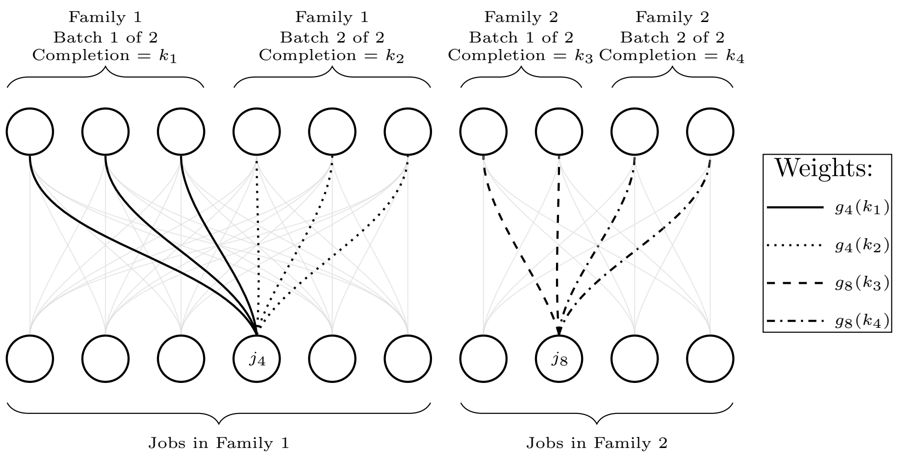


Figure 4.2: Assigning jobs to a fixed sequence of batches.

4.2.2 Fixed batch order

In the problem considered so far, the sequence of batches with respect to their family can be arbitrary, i.e., any batch that is sequenced ith can be a batch of jobs belonging to any family. We now consider the case where the sequence of batches is fixed, i.e., any batch sequenced ith must be a batch of jobs from a particular family, decided a priori. Then the problem simplifies to deciding which jobs will be assigned to which position in each batch on the schedule in order to optimise some performance measure. Assume that, in the fixed sequence, there are $r_f = \lceil \frac{n_f}{b_f} \rceil$ batches of family f, and each batch can include up to b_f jobs, $f = 1, \ldots, m$.

For a single objective $\sum g_{f,j}$ or $\max_{f,j} g_{f,j}$, which is not necessarily regular, it is possible to optimally assign jobs to the fixed sequence of batches in a way similar to that in Brucker et al. [16]. Consider a weighted bipartite graph where jobs form one set of vertices, and positions in the batches form another set of vertices. Edges exist between any job and any position in any batch of the same family, see Figure 4.2.

Since the completion time of each batch is known in advance, the cost of any edge between job (f, j) and position of the batch finishing at C can be set to $g_{f,j}(C)$. Then the problem boils down to selecting exactly one edge for each job so that the total or maximum cost is minimised.

Notice from Figure 4.2 that the sub-graphs of distinct families are disconnected. This implies that we can solve a smaller Linear Bottleneck Assignment Problem [18] (also known as the Bottleneck Bipartite Matching Problem [132], closely related to the Bipartite Matching Problem) for each family instead of solving one large problem for all families and then combine optimal solutions for families in a global optimal solution. Let the graph for one family include N_f vertices and M_f edges, where $N_f = n_f + b_f r_f$ and $M_f = n_f b_f r_f$.

In the case of the bottleneck objective $\max g_{f,j}$, the linear bottleneck assignment problem for the sub-graph corresponding to family f can be solved in $O(N_f\sqrt{N_f M_f})$ time by the algorithm of Punnen and Nair [132]. In the case of the sum objective $\sum g_{f,j}$, the linear bottleneck assignment problem problem can be solved in $O(N_f^2 \log N_f + N_f M_f) = O(N_f M_f)$ time by Dijkstra's algorithm [47] implemented with Fibonacci heaps as described in Fredman and Tarjan [62].

The described approach can be adapted for the ordered bi-criteria cases $(\max g_{f,j}, \max h_{f,j})$ and $(\max g_{f,j}, \sum h_{f,j})$, so that, once the optimal solution value f_1^* has been determined for the primary objective, every edge with the cost exceeding f_1^* is removed from the bipartite graph. The linear bottleneck assignment problem is then solved for the reduced graph. This approach, given its simplicity and polynomial time complexity, is highly conducive to heuristic approaches where the space of fixed batch orders is explored.

4.3 Computational Approaches

We consider the bi-criteria objective $(\max g_{f,j}, \sum h_{f,j})$ for our computational experiments, which, for g and h in general, was shown to be NP-hard in Hoogeveen [76]. We assume that $g_{f,j}$ and $h_{f,j}$ are regular and all convex piecewise linear. We present an IP approach complemented by Simulated Annealing and Genetic Algorithm metaheuristics, which are considered to be alternative solution approaches when IP is incapable to produce a solution in a desired time limit.

4.3.1 Integer Programming Approach

It is possible to formulate the ILP model for the studied batch scheduling problem in many ways. We have chosen to use *assignment and positional date variables* discussed in Keha [91], as that approach performed the best as the number of jobs increased for the single machine scheduling problem. Our own experimentation with the different formulations reflected their findings for the batch scheduling problem.

Using the assignment and positional date variables approach, we consider a set of ordered positions to which batches can be assigned. Each batch is paired up with a position on the timeline. The starting time of the batch assigned to position k is equal to the completion time of the batch assigned to position $k-1$.

Define variables $u_{f,i,k}$ to be 1 if batch indexed i of family f is assigned to position k on the timeline, or 0 otherwise. Define variables γ_k to be the completion time of the batch which is assigned to position k, and $D_{f,i}$ to be the completion time of batch indexed i of family f. Following from Proposition 1 in Uzsoy [154], the family f has $r_f = \lceil \frac{n_f}{b_f} \rceil$ batches in the optimal schedule, and clearly we require $r = \sum_f r_f$ positions in total.

It is convenient to write the ILP model in two parts: one to assign batches to positions and one to assign jobs to batches. The part of the ILP model that sequences the batches is as follows:

$$\sum_{k=1}^{r} u_{f,i,k} = 1 \qquad 1 \le f \le m,\ 1 \le i \le r_f \tag{4.1}$$

$$\sum_{f=1}^{m}\sum_{i=1}^{r_f} u_{f,i,k} = 1 \qquad 1 \le k \le r \tag{4.2}$$

$$\gamma_1 = \sum_{f=1}^{m}\sum_{i=1}^{r_f} p_f u_{f,i,1} \tag{4.3}$$

$$\gamma_k = \gamma_{k-1} + \sum_{f=1}^{m}\sum_{i=1}^{r_f} p_f u_{f,i,k} \qquad 2 \le k \le r \tag{4.4}$$

$$\gamma_r = C_{\max} \tag{4.5}$$

$$D_{f,i} \ge \gamma_k - C_{\max}(1 - u_{f,i,k}) \qquad 1 \le f \le m,\ 1 \le i \le r_f,\ 1 \le k \le r \tag{4.6}$$

$$u_{f,i,k} \in \{0,1\} \qquad 1 \le f \le m,\ 1 \le i \le r_f,\ 1 \le k \le r \tag{4.7}$$

$$\gamma_k \ge 0 \qquad 1 \le k \le r \tag{4.8}$$

$$D_{f,i} \in [\min_f p_f, C_{\max}] \qquad 1 \le f \le m,\ 1 \le i \le r_f \tag{4.9}$$

The constraints (4.1) and (4.2) ensure that all batches are assigned to exactly one position and that all positions have exactly one batch assigned, respectively. (4.3) establishes the completion time of the first position, and (4.4) establishes the completion times of the remaining positions. (4.6) ensures the completion time of each batch is no earlier than the completion time of the positions to which they have been assigned. Finally, (4.7), (4.8), and (4.9) establish the valid ranges of the variables.

Define variables $A_{f,j,i}$ to be 1 if job (f, j) is assigned to batch indexed i of family f, or 0 otherwise. The part of the ILP model that assigns jobs to batches is as follows:

$$\sum_{i=1}^{r_f} A_{f,j,i} = 1 \qquad 1 \le f \le m,\ 1 \le j \le n_f \tag{4.10}$$

$$\sum_{j=1}^{n_f} A_{f,j,i} \le b_f \qquad 1 \le f \le m,\ 1 \le i \le r_f \tag{4.11}$$

$$D_{f,i} - C_{\max}(1 - A_{f,j,i}) \le C_{f,j} \qquad 1 \le f \le m,\ 1 \le j \le n_f,\ 1 \le i \le r_f \tag{4.12}$$

$$A_{f,j,i} \in \{0,1\} \qquad 1 \le f \le m,\ 1 \le j \le n_f,\ 1 \le i \le r_f \tag{4.13}$$

$$C_{f,j} \in [\min_f p_f, C_{\max}] \qquad 1 \le f \le m,\ 1 \le j \le n_f \tag{4.14}$$

The constraints (4.10) ensure that each job is assigned to exactly one batch, and (4.11) ensure that the number of jobs assigned to each batch is at most b_f—the maximum parallel capacity for batches of family f. (4.12) ensures the completion time $C_{f,j}$ of each job (f, j) is no earlier than the completion time of the batch to which it has been assigned. Finally, (4.13) and (4.14) establish the valid ranges of the variables.

To obtain the optimal primary objective value, we can make use of the polynomial time algorithm discussed in Section 4.2.1, or we can solve the following ILP model:

$$\begin{aligned} G^* = \min: \quad & G && (4.15) \\ \text{s.t.} \quad & G \geq g_{f,j}(C_{f,j}) \quad 1 \leq f \leq m, 1 \leq j \leq n_f && (4.16) \\ & (4.1) - (4.14) \end{aligned}$$

Note that if cost functions of completion times are all convex piecewise linear, then (4.16) can be easily written in linear functions of the existing variables.

Let $\bar{d}_{f,j}$ be the largest value of $C_{f,j}$ where $g_{f,j}(C_{f,j}) = G^*$. Here, $\bar{d}_{f,j}$ represents the deadline for job (f, j). As long as $C_{f,j} \leq \bar{d}_{f,j}$ for all jobs, the solution will remain optimal with respect to the primary criterion. We can then solve the bi-criteria problem with the following IP model:

$$\begin{aligned} \min: \quad & \sum_{f,j} h_{f,j}(C_{f,j}) && (4.17) \\ \text{s.t.} \quad & 0 \leq C_{f,j} \leq \bar{d}_{f,j} \quad 1 \leq f \leq m, 1 \leq j \leq n_f && (4.18) \\ & (4.1) - (4.14) \end{aligned}$$

where (4.18) ensures that the solution remains optimal with respect to the primary objective.

4.3.2 Simulated Annealing Approach

Since the considered problem with bi-criteria objective $(\max g_{f,j}, \sum h_{f,j})$ is NP-hard in general for g and h, we can expect that we will not be able to obtain exact solutions to problem instances beyond a certain size in practically acceptable time. In such circumstances, an approximate solution of good quality that can be obtained in a reasonable time is often adequate. Simulated Annealing (SA) is a probabilistic, general purpose framework for solving optimisation problems using a local search paradigm.

By virtue of the results in section 4.2.2, we can optimally assign jobs to a fixed sequence of batches in polynomial time. The correct sequence of batches will produce the optimal schedule to the original problem. The task of SA is therefore to explore the space of batch sequences in attempt to find the sequence of batches that gives the optimal schedule. In the context of our SA algorithm, a "solution" is a sequence of batches, and is distinguished from a "schedule", which is characterised by a sequence of batches together with a complete assignment of jobs to batches.

The neighbourhood function generates a neighbour solution to an existing one by swapping one or more pairs of batches in the sequence. With probability π_1, one pair of batches is swapped; with probability π_2, two pairs of batches are swapped; and so on, with the relation $\pi_1 \geq \ldots \geq \pi_r$ and $\sum_{i=1}^{r} \pi_i = 1$. The aim of this extended neighbourhood is to prevent the algorithm getting stuck at a solution where there exists no feasible neighbour if only a single pair of batches can be swapped.

Checking the feasibility of a potential neighbour against the primary criterion can be accomplished with $O(n)$ operations, where n is the number of jobs in the sequence. Recall the definition of the deadlines $\bar{d}_{f,j}$ from Section 4.3.1: $\bar{d}_{f,j}$ is largest value of $C_{f,j}$ where $g_{f,j}(C_{f,j}) = G^*$. One can dispatch the jobs in ascending order of $\bar{d}_{f,j}$ into the earliest available position in a batch of the same family. If all jobs can be arranged such that $C_{f,j} \leq \bar{d}_{f,j}$, then the batch sequence is feasible with respect to the primary criterion. If, however, $C_{f,j} > \bar{d}_{f,j}$ for any job, then the batch sequence cannot produce a solution of jobs that is feasible with respect to the primary criterion.

To prevent visiting the same solutions multiple times, we introduce a finite-length, first-in-first-out list that records recently visited solutions. The neighbourhood function is forbidden from generating solutions that appear in this list. This approach was discussed in Gunwan [73], where Simulated Annealing was hybridised with Tabu Search in a similar way, which improved the performance of the algorithm. Each time a new solution is visited, it is added to the list; if the list is full, the oldest solution is removed.

Our complete SA implementation is as follows:

Step 1. Based on the problem size, i.e. the number of batches, determine the initial temperature T_0, the termination temperature T_{min}, the cooling rate $0 < \alpha < 1$, the reheating threshold $\kappa > 1$, the reheating amount $\nu > 1$, the length of the visited solution list ψ, and the time limit $t_{\max}$.

Step 2. Initialise the iteration counter $i := 0$, the initial solution S_0, the best-known solution $S^* := S_0$.

Step 3. Generate a candidate neighbour solution S_{i+1} from current solution S_i, which doesn't exist in the list of previously visited solutions and which is feasible with respect to the primary criterion.

Step 4. If $x \leq exp(\frac{C(S_i)-C(S_{i+1})}{T_i})$ for some random $x \sim U(0,1)$ and cost function $C(S)$, accept S_{i+1} as the next solution and add it to the list of previously visited solutions. Otherwise $S_{i+1} := S_i$.

Step 5. If $C(S_i) < C(S^*)$ then $S^* := S_i$. If S^* hasn't been updated in κ iterations, $S_i = S^*$, and $T_i := \nu T_i$.

Step 6. Apply the cooling function: $T_{i+1} := \alpha T_i$, and increment the iteration counter i.

Step 7. If $T_i < T_{min}$ or the elapsed time exceeds $t_{\max}$, terminate the algorithm reporting back S^*, otherwise go to Step 3.

The initial solution S_0 is taken to be the batch sequence obtained when solving the problem for the primary criterion.

Our SA implementation explores the space of batch orders rather than directly exploring the space of solutions, and it uses exact methods to produce optimal schedules subject to those batch orders.

Such hybridisations between metaheuristics and mathematical programming techniques are known as matheuristics, as described in Burke [20].

4.3.3 Genetic Algorithm Approach

Another heuristic solution approach we consider is Genetic Algorithms (GA). As with most other metaheuristics, there are several variants of GA and there is no one agreed specification of precisely how it should be implemented. We consider a steady-state GA, where new individuals are added to, and unfit individuals are discarded, from a single population [145].

In GA, the representation of a solution is important so that they not only fully characterise the solution but that is also amenable to the necessary genetic operators. In our implementation of GA, we have chosen to represent a solution as an ordered sequence of batch indices. For example:

6	1	3	7	2	8	5	4

represents a solution where batch 6 is sequenced first, batch 1 is sequenced second, and so on, with batch 4 sequenced last. The indices that identify each batch is determined a priori. In any solution, each batch index must appear exactly once, and each solution must remain feasible with respect to the primary criterion.

The purpose of the selection operator is to decide which solution(s) will be selected from the population to produce offspring. It is possible to design the selection operator in a number of ways, and we have chosen to use the `n`-tournament operator [135]. The `n`-tournament operator takes a set of randomly selected individuals from the population and returns the one with best fitness, which is calculated as the objective function value. In our implementation, we use the `n`-tournament operator to select two parent solutions from which an offspring will be generated.

The purpose of the crossover operator is to take two or more solutions and combine them to produce a new solution. In our implementation, the crossover operator takes a random half of the elements from the first parent and places them in the same position in the offspring partial solution and then fills in the missing indices in the remaining places in the order they are defined in the second parent. For example, for parents

6	**1**	3	7	**2**	8	**5**	**4**

and

7	5	2	4	1	6	8	3

The crossover operator produces the partial solution from the first parent:

	1			2		5	4

and to complete the solution from the second parent, the remaining places are filled in with the missing indices in the order they are defined in the second parent:

7	1	6	8	2	3	5	4

The purpose of the mutation operator primarily to avoid stagnation of genetic diversity in the population, but there is also a possibility to improve a solution using random mutation by sheer chance. In our implementation of GA, the mutation operator is identical to the neighbourhood function in our SA implementation, i.e. one or more pairs of batches are swapped. For example:

7	1	**6**	8	2	**3**	5	4

after swapping the batches indexed 3 and 6, becomes:

7	1	**3**	8	2	**6**	5	4

To generate the initial population, the mutation operator is used repeatedly to produce a diverse set of feasible solutions. First, the initial solution, obtained when solving for the primary criterion, is added to the population. Then, until the population is of the desired size P, a random member of the population is chosen, the mutation operator is applied to it, and, if feasible, the mutated solution is added to the population.

When culling (discarding) solutions from the population, they are chosen randomly with non-uniform probability. The solutions with the best objective values have the lowest probability of being chosen, while the solutions with the worst objective values have the highest probability of being chosen. We create a reference list of indices Φ, which contains the index 1 once, the index 2 twice, the index 3 three times, and so on. Since the population contains at most P solutions, Φ contains $P(P+1)/2$ elements. The population is kept sorted at all times by objective value, from lowest to highest. When culling, the index of the solution to remove from the population is selected uniformly from Φ.

Our complete GA implementation is as follows:

Step 1. Based on the problem size, i.e. the number of batches, determine population size P, time limit $t_{\max}$, the tournament size $0 < \tau < P$, the cull rate $0 < \mu < P$, and the mutation rate $0 \leq \lambda \leq 1$.

Step 2. Initialise the initial population of solutions and identify the solution S^* with the best objective value.

Step 3. While the elapsed time is less than $t_{\max}$:

- Randomly discard μ solutions from the population.

Group	Families	Jobs/Family	Batch Sizes	Total Batches	Total Jobs
A	3	5	2	9	15
B	5	6	2	15	30
C	6	8	2	24	48
D	8	16	6	24	128
E	10	32	8	40	320
F	10	48	8	60	480
G	10	64	8	80	640

Table 4.1: Outline of the 7 difficulty groups.

- While the population size is less than P:

  i. use the `n`-tournament operator to pick two sets of τ solutions from the population and return the best solution from each set.

  ii. create a new offspring by applying the crossover operator on the first and second parents, and with probability λ apply the mutation operator.

  iii. if the offspring is feasible with respect to the primary criterion, add it to the population.

- If the best solution in the population has a better objective value than S^*, then S^* becomes the best solution in the population.
- If the elapsed time exceeds $t_{\max}$, terminate the algorithm reporting back S^*.

4.4 Computational Results

Using training records from the electricity distributor as a template, we generated 105 test cases divided into seven difficulty groups: A to G with increasing difficulty, each test group with 15 test cases. For each test case, employees were selected randomly from the organisation and their training requirements and due dates were used to create jobs. Table 4.1 outlines the properties of the seven test case groups. The columns from left to right are as follows: The name of the test group, the total number of families considered, the number of jobs in each family, the maximum size of any batch, the total number of batches in the problem instance, and the total number of jobs in the problem instance.

For our computational experimentation, we consider the maximum weighted tardiness primary objective and total weighted tardiness secondary objective, where the weights for the primary and secondary objectives are not necessarily equal. Recall that job tardiness is the time elapsed after the job due date before its completion. Since this problem is inspired by industry, we wish to explore problems and provide solutions that make practical sense to the organisation, and it is a simple matter to justify this particular ordered bi-criteria objective in the context of scheduling at the electricity distributor. We want a schedule that minimises tardiness, however a maximum weighted tardiness objective is not suitable as the quality of the schedule is determined solely based on the job whose weighted tardiness is greatest: all other jobs are therefore allowed to be scheduled such that their weighted tardiness can be up to that value without penalty. A total weighted tardiness objective, on the other hand, is also not suitable on its own as there may be little or no difference, with respect to the objective value, between a schedule that has a low but uniform tardiness across all jobs, and a schedule that has little to no tardiness across most jobs but exceptionally high tardiness across a small few. By minimising total weighted tardiness over the set of schedules that minimises the maximal weighted tardiness, we will obtain a solution that is of more practical use to the organisation.

Since the electricity distributor does not currently have any established way to quantify the cost or penalty for staff missing their training deadlines, random tardiness functions were generated for each employee. The $g_{f,j}(C_{f,j})$ and $h_{f,j}(C_{f,j})$ functions were unique to each staff member. For each staff member, we generated two random numbers $\alpha_{f,j}$ and $\beta_{f,j}$ based on their position in the company, distributed uniformly on the interval $[\frac{1}{2}, 1)$ for support workers and uniformly on the interval $[1, 2]$ for supervisors and managers. $g_{f,j}(C_{f,j})$ is then defined to be 0 if $C_{f,j} \leq d_{f,j}$, or $\alpha_{f,j}(C_{f,j} - d_{f,j})$ otherwise. Similarly, $h_{f,j}(C_{f,j})$ is 0 if $C_{f,j} \leq d_{f,j}$, or $\beta_{f,j}(C_{f,j} - d_{f,j})$ otherwise.

The computer used for all experiments was an Intel i7-2640M dual-core 2.8Ghz system with 4GB RAM running Microsoft Windows 7 Professional 64-bit, Service Pack 1. All ILP models were solved using IBM ILOG CPLEX 12.5.0.0 using default solver parameters [78]. The bipartite matching, polynomial time algorithms from section 4.2.1, and SA procedure were written in C#. Where applicable, our software communicated with IBM ILOG CPLEX from C# using the IBM ILOG Concert API.

The experimental process was as follows: For each of the 15 test cases in each of the 7 groups, Algorithm 1 was used to determine the optimal solution and objective value with respect to the primary criterion. Then, for each test case, SA and GA were each applied 20 times with different random seeds. The pseudorandom number generator we used was the MT19937 Mersenne Twister, re-seeded with the `CoCreateGuid` function from the Windows API for each run. For SA, the starting solution in each case was the solution from Algorithm 1. For both SA and GA, infeasible solutions were not permitted; when a neighbour, offspring, or mutation produced an infeasible solution, the solution was discarded and the operator(s) were applied again until a feasible solution was found.

The heuristic parameters for SA and GA were scaled linearly given the problem size. Since the SA and GA heuristics are exploring the sequence of batches, the problem size is defined by the total number of batches. Table 4.2 shows the parameters used for SA across the 7 test groups. The rows correspond to the initial temperature T_0, the terminating temperature $T_{\min}$, the annealing rate α, the

	A	B	C	D	E	F	G
T_0	1.0×10^2	8.5×10^4	2.1×10^5	2.1×10^5	4.4×10^5	7.2×10^5	1.0×10^6
T_{min}	0.01000	0.00916	0.00789	0.00789	0.00563	0.00282	0.000001
α	0.99000	0.99076	0.99190	0.99190	0.99393	0.99647	0.999000
κ	10	20	30	30	50	75	100
ν	1.1	1.1	1.1	1.1	1.1	1.1	1.1
ψ	50	100	150	150	250	375	500
t_{max}	5	15	40	120	1200	6800	9600

Table 4.2: The SA parameters used across the 7 test groups.

	A	B	C	D	E	F	G
P	50	88	145	145	246	373	500
τ	2	4	6	6	10	15	20
μ	5	7	8	8	12	16	20
λ	0.1	0.1	0.1	0.1	0.1	0.1	0.1
t_{max}	5	15	40	120	1200	6800	9600

Table 4.3: The GA parameters used across the 7 test groups.

number of iterations κ after which to return to the best known solutions if it has not been improved, the reheating rate ν which multiplies the temperature when returning to the best known solution, and the time limit in seconds t_{max}. Table 4.3 shows the parameters used for GA across the 7 test groups. The rows correspond to the constant population size P, the tournament size for selection τ, the number of unfit individuals to cull at each iteration μ, the mutation rate λ, and the time limit in seconds t_{max}.

The chosen values of $\pi_1, \pi_2, \ldots, \pi_r$ are shown in Table 4.4. These values were fixed for all test cases in all groups. We permitted a maximum of $r = 6$ batches to be swapped when generating a random neighbour.

We found that Algorithm 1 was able to compute an optimal solution for the primary criterion for even the hardest test cases very quickly (See Figure 4.3). It took between about half a millisecond on average for the A test cases, up to about 47 milliseconds on average for the G test cases. By comparison, the straight forward ILP approach for the primary criterion took 581.6ms on average to

π_1	π_2	π_3	π_4	π_5	π_6
0.7500	0.1875	0.0469	0.0117	0.0029	0.0007

Table 4.4: The probabilities π_k associated with swapping k pairs of batches in the neighbourhood function

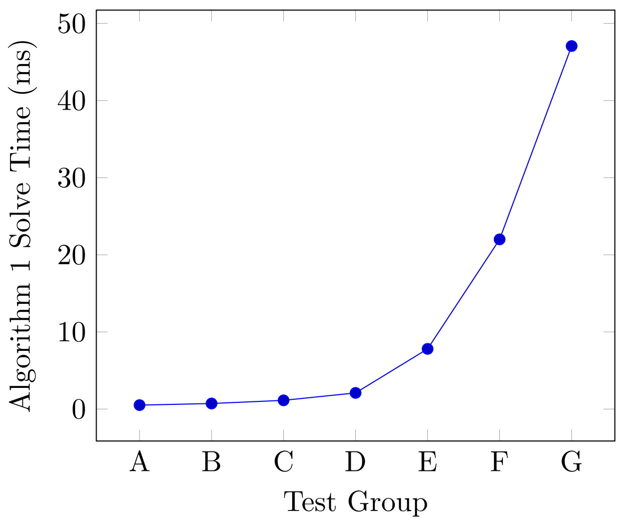


Figure 4.3: The average solve time in milliseconds for the test groups using Algorithm 1.

	A	B	C	D	E	F	G
Min	0.01	0.04	3.68	9.33	-	-	-
Mean	0.03	3.04	2990	8756	-	-	-
Max	0.14	19.8	14420*	34608*	-	-	-

Table 4.5: The minimum, mean, and maximum reported time, in seconds, for CPLEX to find an optimal solution.

solve the A test case, 13.15 seconds on average to solve the B test case, and was only able to find the optimal solution about a quarter of the time for the C test case given the five minute limit. Even when we increased the limit to 10 minutes, we were not able to find a solution for many of the C test cases. While there are some problems where methods with exponential worst case bounds are often preferred over polynomially bounded algorithms, such as the simplex algorithm for linear programming, our empirical results suggest that our polynomially bounded algorithm performs very well in practice and is preferable over mathematical programming.

Table 4.5 shows the minimum, mean, and maximum reported time, in seconds, for CPLEX to find an optimal solution for each of the 7 test groups. Test group C had a 12-hour time limit for CPLEX, which failed to find an optimal solution to one of the test cases. The lowest optimality gap reported by CPLEX in the C group was 13.39%, and the corresponding solution matched the best-known solution from SA and GA. For test group D, which had a 16-hour time limit for CPLEX, we were unable to find optimal, or even feasible, solutions to two of the test cases. We were unable to produce any optimal solutions at all for the remaining test groups using CPLEX, and the lowest optimality gap we were able to achieve in a 16-hour limit was 58.23%, however the corresponding incumbent solution matched the best solution produced by SA/GA for that test case.

	A	B	C	D	E	F	G
SA Min	1.00	1.00000	1.0000	1.0000	1.0000	1.0000	1.0000
SA Mean	1.00	1.00008	1.0006	1.0003	1.0011	1.0062	1.0090
SA Max	1.00	1.01458	1.0145	1.0117	1.0118	1.0258	1.0338
SA Opt	300	295	275	261	168	47	18
GA Min	1.00	1.00000	1.0000	1.0000	1.0000	1.0188	1.0397
GA Mean	1.00	1.00068	1.0040	1.0023	1.0123	1.0521	1.0740
GA Max	1.00	1.06649	1.0492	1.0387	1.0616	1.1294	1.1279
GA Opt	300	270	190	194	19	0	0

Table 4.6: The performance of SA and GA with respect to the optimal or best-known solutions.

Table 4.6 shows the performance of SA and GA with respect to solution quality. The rows show the average ratio of the best-known reported solution from the algorithm to the best-known solution overall from SA, GA, or IP where available, across each of the test cases; the average ratio of the mean reported solution to the best-known solution overall, across each of the test cases; the average ratio of the worst solution reported to the best-known solution overall, across each of the test cases; and the total number of reported solutions that matched the optimal or best-known solution for the corresponding test case out of 300, for SA and GA respectively. For the A test group, the SA and GA heuristics were able to find the optimal solution for all 15 cases in all 20 runs. For the B test group, both algorithms performed consistently well, with the reported solutions being just 0.008% and 0.068% greater than the optimal on mean for SA and GA respectively. The average performance of SA and GA continues to fall for groups C and higher, with the average performance of GA deteriorating faster. For the E group, SA continued to find the best-known solution 56% of the time, while GA only found the best-known solution about 6% of the time.

We recorded the best known solution at each iteration of each of run of SA and GA for each test case. In order to visualise the aggregated convergence behaviour of these algorithms, we normalised the time dimension and objective value dimension of the data to the range $[0, 1]$. In the time dimension, 0 refers to when the algorithm begins and 1 refers to the longest running time across the algorithms for that group. In the objective value dimension, 1 refers to the initial solution's objective function value from Algorithm 1 and 0 refers to the best-known or optimal solution's objective function value for the test case. For each point in the time dimension, we have a point in the objective value dimension for each of the runs in the test group. We use these samples to report the mean and median convergence of each algorithm in each test group, as well as the interquartile range of the convergence values across the 300 runs in each group.

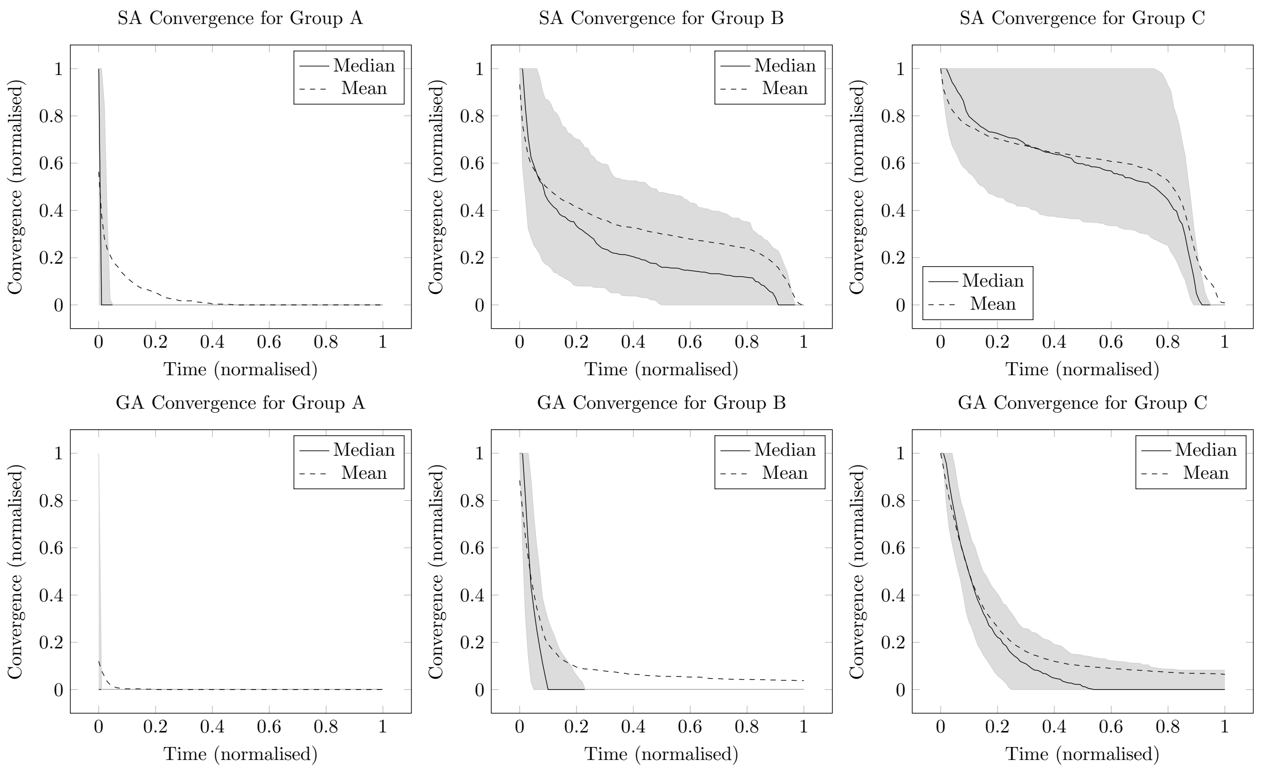


Figure 4.4: Convergence behaviour of SA and GA for test groups A, B, and C.

Figure 4.4 shows the performance of SA and GA for groups A, B, and C. SA was able to find the optimal solution in almost all of the test cases. For the A group, both algorithms found the optimal solution extremely quickly, however as the difficulty increased, both algorithms required more computational effort to find new and better solutions.

A representation of the behaviour of SA and GA across all the test cases in the D group can be seen in Figure 4.5. Observe that for the D test group, both SA and GA have an initial phase of rapid improvement, however the rate of improvement slows earlier in SA than in GA. SA has a second phase of rapid improvement towards the end of the algorithm as the temperature falls quite low, whereas the rate of improvement in GA appears to stagnate. SA has a better average convergence at the termination of the algorithm compared with GA.

The behaviour of the SA and GA algorithms for the E group is shown in Figure 4.6. It can be clearly seen that, on average, SA has difficulty improving the solution in the early stage while the temperature is high, but makes progress towards the end of the algorithm. Even though there is relatively high variability in the progress of SA early on, the overwhelming majority of the runs converge to the best-known solution near the end. In contrast, GA makes good progress early on, however, as with the D set, the rate of improvement slows towards the end. At the end of the algorithm, many runs fail to converge to the best-known solution, and there is a relatively high degree of variability among the performance across the runs.

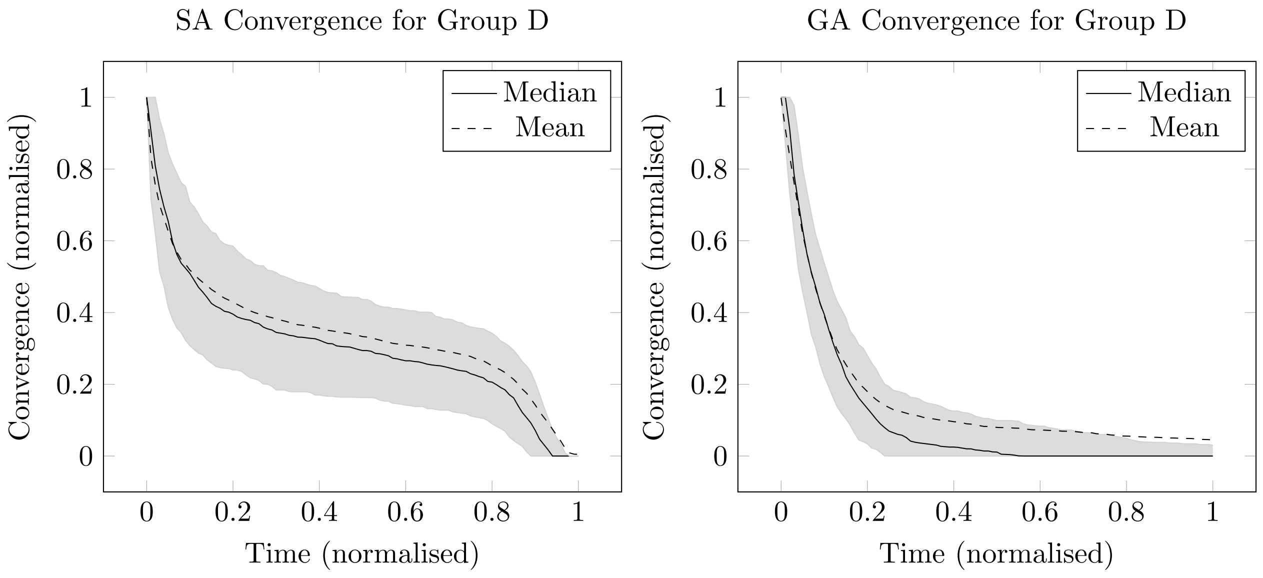


Figure 4.5: Convergence behaviour of SA and GA for the D test group.

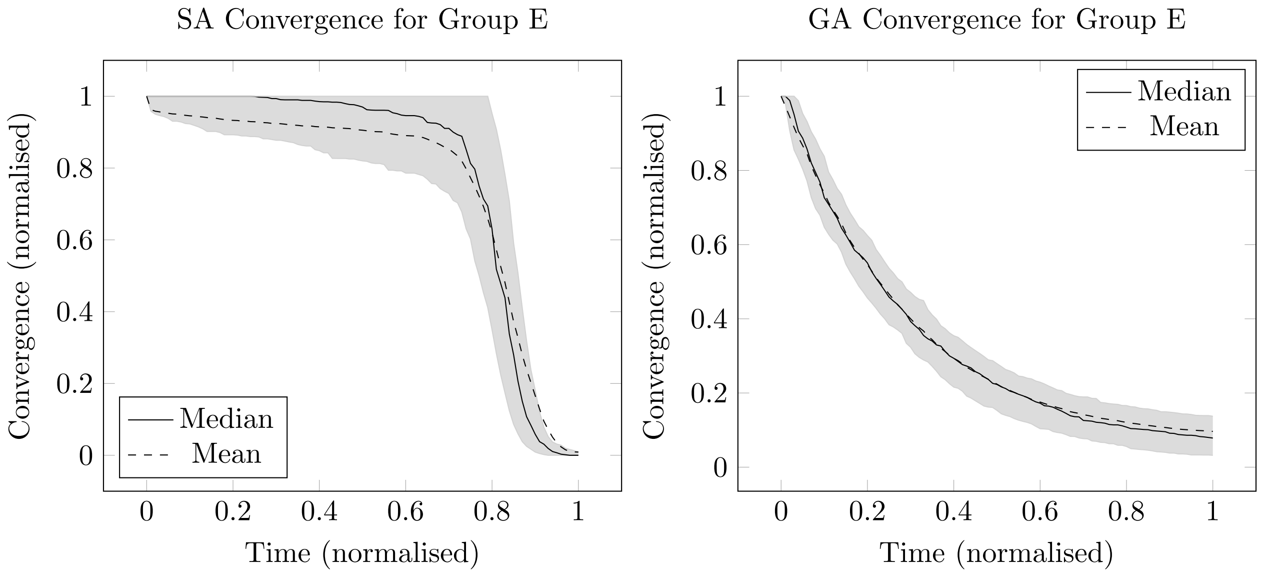


Figure 4.6: Convergence behaviour of SA and GA for the E test group.

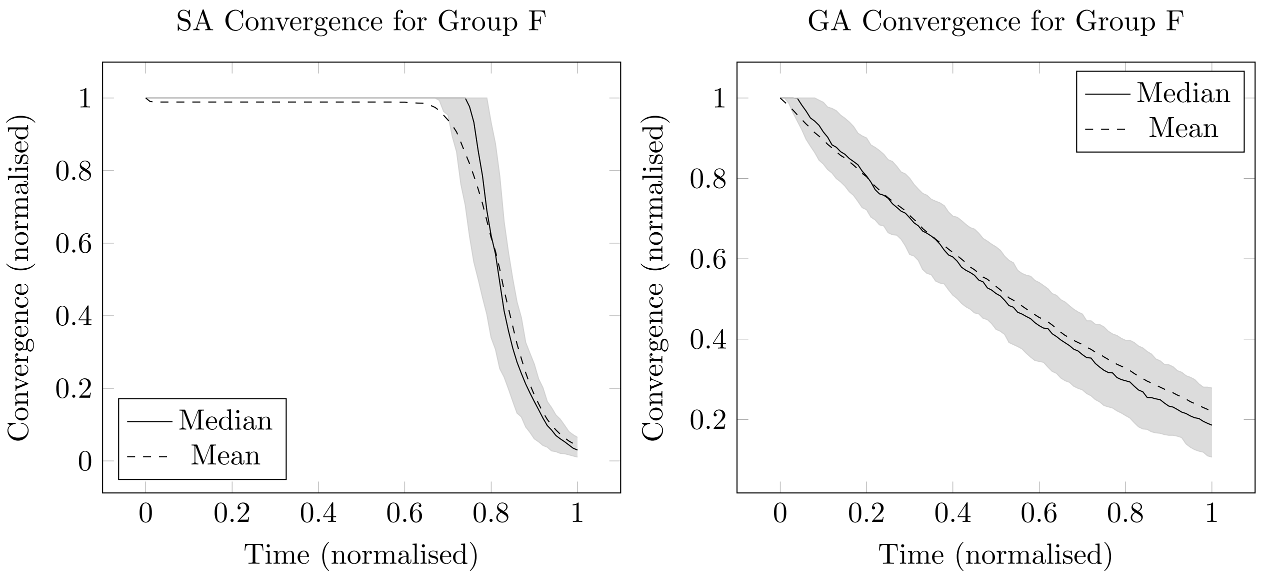


Figure 4.7: Convergence behaviour of SA and GA for the F test group.

The behaviour of the SA and GA algorithms for the F and G groups are shown in Figure 4.7 and Figure 4.8, respectively. For these groups, SA had pronounced difficulty improving the initial solution in the early stages of the algorithm, but made rapid and consistent improvement in the later stage, as the temperature became low. GA made slow and consistent improvements throughout the algorithm, but failed to find the best-known solution much of the time given the time limit.

These computational results suggest that, on average, while SA appears to find better solutions than GA given the time we allowed, it became progressively more difficult for the algorithm to make early improvement as the problem size increased. Had the allowable run time of the algorithm been just 25% shorter, most of the SA runs would have failed to improve the initial solution at all for the F or G groups. In contrast, GA appeared to have less difficulty making slow but regular improvements even as the problem size grew larger.

4.5 Conclusions

In this chapter, we studied the problem of class formation and sequencing for multiple courses subject to a bottleneck classroom with an ordered bi-criteria objective. We showed that this problem for the objective $(\max g_{f,j}, \max h_{f,j})$ is solvable in polynomial time, and that the $(\sum g_{f,j}, \sum h_{f,j})$ and $(\sum g_{f,j}, \max h_{f,j})$ cases are NP-hard in the strong sense in general for g and h by proving that minimising $\sum U_{f,j}$ is NP-hard in the strong sense; the problem may be solvable in polynomial time for particular g and h. The case of the $(\max g_{f,j}, \sum h_{f,j})$ objective was already shown to be NP-hard in the strong sense by Hoogeveen [76] for a distinct but comparable problem, for g and h in general; while again the problem may be solvable in polynomial time for particular g and h. We showed that that the problem

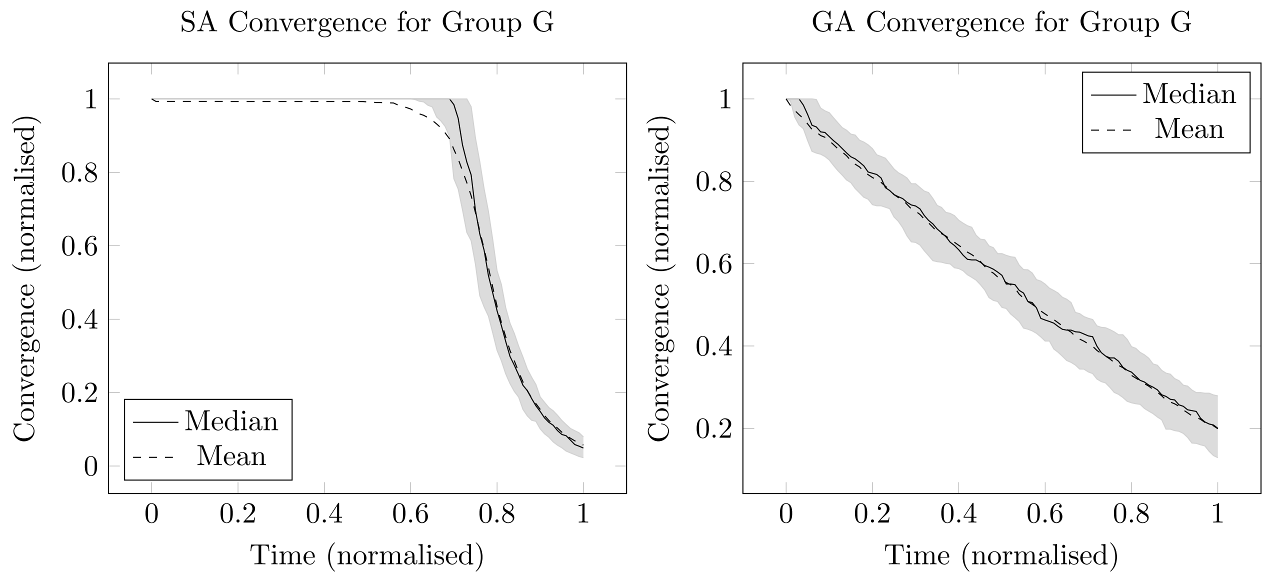


Figure 4.8: Convergence behaviour of SA and GA for the G test group.

of minimising $\sum h_{f,j}$ can be solved in polynomial time as a linear bottleneck assignment problem, assuming the sequence of batches is fixed in advance, and also showed this result can be easily extended to the ordered bi-criteria case.

We provided an $O(n^2)$ algorithm that is optimal for the $\max h_{f,j}$ objective, or $(\max g_{f,j}, \max h_{f,j})$ with minor modification. Empirical results show that this algorithm performs well in practice, even for large sized problem instances.

We provided an ILP model for the single machine batching problem, which can be used to find optimal solutions to small instances in a reasonable time. We proposed SA and GA matheuristics to solve larger problem instances in practically acceptable time, and showed that feasibility of a batch sequence with respect to the primary criterion can be established in $O(n)$ operations. We showed that our SA algorithm is able to find very good solutions in a short time for small to moderately sized problem instances. We showed that our GA algorithm was also able to find good solutions for medium to large problem sizes, making slower but more consistent improvements compared to SA.

Future research for this problem can be directed towards investigating more sophisticated mathematical programming tools, such as decomposition methods, to allow finding optimal solutions of larger sized problems. To supplement this preliminary analysis of two heuristic approaches, future research can investigate other metaheuristics and approximation methods. Future research can address the related problem of bi-criteria sequencing of courses and class formation for multiple classrooms.

Chapter 5

Partitioning of Students into Classes

Contents

The research in this chapter is an amalgamation of what was presented at the 10th International Workshop on Algorithms and Computation (WALCOM) 2016 [157, 41] and included in the refereed conference proceeding as "O. G. Czibula, H. Gu, and Y. Zinder. A Lagrangian relaxation-based heuristic to solve large extended graph partitioning problems. In *WALCOM: Algorithms and Computation*, pages 327–338. Springer, 2016", and what was presented at the 4th International Symposium on Combinatorial Optimization (ISCO) 2016 [80, 42] and included in the refereed conference proceeding as "O. G. Czibula, H. Gu, and Y. Zinder. Scheduling personnel retraining: Column generation heuristics. In *International Symposium on Combinatorial Optimization*, pages 213–224. Springer, 2016". The research was extended and submitted to the WALCOM 2016 Special Issue by Theoretical Computer Science, ans was accepted for publication [43], and submitted to the ISCO 2016 Special Issue by the Journal of Combinatorial Optimisation.

Abstract

This chapter is concerned with a partitioning problem. One of the applications, and the motivation for this research, is the problem of class formation for training and retraining sessions at large electricity distributors. The large scale of this problem together with various restrictions on the resultant assignment to classes make this planning a significant challenge. The chapter presents a complexity analysis of this problem together with linear and nonlinear mathematical programming formulations. Four different approaches are developed. The first is based on the Quadratic Multiple Knapsack formulation and Lagrangian Relaxation. The second is a collection of three different Column Generation based optimisation procedures. The third is a Large Neighbourhood Search procedure, incorporating Column Generation. The fourth is a matheuristic developed as an amalgamation of Genetic Algorithms and Integer Programming. The approaches are tested by means of computational experiments. The experiments used data typical to large electricity distributors.

5.1 Introduction

This chapter is concerned with the combinatorial optimisation problem where the union of disjoint sets $F_1, \dots, F_K$ (referred to as families) must be partitioned into sets $S_1, \dots, S_N$ in such a way that each S_i is either empty, or has cardinality between a_i and b_i, i.e. $a_i \le |S_i| \le b_i$, and each nonempty set S_i must contain elements from at least p_i and at most q_i families. All a_i, b_i, p_i, and q_i are given.

There is a cost $c_{i,j}$ associated with the inclusion of each element j of $F_1 \cup \dots \cup F_K$ into each set S_i. For any two distinct families F_k and F_l, and any set S_i, the cost $b_{k,l}$ is incurred for the simultaneous presence of elements from both families F_k and F_l in S_i. This cost does not depend on the number of elements from F_k and F_l in S_i. The objective is to minimise the total cost associated with the partition $S_1, \dots, S_N$.

The above mentioned combinatorial optimisation problem was motivated in particular by the problem of class formation for training and retraining at large electricity distributors. Such organisations typically have thousands of workers of different types, and some also provide training to their contractors and to third parties. Due to the multitude of hazards that exist when working with high voltages, at heights, in confined spaces, or in the presence of harmful substances, local laws often require all such workers to undergo regular safety, technical, and professional training.

Keeping in mind the primal goal of the training provider, it is reasonable to refer to the people undergoing training as workers, although such terms as students and trainees, justified by the broad variety of participants, are often used in practice, and in the context of this chapter have the same meaning.

Many of these students have very different learning outcomes from these courses, different learning styles, different levels of education or English proficiency, and different levels of technical proficiency for certain tasks. Although assigning only one type of student into any class enables the training delivery to be better tailored to the needs of the specific group, in some cases it is beneficial to combine several compatible types of students in the same class, for example to facilitate the exchange of knowledge and experience between student types. Furthermore, due to the cost of delivering training and the scarcity of training resources, it is often not possible to run segregated classes, and some undesirable blending of student types may be necessary. Each class can either be empty, in which case it will be cancelled with no penalty, otherwise it must contain between a minimum and a maximum number of students, and between a minimum and maximum number of student types.

Most of the training required for anyone working on or near the electricity network in Australia has a limited period in which it is valid. Workers are only permitted to work in roles for which they have up-to-date training. Therefore the company incurs a certain cost each time an employee is not permitted to work due to the expiration of required training. Furthermore, training sessions (classes) have different suitability for a trainee not only because they are held at different dates but also because they differ by location, teaching mode, etc. Instead of specifying for each class all these attributes, the models below involve a cost (penalty) for allocating a trainee to a particular class. The objective is to minimise the total penalty. When a single course is considered, the above leads to the combinatorial optimisation problem studied in this chapter.

The remainder of this chapter is organised as follows. Section 5.2 presents a Linearly Constrained Quadratic Programming formulation, as well as its linearisation, for the studied problem. Section 5.3 analyses the complexity of the considered partitioning problem. Section 5.4 discusses a Lagrangian Relaxation of the Quadratic Program. Section 5.5 presents the first approach, which is a heuristic based on the Lagrangian Relaxation. Section 5.8 presents the second approach, which is an amalgamation of Genetic Algorithms and Integer Programming. Section 5.9 discusses the results of the computational experimentation. Our concluding remarks are given in Section 5.10.

5.2 Quadratic Programming Formulation

Let $\mathcal{N} = \{1, \cdots, N\}$, $\mathcal{M} = \{1, \cdots, M\}$, and $\mathcal{K} = \{1, \cdots, K\}$ be the set of available classes, the set of students to be assigned, and the set of student types, respectively. Denote the penalty of assigning student $j \in \mathcal{M}$ to class $i \in \mathcal{N}$ by $c_{i,j}$, and the penalty of pairing student types $k \in \mathcal{K}$ and $l \in \mathcal{K}$ together in the same class by $b_{k,l}$. Each student has exactly one type, and the set of students who are of type k is represented by F_k, $k \in \mathcal{K}$. Each student must be assigned to exactly one class, but not all classes must be run. Each class $i \in \mathcal{N}$ that is run must contain at least a_i and at most b_i students, and at least p_i and at most q_i student types. Students or student types cannot be assigned to classes that are not run. The binary variable $X_{i,j}$ is defined to be 1 if student j is assigned to class i, or 0 otherwise; the binary variable $Y_{i,k}$ is defined to be 1 if student type k is assigned to class i, or 0 otherwise; The binary variable Z_i is defined to be 1 if class i is run, or 0 otherwise. The following Quadratic Program describes the problem:

$$\text{(QP) min:} \quad \alpha \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{K} b_{k,l} Y_{i,k} Y_{i,l} + \beta \sum_{i=1}^{N} \sum_{j=1}^{M} c_{i,j} X_{i,j} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} X_{i,j} = 1 \qquad j = 1, \ldots, M \tag{5.2}$$

$$a_i Z_i \le \sum_{j=1}^{M} X_{i,j} \le b_i Z_i \quad i = 1, \ldots, N \tag{5.3}$$

$$p_i Z_i \le \sum_{k=1}^{K} Y_{i,k} \le q_i Z_i \quad i = 1, \ldots, N \tag{5.4}$$

$$X_{i,j} \le Y_{i,k} \qquad i = 1, \ldots, N; k = 1, \ldots, K; j \in F_k \tag{5.5}$$

$$Y_{i,k} \leq \sum_{j \in F_k} X_{i,j} \qquad i = 1, \ldots, N; k = 1, \ldots, K \tag{5.6}$$

$$X_{i,j} \in \{0, 1\} \qquad i = 1, \ldots, N; j = 1, \ldots, M \tag{5.7}$$

$$Y_{i,k} \in \{0, 1\} \qquad i = 1, \ldots, N; k = 1, \ldots, K \tag{5.8}$$

$$Z_i \in \{0, 1\} \qquad i = 1, \ldots, N \tag{5.9}$$

The quadratic term in (5.1) represents the penalty of pairing student types together, and the linear term represents the penalty of assigning students to classes, weighted by coefficients α and β, respectively.

The constraints (5.2) express the requirement that each student must be assigned to exactly one class. The constraints (5.3) and (5.4) express the requirement that each running class must have between a_i and b_i students, and between p_i and q_i student types, respectively, if the class is run, or zero otherwise. The constraints (5.5) express the requirement that a student may only be assigned to a class if that student's type has also been assigned to that class. The constraints (5.6) ensure that a class can only be assigned a type if at least one student of that type is in the class.

It is possible to linearise the quadratic term in (5.1) by introducing $\hat{Y}_{i,k,l} = Y_{i,k}Y_{i,l}$ together with constraints:

$$\hat{Y}_{i,k,l} \leq Y_{i,k} \qquad i = 1, \ldots, N; k = 1, \ldots, K; l = 1, \ldots, K \tag{5.10}$$

$$\hat{Y}_{i,k,l} \leq Y_{i,l} \qquad i = 1, \ldots, N; k = 1, \ldots, K; l = 1, \ldots, K \tag{5.11}$$

$$\hat{Y}_{i,k,l} \geq Y_{i,k} + Y_{i,l} - 1 \qquad i = 1, \ldots, N; k = 1, \ldots, K; l = 1, \ldots, K \tag{5.12}$$

to give the linearised model:

$$\text{(LQP) min:} \quad \alpha \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{K} b_{k,l} \hat{Y}_{i,k,l} + \beta \sum_{i=1}^{N} \sum_{j=1}^{M} c_{i,j} X_{i,j} \tag{5.13}$$

$$\text{s.t.} \quad (5.2) - (5.12)$$

$$\hat{Y}_{i,k,l} \in \{0, 1\} \quad i = 1, \ldots, N; k = 1, \ldots, K; l = 1, \ldots, K \tag{5.14}$$

An augmented model (AQP) can be constructed by relaxing constraints (5.2) and introducing variables $S_j \in \mathbb{Z}^+$ and $T_j \in \mathbb{Z}^+$, where $\mathbb{Z}^+$ is the set of nonnegative integers, which together represent the deviation in the number of classes to which student j is assigned. The sum of these variables is then heavily penalised in the objective function:

$$\text{(AQP) min:} \quad \alpha \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{K} b_{k,l} Y_{i,k} Y_{i,l} + \beta \sum_{i=1}^{N} \sum_{j=1}^{M} c_{i,j} X_{i,j} + \gamma \sum_{j=1}^{M} (S_j + T_j) \tag{5.15}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} X_{i,j} + S_j - T_j = 1 \quad j = 1, \ldots, M \tag{5.16}$$

$$
\begin{aligned}
&(5.3) - (5.9) && \\
&S_j \in \mathbb{Z}^+ && j = 1, \ldots, M \qquad (5.17)\\
&T_j \in \mathbb{Z}^+ && j = 1, \ldots, M \qquad (5.18)
\end{aligned}
$$

where γ is very large, typically several orders of magnitude greater than α and β. The (LQP) model can be modified in the same way to form the (ALQP) augmented linearised model.

The advantage of the augmented model is that feasible solutions are significantly easier to find, as violations of (5.2) allow (5.3) and (5.4) to be satisfied more easily. In the event that a solution cannot be found in which $\max_j S_j = \max_j T_j = 0$, the organisation can then decide what steps to take next, for example to create additional classes, to modify class size constraints, etc.

The time required to find an optimal solution to the above mentioned models grows rapidly, where even some small test cases with just a few dozen students can take several hours to solve. As the problem instances we hope to solve are significantly larger than this, we propose to use a heuristic approach.

5.3 NP-completeness

It is known that the clique problem, specified by an integer k and a graph $G(V, E)$, where V is the set of nodes (vertices) and E is the set of edges, is NP-complete in the strong sense, requiring an answer to the question whether or not $G(V, E)$ contains k nodes that form a complete subgraph [64].

This section presents a proof of the NP-completeness in the strong sense of the following problem:
C_PARTITION
Input: a set S; a function c that associates with each pair $\{j, g\} \subset S$ of distinct elements a nonnegative integer $c(j, g)$; a nonnegative integer C; and a set of pairs of positive integers

$$\{(a_1, b_1), \ldots, (a_N, b_N)\},$$

where $a_i \leq b_i$ for all $1 \leq i \leq N$.
Question: does there exist a partition $S = S_1 \cup \ldots \cup S_N$ such that, for each $1 \leq i \leq N$, either $S_i = \emptyset$ or $a_i \leq |S_i| \leq b_i$, and

$$\sum_{1 \leq i \leq N} \sum_{\substack{\{j,g\} \subseteq S_i \\ j \neq g}} c(j, g) \leq C?$$

It is easy to see that the C_PARTITION problem is a decision version of a particular case of the partitioning problem considered in this chapter. Therefore, the NP-completeness in the strong sense of C_PARTITION implies the NP-hardness in the strong sense of the original partitioning problem. The proof is a polynomial reduction to C_PARTITION the following problem:

CLIQUE

Input: undirected graph $G(V,E)$ and positive integer k

Question: does $G(V,E)$ contain a complete subgraph of k nodes?

It is well known that the CLIQUE problem is NP-complete in the strong sense. [64].

Theorem 2. *CLIQUE $\propto$ C_PARTITION and therefore C_PARTITION is NP-complete in the strong sense.*

Proof. For any instance of CLIQUE with graph $G(V,E)$ and integer k, the corresponding instance of C_PARTITION is constructed as follows:

- $S = V$,
- $N = |V| - k + 1$,
- $C = 0$,
- $a_i = b_i = 1$ for $1 \le i < N$ and $a_N = b_N = k$,
- $c(j,g) = \begin{cases} 0, & \text{if } [j,g] \in E \\ 1, & \text{otherwise} \end{cases}$.

Then, for any feasible partition, $|S_N| = k$ and $|S_i| = 1$ for all $1 \le i < N$. Hence,

$$\sum_{1\le i\le N} \sum_{\substack{\{j,g\}\subseteq S_i \\ j\ne g}} c(j,g) = \sum_{\substack{\{j,g\}\subseteq S_N \\ j\ne g}} c(j,g) \le 0$$

if and only if $G(V,E)$ contains a complete subgraph of k nodes. □

5.4 Lagrangian Relaxation

We can formulate the Lagrangian Relaxation model by moving the constraints (5.2) to the objective function:

$$\text{(QR) min:} \quad \alpha \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{K} b_{k,l} Y_{i,k} Y_{i,l} + \beta \sum_{i=1}^{N} \sum_{j=1}^{M} c_{i,j} X_{i,j} + \sum_{j=1}^{M} \lambda_j \left(1 - \sum_{i=1}^{N} X_{i,j}\right) \tag{5.19}$$

$$\text{s.t.} \quad (5.3) - (5.9) \tag{5.20}$$

where $\lambda_j \in \mathbb{R}$, $j = 1, \ldots, M$, is the Lagrangian multiplier corresponding to the jth constraint (5.2), which can be interpreted as the penalty of not assigning the jth student to exactly one class. The quadratic term in (5.19) can be linearised in the same way as with (5.1).

In (QP), only constraints (5.2) couple together the N subproblems of assigning students to *a particular* class i. In (QR), these constraints are moved to the objective function, therefore it is possible to express (QR) as N smaller, class-specific subproblems (QR$_i$):

$$\begin{aligned}
(\mathrm{QR}_i)\ \min:\quad & \alpha\sum_{k=1}^{K}\sum_{l=1}^{K} b_{k,l}Y_{i,k}Y_{i,l} + \beta\sum_{j=1}^{M} c_{i,j}X_{i,j} - \sum_{j=1}^{M}\lambda_j X_{i,j} && && (5.21)\\
\text{s.t.}\quad & a_i Z_i \le \sum_{j=1}^{M} X_{i,j} \le b_i Z_i && && (5.22)\\
& p_i Z_i \le \sum_{k=1}^{K} Y_{i,k} \le q_i Z_i && && (5.23)\\
& X_{i,j} \le Y_{i,k} && k = 1,\ldots,K; j \in F_k && (5.24)\\
& X_{i,j} \in \{0,1\} && j = 1,\ldots,M && (5.25)\\
& Y_{i,k} \in \{0,1\} && k = 1,\ldots,K && (5.26)\\
& Z_i \in \{0,1\} && && (5.27)
\end{aligned}$$

where the N solutions to (QR$_i$) for $1 \le i \le N$ are combined to form the solution to (QR).

The Lagrangian Relaxation (QR), as its name suggests, is a relaxation on the original model (QP). That is to say, an optimal solution to (QR) given some vector λ gives a lower bound on the optimal objective value to (QP). The Lagrangian Dual corresponds to the problem of finding the λ vector that gives the tightest lower bound. This vector is denoted as λ^*. It is a well-known result that the optimal solution to the Lagrangian Dual gives a lower bound to (QP) that is at least as tight as the lower bound provided by the linear relaxation, i.e. the removal of the integrality constraints, of (QP) [72, 59]. A typical way to solve the Lagrangian Dual is by means of the iterative subgradient algorithm, which iteratively updates λ using the relationship

$$\lambda^{k+1} = \lambda^k + \frac{s^k \cdot \epsilon_k(\eta^* - \eta^k)}{||s^k||^2} \tag{5.28}$$

The value of the Lagrangian multipliers λ at the kth iteration is denoted by λ^k. The optimal objective value of the Lagrangian Dual problem is given by η^*, and η^k is the objective value of (QR) at the kth iteration. Since the value of η^* is generally not known in advance, it is typically estimated by means of some heuristic, which will likely give an underestimation on the true value. Therefore, the positive scaling factor ϵ_k is introduced, which is typically given an initial value of 2 and halved whenever the best-known (tightest) lower bound hasn't been improved in a certain number of iterations [72]. A subgradient s^k of the Lagrangian Dual at iteration k can be given by

$$s_j^k = 1 - \sum_{i=1}^{N} X_{i,j}^k \quad j = 1,\ldots,M \tag{5.29}$$

where $X_{i,j}^k$ are the values of the variables obtained at the kth iteration.

5.5 Lagrangian Heuristic

The Lagrangian Dual problem from Section 5.4 is solved by means of the subgradient algorithm. In doing so, one solution to the Lagrangian Relaxation problem is generated for each iteration. These solutions may be infeasible with respect to the original problem, since constraints (5.2) are relaxed. However, by observing trends in certain features of these solutions, it is possible to impose assumptions on the solutions to the underlying problem that reduce the decision space significantly.

The nature of the assumptions we propose to impose is one in which pairs of students must be assigned to the same class. Naturally, if an assumption is imposed that students j_1 and j_2 must be paired together, and that students j_2 and j_3 must be paired together, then it follows that students j_1 and j_3 must be paired together also. One can represent these assumption in the form of a graph, where each vertex corresponds to a student, and each edge between vertices j_u and j_v corresponds to a pairing assumption between students j_u and j_v. From the terminology of graph theory, a set of such students that must be assigned together is represented by a clique. Due to the transitive nature of student pairing assumptions in this problem, all such cliques are necessarily disjoint.

One of the parameters of the proposed LR-based heuristic determines whether or not to allow cliques with more than 2 vertices. If these cliques are not permitted, then once student pair $\{j_u, j_v\}$ is added to the set of assumptions, no further assumptions may be imposed involving students j_u or j_v.

Another parameter of the proposed heuristic is $\rho \in \{0, \ldots, M\}$, which defines the number of explicit pairs to impose. For example, if the pairs $\{j_1, j_2\}$ and $\{j_2, j_3\}$ are imposed, then it is not necessary to explicitly impose the pair $\{j_1, j_3\}$ as it will be implicitly satisfied.

During the subgradient algorithm to compute the Lagrangian Dual, many partial solutions are generated, and these are added to a solution pool. For all pairs of students $\{j_u, j_v\}$, where $u < v$, each solution in the pool is examined. The number of solutions where j_u and j_v are together in the same class is denoted $t_{u,v}$. We impose an assumption that those ρ student pairs $\{j_u, j_v\}$ who have the highest $t_{u,v}$ values must be paired together in the final solution. Other ranking criteria can be used to decide which pairs of students must be assigned together.

It is important that the imposed assumptions reduce the problem size in such a way that the resulting problem is not more difficult to solve. For example, imposing too many assumptions about paired students could make finding feasible solutions very difficult, impairing the performance of the algorithm. A number of other assumption types are considered in [41], including pairs of students who must be assigned separately. The approach of imposing assumptions where students should be assigned separately is less effective as the problem size grows much larger, such as the test cases considered in this chapter, as each such assumption on student pairs $\{j_u, j_v\}$ assigned separately requires new constraints $X_{i,j_u} + X_{i,j_v} \leq 1$ for each class i. As the number of classes and students grows very large, the large number of these introduced constraints can have the unwanted effect of hindering the performance of the algorithm.

It is possible to impose a wide variety of assumptions in a similar manner. For example, one can impose restrictions on which classes must run, and which classes must not, by examining the frequency with which they run or do not run in the solutions from the subgradient algorithm. Likewise, one can impose restrictions on which student types must be assigned to particular classes and which student types must not. In our experience these other types of assumptions can be quite heavy handed, and subsequently make finding feasible solutions to the underlying problem, subject to the assumptions, difficult. It is a matter for further research to investigate whether these assumptions, or others, can be successfully incorporated into this LR-based heuristic framework.

Denote the set of student pairs who must be assigned to the same class as $\mathscr{T}$. The (QP) model, subject to imposed assumptions, is as follows:

$$\begin{aligned} \text{(AQP) min:} \quad & \alpha \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{K} b_{k,l} Y_{i,k} Y_{i,l} + \beta \sum_{i=1}^{N} \sum_{j=1}^{M} c_{i,j} X_{i,j} && (5.30) \\ \text{s.t.} \quad & (5.2) - (5.9) \\ & X_{i,j_u} = X_{i,j_v} \quad i = 1, \ldots, N; \{j_u, j_v\} \in \mathscr{T} && (5.31) \end{aligned}$$

where (5.31) imposes the assumptions that student pairs in $\mathscr{T}$ must be assigned to the same class.

The (AQP) model above is described with equality constraints (5.31) for convenience, however the model can easily be modified to contain fewer variables. This can be accomplished by redefining the primary variables X so that they no longer correspond to the assignment of a particular student to a class, but a set of one or more students to a class. Minor alterations must also be made to the objective function and the constraints, including a weight coefficient (corresponding to the number of students represented by the variable) added to constraints (5.3).

The proposed **LR-based Heuristic** uses the following parameters: ρ defines the number of non-implicit pairs to add to $\mathscr{T}$, and a parameter that either permits or forbids cliques with more than 2 vertices.

LR-based Heuristic:

Step 1. Initialise a set of assumed pairs $\mathscr{T}$ as an empty set.

Step 2. Run **LD-Subroutine** and store the result $\mathscr{P}$. For each pair of students $\{j_u, j_v\}$, where $u < v$, count the number of instances $t_{u,v}$ in $\mathscr{P}$ where j_u and j_v are assigned to the same class.

Step 3. Find the student pair $\{j_u, j_v\}$ not already in $\mathscr{T}$, or if only 2-vertex cliques are permitted then the pair $\{j_u, j_v\}$ where neither j_u nor j_v are present in $\mathscr{T}$, whose corresponding $t_{u,v}$ has the greatest value, and add that pair to $\mathscr{T}$. If $|\mathscr{T}| < \rho$ then repeat Step 3.

Step 4. Solve (AQP) subject to $\mathscr{T}$, and return the solution.

LD-Subroutine:

Step 1. Initialise the Lagrangian multiplier vector to its starting value $\lambda^1 := 0$, ϵ_1, determine an estimate for η^* by means of a heuristic, set the tightest lower bound $\bar{\eta} := -\infty$, initialise the empty solution pool $\mathscr{P}$, and initialise the iteration counter $k := 1$.

Step 2. Construct the (QR$_i$) models for $1 \leq i \leq N$, solve them, and combine the N partial solutions to form the complete solution and add it to $\mathscr{P}$. Compute λ^{k+1} according to (5.28).

Step 3. If the objective value of the complete solution η^k is greater than $\bar{\eta}$, then set $\bar{\eta} := \eta^k$.

Step 4. If $\bar{\eta}$ has not been improved in the last $\tilde{\epsilon}$ iterations, then $\epsilon_{k+1} := \frac{1}{2}\epsilon_k$. Otherwise, $\epsilon_{k+1} := \epsilon_k$.

Step 5. If k exceeds the maximum number of iterations, then terminate `LD-Subroutine` and return $\bar{\eta}$ and $\mathscr{P}$; otherwise set $k := k + 1$ and return to Step 2.

5.6 Column Generation Approaches

Define $\mathcal{P}_i$ to be the set of all feasible sets of trainees that can be assigned to class i. We define $p \in \mathcal{P}_i$ to be a *pattern*.

We can then define the set-covering formulation:

$$\text{(M) min:} \quad \sum_{i=0}^{N} \sum_{p \in \mathcal{P}_i} c_p^i X_p^i \tag{5.32}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_i} X_p^i \leq 1 \qquad i = 1, \ldots, N \tag{5.33}$$

$$\sum_{i=1}^{N} \sum_{p \in \mathcal{P}_i} s_{p,j}^i X_p^i = 1 \quad j = 1, \ldots, M \tag{5.34}$$

$$X_p^i \in \{0, 1\} \qquad i = 1, \ldots, N; p \in \mathcal{P}_i \tag{5.35}$$

where the binary variable X_p^i is 1 if pattern p is selected for class i or zero otherwise; c_p^i is the cost of selecting pattern p for class i, and $s_{p,j}^i$ is 1 if student j exists in pattern p, or 0 otherwise.

The constraints (5.33) express the requirement that each class can have at most one pattern selected, and the constraints (5.34) express the requirement that each trainee must be assigned to exactly one class.

For problems of practical size, there are far too many patterns in the master problem (M) to consider explicitly. For all such problems, it is useful to consider the reduced master problem (RM) that is identical to (M), but contains only a subset of patterns $P_i \subseteq \mathcal{P}_i$.

The (RM) objective function and constraints equivalent to (5.34) can easily be modified to incorporate the S_j and T_j variables, as with (AQP), to give the augmented reduced master problem (ARM).

According to the Column Generation approach [46], new columns (patterns) are iteratively added to P_i for each class i. First, the linear relaxation of the reduced master problem is solved for an initial set of patterns. If we solve the augmented variant, the initial set of patterns can be empty. At each iteration, new patterns can be generated by solving with an IP solver, for each class i, the subproblem:

$$\begin{aligned}
(\text{SP}_i)\ \text{min:}\quad & \alpha\sum_{k=1}^{K}\sum_{l=1}^{K} b_{k,l}Y_{i,k}Y_{i,l} + \beta\sum_{j=1}^{M} c_{i,j}X_{i,j} - \pi_{1,i} - \sum_{j=1}^{M}\pi_{2,j}X_{i,j} && (5.36)\\
\text{s.t.}\quad & a_i \le \sum_{j=1}^{M} X_{i,j} \le b_i && (5.37)\\
& p_i \le \sum_{k=1}^{K} Y_{i,k} \le q_i && (5.38)\\
& X_j \le Y_{i,k} \qquad k=1,\dots,K; j\in F_k && (5.39)\\
& Y_{i,k} \le \sum_{j\in F_k} X_k \qquad k=1,\dots,K && (5.40)\\
& X_{i,j}\in\{0,1\} \qquad j=1,\dots,M && (5.41)\\
& Y_{i,k}\in\{0,1\} \qquad k=1,\dots,K && (5.42)
\end{aligned}$$

where $\pi_{1,i}$ is the dual variable corresponding to constraint (5.33) for class i, and $\pi_{2,j}$ is the dual variable corresponding to constraint (5.34) for student j from the linear relaxation of (ARM). The objective of (SP$_i$) is therefore to find the pattern with the most negative reduced cost to add to P_i.

At each iteration, the linear relaxation of (RM) or (ARM) is solved subject to the set of available patterns P_i, and (SP$_i$) is solved for each class i to find new patterns with the most negative reduced cost. If (SP$_i$) cannot produce a pattern with negative reduced cost, then no new patterns are available for class i in this iteration. When no new patterns are available for any class, the Column Generation procedure has reached its conclusion and can be terminated.

The above-mentioned Column Generation procedure will most likely not yield a feasible integer solution to the original problem as it is solved for the linear relaxation of (RM) or (ARM). We now present three solution approaches, incorporating the Column Generation procedure, to produce integer solutions.

5.6.1 Reduced Master Heuristic

The most simple solution approach is to solve (RM) or (ARM) with their original, unrelaxed integer variables, subject to the patterns generated according to the Column Generation approach. This approach is not guaranteed to yield an optimal solution since the set of generated patterns may not contain the necessary patterns required for such a solution. Moreover, the (RM) may not have a feasible integer solution if the generated patterns cannot satisfy (5.34), however the (ARM) problem will always yield a feasible integer solution, regardless of the available patterns.

Solving (RM) or (ARM) with very many patterns can still be computationally challenging. In cases where the number of patterns is very large, solving (RM) or (ARM) may not be possible in acceptable time, and alternative approaches may need to be considered.

5.6.2 Fix Columns

Another solution approach based on Column Generation is to make assumptions about which patterns will run in the final solution using information from the linear relaxation of the master problem. Those patterns whose corresponding variables in the linear relaxation of the master problem have the greatest value are assumed to run, i.e. the corresponding X_p^i variable is set to 1 in (RM) or (ARM). We refer to these as accepted patterns. If the corresponding variable's value in the solution to the linear relaxation was already 1, then setting this variable equal to 1 will not affect the objective value since no change will have been made. If, however, the value of the variable in the solution was less than 1, then setting it equal to 1 may cause the objective value to deteriorate. The process of finding these accepted patterns is applied iteratively, with more and more patterns assumed in the final solution. Whenever the objective value deteriorates by more than a factor of τ since the previous iteration, typically around 1%, the Column Generation procedure is run again to generate new patterns subject to the set of accepted patterns. Since there are only N classes and at most one pattern can be run per class, at most N patterns can be fixed.

Pattern-fixing heuristic

Step 1. Generate the initial set of patterns P by the Column Generation procedure described above. Initialise the empty set of accepted patterns $\hat{P}$. Initialise the iteration counter c to 1.

Step 2. Solve the linear relaxation to (ARM) subject to the patterns P, with the additional assumption that the X_p^i variables corresponding to each pattern in $\hat{P}$ must have value 1. The optimal LP solution is denoted by σ_{c}^*.

Step 3. Find a pattern $\tilde{p}$ from $P \setminus \hat{P}$ whose corresponding X_p^i variable in σ_{c}^* has greatest value. Add $\tilde{p}$ to $\hat{P}$, and remove all patterns from P that are mutually exclusive with $\tilde{p}$.

Step 4. If the objective value of σ^*_{c} is worse than $\sigma^*_{\mathsf{c}-1}$ by more than a factor of τ then generate additional patterns by running the Column Generation procedure described above, with the assumption in (ARM) that the X^i_p variables corresponding to each pattern in $\hat{P}$ must have value 1. Add these new patterns to P.

Step 5. If some students remain unallocated in $\hat{P}$, increment c and return to Step 2.

The only parameter in the pattern fixing heuristic is τ, which determines how much objective value deterioration is required to trigger the subroutine to generate additional patterns. In practice, however, it is useful to impose a time limit and/or iteration limit on both the subroutine to generate more patterns and also the algorithm as a whole. When the time or iteration limit is reached in the pattern generating subroutine, the subroutine terminates with the patterns generated so far. When the time or iteration limit is reached on the whole algorithm between Step 2 and Step 4, the current (ARM) is solved as an IP to produce a solution.

Aside from the time and iteration limit, the pattern generating subroutine normally terminates when no more patterns can be generated, i.e. when there are no more patterns with reduced cost less than zero. In practice, this subroutine can sometimes continue generating a very large number of new patterns for a long time, with reduced costs very close to zero. While having more patterns allows more possible solutions in (ARM), having too many patterns results in a problem with very many variables, which can be computationally difficult to solve. When the number of patterns grows very large, typically above ten thousand, even the linear relaxation can take a minute or more to solve. For this reason, the pattern generating subroutine should also terminate when no patterns can be generated with reduced cost with absolute value greater than some small ϵ.

5.6.3 Student Clustering

Another solution approach based on the Column Generation procedure is to make assumptions about which pairs of students will be assigned to classes together using information from the linear relaxation of the master problem, in a similar way as was described in Section 5.5. Let $p_{j_1,j_2} = \sum_{p \in \tilde{P}_{(j_1,j_2)}} X^{i\,*}_p$, for set of patterns $\tilde{P}_{(j_1,j_2)} \subseteq P$ containing student pair (j_1, j_2) and solution values $X^{i\,*}_p$, which can be interpreted as an indication that students j_1 and j_2 should appear in the same pattern in the final solution. The pairs with highest p_{j_1,j_2} values will be enforced in the solution process. We refer to these as accepted pairs of students. The process of finding accepted pairs of students is applied iteratively, with more and more pairs of students assumed in the final solution. As with the pattern-fixing heuristic from Section 5.6.2, when the objective value deteriorates by more than a factor of τ since the previous iteration, new patterns are generated subject to the pairs of students that have been fixed thus far.

Student-clustering heuristic

Step 1. Generate the initial set of patterns P by the Column Generation procedure described above. Initialise the empty set of accepted student pairs $\hat{S}$. Initialise the iteration counter c to 1.

Step 2. Solve the linear relaxation to (ARM) subject to the patterns P. The optimal LP solution is denoted by σ^*_{c}.

Step 3. Find a pair of students (j_1, j_2) not already in $\hat{S}$ where p_{j_1,j_2} has greatest value. Add this pair of students to $\hat{S}$ and delete all patterns from P that contain j_1 but not j_2, or contain j_2 but not j_1.

Step 4. If the objective value of σ^*_{c} is worse than $\sigma^*_{\mathsf{c}-1}$ by more than a factor of τ then generate additional patterns by the Column Generation procedure described above, with the assumption in each (SP$_i$) that $X_{j_1} = X_{j_2}$ for all pairs $(j_1, j_2) \in \hat{S}$, and increment c.

Step 5. If some students remain unpaired in $\hat{S}$, increment c and return to Step 2. Otherwise, solve (AQP) subject to the pairs specified in $\hat{S}$.

As with the pattern-fixing heuristic described in Section 5.6.2, the only parameter is τ, but it is useful to impose time and iteration limits to the main algorithm and the pattern generating subroutine. It is again also useful to terminate the pattern generating subroutine when no new patterns can be generated whose reduced cost has absolute value greater than some small ϵ.

5.7 Large Neighbourhood Search with Column Generation Heuristics

The Column Generation procedure and three proposed solution approaches mentioned above may not produce good-quality violation-free solutions in acceptable time for larger, more difficult instances of the problem. Therefore, we propose to incorporate the above-mentioned procedures within a fix-and-optimise [67, 130] Large Neighbourhood Search (LNS) [141, 129] framework.

In the iterative fix-and-optimise LNS procedure, the problem is solved for a subset of variables at each iteration, with the remaining variables “fixed” as constants. The neighbourhood to be explored at each iteration is the set of feasible solutions to this restricted problem. At each iteration, a different subset of variables is chosen to be optimised over. Each iteration results in a solution no worse than the previous, since the previous solution must be feasible in the subsequent iteration. This search algorithm can be vulnerable to being trapped in local minima unless neighbourhoods are carefully chosen and sufficiently large.

We propose to solve our problem using a fix-and-optimise LNS procedure, where at each iteration the restricted problem is solved by the Column Generation heuristics described above. Doing so can have several advantages:

- Using the Column Generation heuristics to solve the restricted problem allows much larger neighbourhoods to be explored in the LNS framework, which can help to improve the convergence of the algorithm;
- The Column Generation procedure can be initialised with any compatible already-generated columns from previous iterations; and
- Since the Column Generation heuristics do not guarantee an optimal solution, the LNS procedure has the opportunity to move to solutions with worse objective values, which may allow it to escape from local minima.

At each iteration, the compositions of a set of classes are fixed, while the remaining classes are optimised. The sets of classes are chosen at random, however they are chosen so that the proportion of students they cover fall within a lower and upper bound. These two parameters we denote $\rho^{\min}$ and $\rho^{\max}$. For example, if $\rho^{\min} = 0.2$ and $\rho^{\max} = 0.3$, and $M = 100$, then the set of fixed classes should contain between 20 and 30 students, and the remaining students are assigned to the remaining classes.
Fix-and-optimise Large Neighbourhood Search heuristic incorporating Column Generation

Step 1. Generate an initial solution by randomly assigning students to classes. Students may be assigned to multiple classes to satisfy minimum and maximum class sizes.

Step 2. Pick a random subset of classes containing between $\rho^{\min} \times M$ and $\rho^{\max} \times M$ students to be fixed.

Step 3. Solve the problem for the remaining classes and remaining students using one of the Column Generation heuristics from Section 5.6, storing all columns that are generated in a column pool. Compatible columns from this pool can be used to form the initial set of columns in the Column Generation procedure.

Step 4. If no improvement on the best-known solution has been made after a certain number of iterations, or if a time limit has been reached, terminate the algorithm reporting the best-known solution. Otherwise, return to Step 2.

5.8 Genetic Algorithm Based Matheuristic

Although Genetic Algorithms (GA) are often used in solving the QKP and QMKP [137, 142, 75], these publications on GA are not directly applicable to our problem due to the existence of lower bounds on class sizes. Moreover, the existence of lower and upper bounds on class sizes renders classical operations of crossover and mutation inefficient, i.e. both operations produce too many infeasible solutions. The computational experiments indicated that the common approach of introducing a penalty for infeasibility does not adequately address the problem of infeasibility. These observations lead to the development of a matheuristic—presented in this chapter—which is an amalgamation of GA and IP. This matheuristic was compared with the other approaches described above.

In the developed version of GA, feasible solutions in the initial population are generated using a two-step procedure. First, we decide which classes will be run, and how many students of each type will be in each class by solving an IP. Define variables $\bar{X}_{i,k} \in \mathbb{Z}^+$, where $\mathbb{Z}^+$ is the set of nonnegative integers, to be the number of students of type k to assign to class i. Any feasible solution to the following IP describes a feasible allocation of anonymous students of each type to each class:

$$\text{(FP) min:} \quad \sum_{i=1}^{N}\sum_{k=1}^{K} \gamma_{i,k}\bar{X}_{i,k} \tag{5.43}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \bar{X}_{i,k} = |F_k| \qquad k = 1, \dots, K \tag{5.44}$$

$$a_i Z_i \le \sum_{k=1}^{K} \bar{X}_{i,k} \le b_i Z_i \quad i = 1, \dots, N \tag{5.45}$$

$$p_i Z_i \le \sum_{k=1}^{K} Y_{i,k} \le q_i Z_i \quad i = 1, \dots, N \tag{5.46}$$

$$\bar{X}_{i,k} \le |F_k| \times Y_{i,k} \qquad i = 1, \dots, N; k = 1, \dots, K \tag{5.47}$$

$$Y_{i,k} \le \bar{X}_{i,k} \qquad i = 1, \dots, N; k = 1, \dots, K \tag{5.48}$$

$$\bar{X}_{i,k} \in \mathbb{Z}^+ \qquad i = 1, \dots, N; k = 1, \dots, K \tag{5.49}$$

$$Y_{i,k} \in \{0, 1\} \qquad i = 1, \dots, N; k = 1, \dots, K \tag{5.50}$$

$$Z_i \in \{0, 1\} \qquad i = 1, \dots, N \tag{5.51}$$

The constraints (5.44) express the requirement that the total number of students assigned for each type is equal to the total number of students of that type. The constraints (5.45) and (5.46), analogous to (5.3) and (5.4), express the requirements that the number of students and student types, respectively, must fall within the allowed ranges for each running class. The constraints (5.47) and (5.48), analogous to (5.5) and (5.6), express the requirement that students can only be assigned to classes if their type is also assigned, and that types can only be assigned to classes if at least one student of that type is also assigned. The constants $\gamma_{i,k}$ are chosen randomly each time (FP) is solved, enabling a diverse range of solutions to be produced.

While the feasibility problem was shown to be NP-complete in Section 5.3, the instances of (FP) that arise from our real-world-inspired test cases can be solved quickly. For all test cases, including all cases where CPLEX could not solve the original QP problem, CPLEX solved (FP) model in under a second.

Next, students of each type are optimally assigned to classes according to the numbers obtained from the solution to (FP). This is possible in polynomial time, since the problem of assigning students to classes in fixed numbers can be represented by a minimum cost network flow model. In the network, there is one node for each student, and one node for each student type that is assigned to each class. For example, if all students assigned to a class are of the same type, then there is only one node for that class. If k different types of students are assigned to the class, then the class is represented by k nodes; one for each type. Directed arcs connect the source to each student node with lower bounds on flow equal to one, and directed arcs connect the nodes from each student type node in each class to the sink with upper bound on flow equal to the number of students of the respective type in that class. Student nodes are connected to all nodes of their type by directed arcs with unrestricted flow, and with flow cost equal to the cost of assigning that student to the corresponding class.

The selection operator we propose to use is the so-called `n`-tournament selection operator [135]. The `n`-tournament selection operator chooses a solution by randomly sampling `n` solutions from the population, and choosing one based on some criterion. Most often, the criterion is the best objective value.

The crossover operator, which addresses the challenge imposed by the existence of lower and upper bounds on class sizes, is defined as follows. As with conventional crossover operators, the designed crossover operates on two solutions, referred to as parents. The results of the designed crossover is a single solution, referred to as a child. As with the procedure that generates solutions for the initial population, the crossover operator involves two stages. First, (FP) is solved to determine which classes should be run in the child, and the number of students of each type in each class, with the additional restriction that those classes that are run in both parents must run in the child, and those classes that are not run in either parent will not run in the child.

Next, students are assigned to classes in numbers specified by the numbers obtained from the solution to (FP). Here, the additional restriction is imposed that those students that are assigned to the same class in both parents must be assigned to that class in the child, and any student whose classes in one or both of the parents are running in the child must be assigned to one of those classes in the child. The remaining students are assigned according to the minimum cost network flow approach discussed earlier.

The complete GA algorithm is as follows:

GA-based Matheuristic:

Step 1. Initialise the solution pool $\mathscr{P}$, and the iteration counter $k := 1$.

Step 2. Solve (FP) given random coefficients $\gamma_{i,k}$. Construct a solution to the studied problem by randomly assigning students to classes according to the numbers in the obtained solution to (FP). Add the constructed solution to $\mathscr{P}$. If the $\mathscr{P}$ has not yet reached the required size, repeat Step 2.

Step 3. Pick `n` solutions at random from $\mathscr{P}$, and identify a solution with the worst objective value. Delete this solution from $\mathscr{P}$.

Step 4. Pick `n` solutions at random from $\mathscr{P}$, and identify a solution with the best objective value. Denote this solution $\mathscr{A}$.

Step 5. Pick `n` solutions random from $\mathscr{P}$, and identify a solution with the best objective value. Denote this solution $\mathscr{B}$.

Step 6. Find the optimal solution F^* to (FP) given random coefficients $\gamma_{i,k}$, with the additional assumptions that $Z_i = 0$ for all classes i that are not run in either $\mathscr{A}$ or $\mathscr{B}$, and $Z_i = 1$ for all classes i that are run in both $\mathscr{A}$ and $\mathscr{B}$. Add to $\mathscr{P}$ a new solution constructed by assigning students to classes one at a time as follows:

i. Any student that is assigned to class i in both $\mathscr{A}$ and $\mathscr{B}$ should be assigned to class i in the constructed solution if there is remaining space in the class.

ii. Any student that is assigned to classes $i_{\mathscr{A}}$ in $\mathscr{A}$ and $i_{\mathscr{B}}$ in $\mathscr{B}$, where $i_{\mathscr{A}}$ and/or $i_{\mathscr{B}}$ are running in F^*, should be assigned to either class $i_{\mathscr{A}}$ or $i_{\mathscr{B}}$ in the constructed solution, as long as the remaining capacity of the class permits.

iii. Any remaining students are assigned to the remaining spaces in the constructed solution at random, according to the numbers specified in F^*.

Step 7. Increment the iteration counter k. If k has exceeded the iteration limit, or if the total time limit has been reached, then terminate the algorithm returning a solution from $\mathscr{P}$ with the best objective value. Otherwise, return to Step 3.

5.9 Computational Results

Using typical training data from an Australian electricity distributor as a template, we randomly generated a set of 27 of test cases[1]. Each test case had between 512 and 2048 students across 40 student types, and between 32 and 128 classes.

[1]All test cases and solution files are available for download from `https://goo.gl/q5Z1cx`.

Case	Classes	Students	Density	Case	Classes	Students	Density	Case	Classes	Students	Density
01	32	512	L	10	64	512	L	19	128	512	L
02	32	512	M	11	64	512	M	20	128	512	M
03	32	512	H	12	64	512	H	21	128	512	H
04	32	1024	L	13	64	1024	L	22	128	1024	L
05	32	1024	M	14	64	1024	M	23	128	1024	M
06	32	1024	H	15	64	1024	H	24	128	1024	H
07	32	2048	L	16	64	2048	L	25	128	2048	L
08	32	2048	M	17	64	2048	M	26	128	2048	M
09	32	2048	H	18	64	2048	H	27	128	2048	H

Table 5.1: The number of classes (Classes), number of students (Students), and density (Density) for the 27 test cases (Case).

The minimum and maximum student capacities for the classes in each test case were chosen such that the test cases fall into one of three density groups: low density (`L`), where approximately 25% of the classes will be needed and the rest will remain empty; medium density (`M`), where approximately 50% of the classes will be needed and the rest will remain empty; and high density (`H`), where approximately 75% of the classes will be needed and the rest will remain empty.

We used IBM ILOG CPLEX 12.6.3.0 64-bit on an Intel i7-4790K quad-core 4.00Ghz system with 16GB of RAM, running Windows 10 Professional. Our code was written in C#, and interacted with CPLEX using the IBM ILOG Concert API. We used default CPLEX settings with the exception of time limits. The GA-based matheuristic is probabilistic, therefore we ran it on each of the test cases 10 times. The pseudorandom number generator we used was the MT19937 Mersenne Twister [108]. For the weighted objective function, we used $\alpha = \beta = 1$. We determined the average amount of time taken for the LR-based heuristic, and permitted the same amount of time for the GA-based matheuristic and for attempting to solve the QP model directly. Similarly, we determined the average amount of time taken for the CG-based heuristic, and permitted the same amount of time for the fix-and-optimise LNS heuristic.

The main properties of the test cases are outlined in Table 5.1. The columns are as follows: the test case identifier (Case), the number of classes N (Classes), The number of students M (Students), and the density groups (Density).

The number of variables (columns) and constraints (rows) in the (QP) and (LQP) models for each test case is given in Table 5.2. The size of the models is not dependent on the test case density, therefore test cases with the same number of classes and students, but with differing densities, are grouped together. The columns are as follows: the test case identifiers (Cases), the number variables in the corresponding (QP) model (Cols_{QP}), the number constraints in the corresponding

Cases	Cols_{QP}	Rows_{QP}	Cols_{IP}	Rows_{IP}
`01, 02, 03`	17697	18304	42656	93184
`04, 05, 06`	34081	35200	59040	110080
`07, 08, 09`	66849	68992	91808	143872
`10, 11, 12`	35393	36096	85312	185856
`13, 14, 15`	68161	69376	118080	219136
`16, 17, 18`	133697	135936	183616	285696
`19, 20, 21`	70785	71680	170624	371200
`22, 23, 24`	136321	137728	236160	437248
`25, 26, 27`	267393	269824	367232	569344

Table 5.2: The number of columns (Cols) and rows (Rows) for the QP model (QP) and LQP model (IP) for the 27 test cases (Cases).

(QP) model (Rows_{QP}), the number variables in the corresponding (LQP) model (Cols_{QP}), and the number constraints in the corresponding (LQP) model (Rows_{QP}). For these test cases, the linearised models (LQP) have, on average, roughly 1.84 times as many variables, and roughly 3.46 times as many constraints, as their (QP) counterparts.

For the Lagrangian Relaxation procedure, the starting ϵ value is commonly taken to be $\epsilon_1 = 2$ [72]. For our problem, however, we found the procedure did not perform well with this value, especially for the larger test cases where the best bound remained at zero even after hours and several hundred iterations. During our initial computational experiments, we found that $\epsilon_1 = 0.05$ with $\tilde{\epsilon} = 2$ produced much better performance for the Lagrangian Relaxation procedure across all our test cases, however for the purposed of our LR-based heuristic, it is better for the Lagrangian Dual procedure to generate a large number of diverse solutions than to have fast convergence with a small set of solutions. We therefore used a starting value of $\epsilon_1 = 0.1$ and set $\tilde{\epsilon}$ to 10.

Table 5.3 shows the bounds provided by linear relaxation of (LQP), by Lagrangian Relaxation, and by Column Generation. The columns are as follows: the test case identifier (Case), the linear relaxation bound (LinRel), the Lagrangian Relaxation bound (LagRel), and the Column Generation bound (ColGen). The Lagrangian Relaxation provided a lower bound to the objective value of the underlying problem that was between 1% and 132%, with an average of 36%, higher than the linear relaxation bound. The bound provided by Column Generation, i.e. the objective value obtained by solving the reduced master problem subject to the generated columns, was between 14% and 625%, with an average of 314%, higher than the bound provided by the linear relaxation of (LQP).

It is important to note that for a minimisation problem, the bound provided by the Lagrangian

Case	LinRel	LagRel	ColGen
01	25035.50	26143.16	28558.89
02	12097.10	13628.89	15129.14
03	12666.40	14002.61	15942.71
04	62811.90	63444.30	85524.05
05	30338.50	31758.52	61447.54
06	22712.00	24404.62	45659.07
07	93526.70	94673.71	240087.11
08	62690.60	64573.21	264742.06
09	49709.40	51102.07	168549.10
10	7258.90	9649.33	12497.39
11	5102.90	7566.36	10056.28
12	3662.90	5325.06	6562.87
13	21060.30	23194.41	53693.10
14	10892.80	13587.90	28465.51
15	11109.40	14069.49	24793.17
16	43627.00	45856.09	102458.21
17	24609.50	27704.03	116886.45
18	18517.30	22071.88	83270.76
19	2812.70	5666.28	8382.02
20	1564.40	3632.29	6060.63
21	1666.50	2949.20	3922.38
22	7232.50	10842.10	28291.06
23	3917.00	7765.47	18342.05
24	2952.60	6476.81	16040.31
25	20935.30	24713.55	130912.61
26	11147.10	15749.39	65753.88
27	6998.30	11906.54	39411.62

Table 5.3: The lower bound provided by the linear relaxation of LQP (LinRel), Lagrangian Relaxation (LagRel), and Column Generation (ColGen) for the 27 test cases (Case).

Dual converges from below, whereas the bound provided using the Column Generation procedure converges from above. The consequence is that a partially converged solution to the Lagrangian Dual will provide a bound that is less tight than one from a fully converged one, and may even be less tight than the bound provided by the linear relaxation, but still serves as a valid lower bound on the optimal objective value. On the other hand, a partially converged solution to the reduced master problem, i.e. one where there exist more columns with negative reduced cost that could be generated, will give a bound that is higher, and may even be higher than the optimal solution to the underlying problem. This should be kept in mind when interpreting the results in Table 5.3, especially given that the imposed termination conditions may lead to the procedures terminating early.

Table 5.4 shows the time required, in seconds, to compute the lower bounds shown in Table 5.3. The columns are identical to those in Table 5.3. The Lagrangian Relaxation procedure took between 2.5 and 816, with an average of 103, times longer than the linear relaxation. The Column Generation procedure took between 2.9 and 2317, with an average of 170, times longer than the linear relaxation. Both the Lagrangian Relaxation and Column Generation procedures have the added benefit that each iteration produces one feasible partial solution for each class.

For the LR-based heuristic, we investigate ρ values corresponding to 10%, 20%, and 30% of the total population size, and we investigate the effect where cliques with more than two vertices are either permitted or are forbidden. In all cases, the time limit, in seconds, to solve the AQP model was equal to the number of students. This time limit was reached for all test cases. Table 5.5 shows the objective values of the solutions produced using the LR-based heuristic. The columns are as follows: The test case identifier (Case), ρ set to 10% of the population size with larger cliques permitted (10% C), and with larger cliques forbidden (10%), ρ set to 20% of the population size with larger cliques permitted (20% C), and with larger cliques forbidden (20%), ρ set to 30% of the population size with larger cliques permitted (30% C), and with larger cliques forbidden (30%).

The (10% C) produced the best solutions for 5 test cases, (10%) for 4 test cases, (20% C) for 4 test cases, (20%) for 5 test cases, (30% C) for 5 test cases, and (30%) for 4 test cases. The runs where larger cliques were permitted produced best solutions for 14 test cases, and produced best solutions for 13 test cases where larger cliques were forbidden. When looking at the ratio between the objective value of the solution produced given certain parameters to the objective value of the best solution produced, (10%) performed the worst, with a ratio of about 1.996, while (30%) performed the best, with a ratio of about 1.248. Overall, the ratio where large cliques were permitted was about 1.649, and the ratio was about 1.609 when large cliques were forbidden. According to these results, it appears that the selecting the largest of the tested values of ρ and forbidding large cliques had the best outcome overall. This is most likely because imposing more student pairs allowed the solver to find solutions to

Case	LinRel	LagRel	ColGen
01	13	627	148
02	19	1036	566
03	24	956	3991
04	29	857	387
05	34	1302	808
06	27	1729	3718
07	64	1544	768
08	47	1726	2090
09	75	1973	5510
10	152	2023	729
11	16	1809	2205
12	10	1612	9481
13	317	1534	928
14	259	2539	2928
15	80	2691	1608
16	680	2197	4286
17	511	3369	16066
18	369	4040	1194
19	29	2420	6613
20	3	2447	6952
21	4	2421	604
22	630	3481	2138
23	91	4084	6429
24	10	4610	1248
25	1744	4370	5352
26	572	5912	11854
27	257	5923	4063

Table 5.4: The time, in seconds, required to compute the lower bound by means of linear relation of LQP (LinRel), Lagrangian Relaxation (LagRel), and Column Generation (ColGen) for the 27 test cases (Case).

Case	10% C	10%	20% C	20%	30% C	30%
01	27728.0	28349.5	28260.0	**27228.5**	30125.0	28181.5
02	15512.5	**15510.5**	15667.0	16607.0	15952.5	15939.0
03	**15901.5**	16493.5	16027.5	16368.5	17407.0	16821.5
04	156498.0	135303.5	149792.5	68155.5	**65624.5**	152989.0
05	113635.5	102934.5	161448.0	34396.5	37772.5	**33468.5**
06	26498.5	28689.5	**25914.0**	26448.0	29558.0	26252.0
07	284443.5	264407.0	337967.0	**96777.0**	310202.5	97785.5
08	71145.5	207659.5	178323.5	235653.5	**70464.5**	205008.0
09	52939.5	161932.5	154799.5	149034.5	**52773.5**	53392.5
10	51398.0	56282.0	83387.5	69566.5	**46425.0**	64256.0
11	54987.0	49014.0	**13425.0**	17713.0	14943.5	15080.0
12	13189.0	11561.0	47952.5	10607.5	**8650.5**	9955.5
13	149670.5	111301.5	121775.5	112428.5	160489.5	108433.0
14	16297.0	16352.5	**15953.0**	63331.0	16090.0	16286.0
15	21600.5	29704.0	24326.0	**20903.0**	24969.0	21562.5
16	193459.0	**184879.5**	224889.0	218679.5	200310.5	278620.0
17	230104.0	**176429.0**	241710.0	250492.0	204366.0	272560.0
18	**29946.0**	320669.5	30025.5	112569.0	269609.5	34162.5
19	60118.5	65537.0	66608.0	57081.5	59630.0	**18540.5**
20	52531.0	59937.0	77608.5	**35564.5**	40345.0	38385.5
21	**85501.5**	86071.0	86321.0	96603.0	96240.0	90446.5
22	107470.5	76535.0	**67545.0**	117384.0	83727.5	100723.5
23	**102924.5**	123642.0	116893.5	149978.0	130494.0	115310.0
24	84763.5	90099.0	160378.5	**74002.0**	92816.5	78813.0
25	245627.0	**186002.5**	190062.0	218624.0	240426.0	243105.0
26	291573.5	331949.0	310532.5	318246.0	285952.5	**243200.5**
27	**247332.5**	320109.5	260695.0	274399.0	276121.5	312599.0

Table 5.5: Objective values from the LR-based heuristic using 10% of students paired with and without large cliques (10% C) and (10%), 20% of students paired with and without large cliques (20% C) and (20%), and 30% of students paired with and without large cliques (30% C) and (30%) for the 27 test cases (Case).

the problem with imposed pairs faster, and because forbidding large cliques would result in a greater variety of students present in imposed pairs for a given value of ρ.

Table 5.6 shows the objective values obtained my means of the Column Generation-based heuristic making use of the Reduced Master Heuristic method (MP) described in Section 5.6.1, the column fixing method (PF) described in Section 5.6.2, and the student clustering method (SC) discussed in Section 5.6.3. The column fixing approach produced solutions with objective values that were between 6.8% and 127%, with an average of 70.6%, compared to those by the reduced master heuristic. The student clustering approach produced solutions with objective values that were between 6.9% and 100%, with an average of 53%, compared to those by the reduced master heuristic.

Table 5.7 shows the time, in seconds, required by the Column Generation-based heuristic to produce the solution by each of the three methods discussed. The columns are identical to those in Table 5.6. The column fixing approach required between 10% and 115%, with an average of 82.9%, the time required by the reduced master heuristic. The student clustering approach required between 60% and 329%, with an average of 121%, the time required by the reduced master heuristic. The column fixing approach required less time on average, and produced better solutions on average than the reduced master heuristic. The student clustering approach required slightly more time on average, but produced significantly better solutions on average than the reduced master heuristic.

Since the student clustering approach performed the best on average, we chose to test the fix-and-optimise LNS heuristic, whereby the student clustering approach is used Step 3 of the procedure described in Section 5.7. The LNS would terminate if no improvement was made in 5 iterations, and $\rho^{\min}$ and $\rho^{\max}$ were taken to be 0.2 and 0.3 respectively. Table 5.8 shows the objective values (LNS) of the solutions produced by the fix-and-optimise LNS heuristic, making use of the student clustering heuristic for each of the 27 test cases (Case). The fix-and-optimise LNS approach produced solutions with objective values that were between 5.6% and 98.8%, with an average of 39.3%, compared to those by the reduced master heuristic.

For the GA-based matheuristic, we explored the tournament size for the `n`-tournament selection operator, $\texttt{n} = \{N, 5N, 10N\}$, with a fixed population size of $100N$. In all cases, the GA-based matheuristic was permitted to run for an arbitrary number of iterations, but had a time limit equal to the average amount of time taken for the LR-based heuristic for that test case.

We also tested variations on the `n`-tournament selection operators, in which parent solutions are selected as inputs for the crossover operator. The three variations for selecting the parents were

1. `Best`: the solution with the best objective value from a tournament of size `n`;
2. `Random`: a random solution[2] from a tournament of size `n`; and

[2]Note that `Random` is equivalent to selecting a random solution from the entire population.

Case	MP	PF	SC
01	28170.5	29263.0	**27309.5**
02	84160.0	75338.5	**69732.5**
03	85005.5	83019.0	**17930.5**
04	198129.0	196931.5	**194957.0**
05	193497.0	129093.0	**122432.5**
06	156389.0	138656.5	**27955.5**
07	**98718.0**	**98718.0**	**98718.0**
08	282679.0	80060.0	**68597.0**
09	262715.5	169529.0	**53137.0**
10	75302.5	62478.5	**15095.0**
11	92955.0	58576.5	**12270.5**
12	122791.0	**8303.0**	8466.0
13	145808.5	100128.5	**98355.0**
14	104104.5	103981.5	**94979.0**
15	178150.0	26021.0	**19842.0**
16	278203.5	**222386.5**	246993.0
17	268233.5	169765.5	**167142.0**
18	34304.5	27570.5	**26653.0**
19	57579.0	**12251.5**	12739.5
20	39603.5	19208.0	**16120.0**
21	60774.0	77234.5	**5884.0**
22	153938.0	90386.5	**24991.0**
23	56289.5	18971.5	**17789.5**
24	14079.5	12561.5	**12383.0**
25	295561.5	**202205.0**	295401.5
26	34731.0	26428.0	**25599.5**
27	24201.5	**20870.5**	20927.5

Table 5.6: Objective values from the Column Generation-based heuristic obtained by using the reduced master heuristic (MP), the column fixing (PF), and the student clustering (SC) methods for the 27 test cases (Case).

Case	MP	PF	SC
01	2629	**321**	1672
02	4870	**1487**	5124
03	12397	**9207**	12009
04	**387**	438	505
05	6140	**1656**	6060
06	12892	**11921**	13227
07	**767**	783	780
08	**2092**	2286	3548
09	14769	**14594**	15199
10	3657	**1070**	3675
11	6520	**4543**	6810
12	18201	**15230**	17737
13	**928**	1063	3065
14	8453	**7790**	8086
15	**9733**	11004	10271
16	**4286**	4310	4326
17	**16069**	17244	23116
18	11332	**7225**	11642
19	9785	**7487**	10085
20	11656	**10643**	11831
21	8815	**5692**	9259
22	**2138**	2224	5975
23	12653	**9177**	12247
24	**10291**	10826	10321
25	**5352**	5397	5550
26	**11857**	12851	18454
27	15842	**14236**	15835

Table 5.7: The time, in seconds, taken by the Column Generation-based heuristic obtained by using the reduced master heuristic (MP), the column fixing (PF), and the student clustering (SC) methods for the 27 test cases (Case).

Case	LNS	Case	LNS	Case	LNS
01	27265.0	10	14854.0	19	10127.0
02	15227.5	11	11830.5	20	14014.5
03	15505.5	12	6922.0	21	5287.0
04	64628.5	13	98298.0	22	23781.0
05	32335.5	14	16575.0	23	17535.5
06	25504.5	15	19473.5	24	12124.0
07	97626.0	16	220319.5	25	198001.0
08	65422.5	17	41449.5	26	25279.0
09	52177.0	18	26593.5	27	19861.0

Table 5.8: Objective values from the fix-and-optimise LNS heuristic (LNS) using the student clustering methods for the 27 test cases (Case).

3. `Worst`: the solution with the worst objective value from a tournament of size `n`.

The selection operator is invoked twice in order to select two parents. We compared the relative performance of selection pairs {`Best`, `Best`}, {`Best`, `Random`}, and {`Best`, `Worst`}.

In all cases tested, {`Best`, `Best`} produced consistently superior results. {`Best`, `Random`} produced final solutions that were, on average, 11.752% worse than {`Best`, `Best`}, and {`Best`, `Worst`} produced final solutions that were, on average, 22.727% worse than {`Best`, `Best`}. Since {`Best`, `Best`} produced consistently superior solutions, we do not report the results of {`Best`, `Random`} and {`Best`, `Worst`} in the remainder of the chapter.

Table 5.9 shows a comparison of the results of the GA-based matheuristic using different tournament sizes. The columns are as follows: the test case identifier (Case), followed by the minimum (Min), average (Avg), and maximum (Max) objective value reported across the 10 runs of the GA-based matheuristic using the different population sizes. For each test case, the parameter set that produced the best results on average are indicated. The results clearly show that using larger tournament sizes results for the GA-based matheuristic produces the best results overall. Using $n = 5N$ results in final solutions with objective values that are, on average, 6.98% worse than $n = 10N$, and using $n = N$ results in final solutions with objective values that are, on average, 13.81% worse than $n = 10N$. Using larger tournament sizes increases the likelihood that better solutions will be chosen from the population, however the selection procedure becomes slightly more time intensive. Using a very large tournament size may not always be advisable, however, as it may decrease the genetic diversity in the population due to the same few good solutions being selected in most iterations for crossover.

Table 5.10 shows the solution quality of the LR-based heuristic, CG-based heuristic, GA-based

Case	N Min	N Avg	N Max	$5N$ Min	$5N$ Avg	$5N$ Max	$10N$ Min	$10N$ Avg	$10N$ Max
01	91242.0	104088.8	126507.0	89317.0	100309.4	107197.5	70969.0	**96212.6**	106917.5
02	80001.5	87148.5	101027.0	78458.0	84956.9	97735.5	71913.5	**81374.2**	87640.0
03	67970.5	73771.3	83557.5	64769.5	**67121.1**	68708.0	57838.5	67702.6	73492.5
04	186384.5	212467.4	243732.0	163696.5	186075.5	199798.0	134469.0	**172069.1**	207858.0
05	152188.5	181547.4	209962.5	166769.0	173756.5	181388.0	127361.5	**158909.7**	187397.0
06	139030.0	152711.7	176796.5	126221.0	144869.3	156351.0	114863.5	**132484.3**	150388.5
07	339135.5	431919.6	483614.0	373937.0	412425.4	450458.0	292499.0	**362869.7**	455445.5
08	345296.5	379839.0	416811.0	332903.5	355297.7	374316.0	270846.5	**336157.9**	373108.0
09	229166.5	247770.2	258267.5	207775.5	**224390.2**	260741.5	200377.5	238324.4	272314.0
10	65488.0	76016.5	87723.0	54077.5	72858.1	81155.5	54570.5	**70972.1**	84968.5
11	62119.5	67928.3	77767.0	58894.0	63194.5	68366.5	51782.5	**59756.5**	67927.5
12	67654.0	77697.6	85777.0	68634.0	72986.8	76799.0	63146.0	**69697.7**	77014.5
13	127503.5	159373.0	191844.0	134490.5	**145765.4**	159078.0	127183.0	150736.9	184762.5
14	107942.0	121064.8	133317.5	99062.0	110802.5	126008.0	86389.0	**104101.8**	117581.0
15	113694.5	133210.4	150128.5	120876.5	133918.6	143327.5	95478.5	**119608.7**	153496.5
16	276536.5	328791.5	372141.0	275411.0	306524.9	339932.5	241305.0	**283971.6**	312162.5
17	218382.0	268251.9	296087.5	224007.0	246367.0	275521.0	204612.0	**233492.8**	262544.0
18	238596.5	262473.8	309772.5	231984.0	263037.9	284629.0	207467.0	**246838.2**	287351.5
19	55876.5	69762.7	82040.0	44653.5	64460.8	85016.5	48913.5	**57696.7**	67454.0
20	48163.5	53883.0	66689.0	48211.0	52363.9	57575.0	39044.0	**45617.2**	55981.5
21	41338.0	75161.5	92132.5	37948.0	57068.9	82655.5	40752.0	**53276.6**	88666.5
22	129922.5	147397.9	163537.0	135758.0	152261.8	175112.5	101933.0	**130587.5**	150005.0
23	119509.0	135456.7	152733.5	106466.5	130127.5	145559.5	94491.0	**120684.6**	144671.0
24	111839.0	137977.1	170726.5	114220.0	124454.0	134612.5	76647.5	**110887.4**	130358.0
25	260619.5	305671.7	350736.0	261987.0	291420.2	324293.5	222027.0	**275145.5**	340572.0
26	201494.5	252628.5	284579.0	212065.0	233238.4	251600.0	188994.0	**228316.5**	249142.0
27	124748.0	223403.6	259918.5	119727.0	211191.7	241518.0	136182.5	**181289.6**	211825.5

Table 5.9: Minimum (Min), Average (Avg), and Maximum (Max) objective value for GA-based matheuristic with tournament sizes N, $5N$, and $10N$ for the 27 test cases (Case).

Case	LR Max	LR Avg	LR Min	CG Max	CG Avg	CG Min	GA Max	GA Avg	GA Min	LNS	AQP
01	28350	28312	28182	28171	28248	27310	104089	100204	96213	**27265**	86541
02	15511	15865	15939	84160	76410	69733	87149	84493	81374	**15228**	86956
03	16494	16503	16822	85006	61985	17931	73771	69532	67703	**15506**	58051
04	135304	121394	152989	198129	196673	194957	212467	190204	172070	**64629**	-
05	102935	80609	33469	193497	148341	122433	181547	171405	158910	**32336**	140779
06	28690	27227	26252	156389	107667	27956	152712	143355	132484	**25505**	81186
07	264407	231930	97786	98718	98718	98718	431920	402405	362870	**97626**	-
08	207660	161376	205008	282679	143779	68597	379839	357098	336158	**65423**	-
09	161933	104145	53393	262716	161794	53137	247770	236828	238324	**52177**	-
10	56282	61886	64256	75303	50959	15095	76017	73282	70972	**14854**	65037
11	49014	27527	15080	92955	54601	12271	67928	63626	59757	**11831**	68323
12	11561	16986	9956	122791	46520	8466	77698	73461	69698	**6922**	60528
13	111302	127350	108433	145809	114764	98355	159373	151958	150737	**98298**	116016
14	16353	24052	16286	104105	101022	94979	121065	111990	104102	**16575**	138668
15	29704	23844	21563	178150	74671	19842	133210	128913	119609	**19474**	109097
16	184880	**216806**	278620	278204	249194	246993	328792	306429	283972	220320	257973
17	176429	229277	272560	268234	201714	167142	268252	249371	233493	**41450**	161866
18	320670	132830	34163	34305	29509	26653	262474	257450	246838	**26594**	190474
19	65537	54586	18541	57579	27523	12740	69763	63973	57697	**10127**	49032
20	59937	50729	38386	39604	24977	16120	53883	50621	45617	**14015**	54660
21	86071	90197	90447	60774	47964	5884	75162	61836	53277	**5287**	96165
22	76535	92231	100724	153938	89772	24991	147398	143416	130588	**23781**	134996
23	123642	123207	115310	56290	31017	17790	135457	128756	120685	**17536**	135129
24	90099	96812	78813	14080	13008	12383	137977	124440	110887	**12124**	116919
25	186003	220641	243105	295562	264389	295402	305672	290746	275146	**198001**	-
26	331949	296909	243201	34731	28920	25600	252629	238061	228317	**25279**	186099
27	320110	281876	312599	24202	22000	20928	223404	205295	181290	**19861**	241981

Table 5.10: Poorest (Max), average (Avg), and best (Min) objective values from the LR-based heuristic (LR), the CG-based heuristic (CG), the GA-based matheuristic (GA), the LNS heuristic (LNS), and QP model (QP) for the 27 test cases (Case).

matheuristic, fix-and-optimise LNS-based heuristic, and straightforward approach of solving (QP). To solve (QP), CPLEX was allowed the same time limit the GA-based matheuristic was allowed, which was equal to the average amount of time taken by the LR-based heuristic. The columns are as follows: the test case identifier (Case), the objective value from the LR-based heuristic using the parameter set that performed the poorest overall (LR Max), the average objective value using the LR-based heuristic (LR Avg), the objective value from the LR-based heuristic using the parameter set that performed the best overall (LR Min), the objective value from the CG-based heuristic using the strategy that performed the poorest overall (CG Max), the average objective value using the CG-based heuristic (CG Avg), the objective value from the CG-based heuristic using the strategy that performed the best overall (CG Min), the objective value from the GA-based matheuristic using the parameter set that performed the poorest overall (GA Max), the average objective value using the GA-based matheuristic (GA Avg), the objective value from the GA-based matheuristic using the parameter set that performed the best overall (GA Min), the objective value from the fix-and-optimise LNS heuristic (LNS), and the objective value using the straightforward approach of solving (QP).

Overall, the fix-and-optimise LNS-based heuristic incorporating Column Generation performed the best, followed by the Column Generation-based approaches, then the Lagrangian Relaxation-based approach, and finally the Genetic Algorithm-based approach. The straight forward approach of solving the QP model did not perform well, and failed to produce any feasible solutions for five of the test cases.

5.10 Conclusions

This chapter studied a partitioning problem. One application of this problem, and the motivation for this research, is the problem of class formation for training and retraining sessions at large electricity distributors. A proof of NP-hardness in the strong sense for the considered problem is presented.

Four different solution approaches were developed. The first is based on the Quadratic Multiple Knapsack formulation and Lagrangian Relaxation. The second is based on Column Generation, and three approaches are proposed to produce integer feasible solutions: a reduced master heuristic, a column fixing heuristic, and a student clustering heuristic. The third is based on a fix-and-optimise Large Neighbourhood Search (LNS) that incorporates the Column Generation approach. The fourth is a matheuristic developed as an amalgamation of Genetic Algorithms and Integer Programming.

The four solution approaches, as well as the direct solution of the Quadratic Multiple Knapsack formulation, are validated and compared by means of computational experimentation. The test cases were randomly generated using typical training data from an Australian electricity distributor as a template. The fix-and-optimise Large Neighbourhood Search-based approach integrating Column

Generation performs the best overall, followed by the Column Generation-based approach with the student clustering strategy, then by the Lagrangian generation-based approach, and finally the Genetic Algorithm-based matheuristic. Direct application of mathematical programming did not perform well, and failed to produce any feasible solutions for five of the test cases in the allotted time.

This chapter considered the Lagrangian Relaxation-based heuristic and Genetic Algorithm-based matheuristic separately, however there are several papers in the literature that report success from hybridising Lagrangian Relaxation with Genetic Algorithms, such as [122, 162, 118]. Future research should investigate whether such Lagrangian Relaxation and Genetic Algorithm hybrid approaches can produce good results for this problem.

Chapter 6

Timetabling of Practice Placements

Contents

The research in this chapter was presented at the 11th International Conference on the Practice and Theory of Automated Timetabling (PATAT) 2016 [126, 38] and included in the refereed conference proceeding as "O. Czibula, H. Gu, and Y. Zinder. Timetabling of workplace training: A combination of mathematical programming and simulated annealing. In *International Conference of the Practice and Theory of Automated Timetabling*, 2016", and was subsequently extended and submitted to the PATAT 2016 Special Issue by the Annals of Operations Research.

Abstract

This chapter is concerned with the problem of scheduling workplace training, which arises in a broad range of organisations, from the hospital placements of nursing students to apprentice training at organisations such as electricity distributors. The problem can be viewed as a generalisation of the Open Shop Scheduling Problem. The chapter discusses the complexity of the considered problem, and presents an optimisation procedure, which is a sequential application of Integer Linear Programming and Simulated Annealing. The chapter also presents a fix-and-optimise Large Neighbourhood Search approach. The effectiveness of the proposed optimisation procedures are demonstrated by computational experiments using data with typical characteristics of real-world problems arising at large electricity distributors. The computational experiments show that the proposed optimisation procedures produce superior solutions on average compared to those from a general purpose MIP solver.

6.1 Introduction

This chapter is concerned with the problem of timetabling workplace training, which arises in a broad range of organisations. This practical training is frequently in the form of a set of so-called "practice placements", sometimes simply referred to as "placements", or as "rotations" [100]. Each practice placement is undertaken by one or more students simultaneously, and is designed to provide them with real-world hands-on experience while under the supervision of a competent instructor.

This training model is encountered in a number of disciplines and in a number of industries. It is perhaps most notable in the area of clinical education, for example in nursing, where students that have completed the theoretical components of their studies must then gain practical experience in a number of areas prior to becoming fully qualified [9, 114].

Another example, and the main motivation for this research, is the problem of creating a placement timetable for apprentices of different types at large electricity distributors. Such organisations may take in hundreds of new apprentices each year, and apprenticeships can take several years to complete. Given the motivation of the research in this chapter, in what follows the discussion is conducted in terms of apprenticeship planning.

The set of placements is partitioned into a number of placement groups. Each placement in a particular group exposes the apprentices to the same core set of skills. Each apprentice has a set of required groups determined by their apprenticeship type, e.g. line worker, cable jointer, electro-mechanic, etc., and each apprentice must therefore complete one placement from each of their required groups in order to complete their practical training. Apprentices of different types may have some overlapping placement groups with one another, however we assume a particular placement will never appear in more than one placement group.

Apprentices require supervision while in a placement; too few apprentices in a placement is impractical as it removes qualified and experienced staff from their regular duties with little benefit to the organisation, and too many apprentices in a placement can be too burdensome for the supervisors. Therefore, at any given time, each placement is permitted to have either no apprentices, or between a minimum and maximum number of apprentices, where these limits are given for each placement. Each apprentice must spend a minimum amount of time in each assigned placement—typically a number of weeks or months—in order to gain the necessary skills and experience. There is no maximum permissible amount of time for time in placements, however apprentices must complete their practice placements by the time their apprenticeships finish.

Although any placement in a particular group provides the same core set of skills, suitability for each apprentice varies from placement to placement. For example, the travel distance for the apprentice is a serious consideration during the assignment process. Suitability and personal preferences for placements are modelled by means of a penalty for each apprentice-placement pair per unit time. The goal is to minimise the total penalty over a given planning horizon.

Apprenticeships at Australia's largest electricity distributor (by number of connections and number of employees) last for four years, where the first 18 months consist of theory lessons, and the remainder is spent in practice placements in office and workshop environments, as well as out in the field. The company claims that the yearly requirement to create placement timetables that are not only feasible, but satisfy as many apprentice preferences for placements as possible, is a significant and recurring challenge.

The remainder of the chapter is organised as follows. Section 6.2 presents the Integer Programming (IP) formulation for the considered problem. Section 6.3 discusses the computational complexity. Section 6.4 presents the first optimisation procedure, consisting of an initial construction stage followed by a Simulated Annealing improvement stage. Section 6.5 presents the second optimisation procedure, consisting of a fix-and-optimise Large Neighbourhood Search heuristic. Section 6.6 presents the results of the computational experimentation. Section 6.7 gives concluding remarks.

6.2 Problem Formulation

Let $\mathcal{N} = \{1, \dots, N\}$ be the set of apprentices, and $\mathcal{M} = \{1, \dots, M\}$ be the set of placements. $\mathcal{M}$ is partitioned into placement groups $\{\mathcal{M}_1, \dots, \mathcal{M}_k\} = \hat{\mathcal{M}}$. Define $G_i \subseteq \hat{\mathcal{M}}$ to be the set of required placement groups for $i \in \mathcal{N}$. The planning horizon is discretised into a number of time periods of equal length indexed from $1, \dots, T$, and define $\mathcal{T} = \{1, \dots, T\}$. Define $X_{i,j,t}$ to be 1 if $i \in \mathcal{N}$ starts $j \in \mathcal{M}$ at or before the beginning of period $t \in \mathcal{T}$, or 0 otherwise. Define $Y_{i,j,t}$ to be 1 if $i \in \mathcal{N}$ completes $j \in \mathcal{M}$ before the beginning of period $t \in \{1, \dots, T+1\}$, or 0 otherwise. In order to facilitate the exposition of the model, we extend the planning horizon by an addition period $T+1$. Without the additional period $T+1$ used for the Y variables, no placement can take place at time T. $Z_{j,t}$ is 1 if $j \in \mathcal{M}$ contains apprentices at $t \in \mathcal{T}$, or 0 otherwise. The constant $c_{i,j}$ is the penalty per unit time of assigning $i \in \mathcal{N}$ to $j \in \mathcal{M}$. The constant l_j is the minimum duration for $j \in \mathcal{M}$. The constants a_j and b_j are the minimum and maximum number of apprentices that $j \in \mathcal{M}$ can hold, respectively. The following Integer Program (IP) describes the problem:

$$\text{(P) min:} \quad \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{t=1}^{T} c_{i,j}(X_{i,j,t} - Y_{i,j,t}) \tag{6.1}$$

$$\text{s.t.} \quad X_{i,j,t} \le X_{i,j,t+1} \qquad i = 1, \dots, N; j = 1, \dots, M; t = 1, \dots, T-1 \tag{6.2}$$

$$Y_{i,j,t} \le Y_{i,j,t+1} \qquad i = 1, \dots, N; j = 1, \dots, M; t = 1, \dots, T \tag{6.3}$$

$$X_{i,j,T} = Y_{i,j,(T+1)} \qquad i = 1, \dots, N; j = 1, \dots, M \tag{6.4}$$

$$X_{i,j,t} \geq Y_{i,j,t} \qquad i = 1, \ldots, N; j = 1, \ldots, M; t = 1, \ldots, T \tag{6.5}$$

$$\sum_{j \in g} X_{i,j,T} = 1 \qquad i = 1, \ldots, N; g \in G_i \tag{6.6}$$

$$\sum_{j=1}^{M} (X_{i,j,t} - Y_{i,j,t}) \leq 1 \qquad i = 1, \ldots, N; t = 1, \ldots, T \tag{6.7}$$

$$\sum_{t=1}^{T} (X_{i,j,t} - Y_{i,j,t}) \geq l_j X_{i,j,T} \qquad i = 1, \ldots, N; j = 1, \ldots, M \tag{6.8}$$

$$\sum_{i=1}^{N} (X_{i,j,t} - Y_{i,j,t}) \geq a_j Z_{j,t} \qquad j = 1, \ldots, M; t = 1, \ldots, T \tag{6.9}$$

$$\sum_{i=1}^{N} (X_{i,j,t} - Y_{i,j,t}) \leq b_j Z_{j,t} \qquad j = 1, \ldots, M; t = 1, \ldots, T \tag{6.10}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M} X_{i,j,1} \geq 1 \tag{6.11}$$

$$X_{i,j,t} \in \{0, 1\} \qquad i = 1, \ldots, N; j = 1, \ldots, M; t = 1, \ldots, T \tag{6.12}$$

$$Y_{i,j,t} \in \{0, 1\} \qquad i = 1, \ldots, N; j = 1, \ldots, M; t = 1, \ldots, T + 1 \tag{6.13}$$

$$Z_{j,t} \in \{0, 1\} \qquad j = 1, \ldots, M; t = 1, \ldots, T \tag{6.14}$$

where the expression $(X_{i,j,t} - Y_{i,j,t})$ results in 1 if apprentice $i \in \mathcal{N}$ is to attend placement $j \in \mathcal{M}$ at $t \in \mathcal{T}$, or 0 otherwise. The expression (6.1) describes the objective, which is to minimise the total assignment penalty. Constraints (6.2) and (6.3) establish the temporal relationships between the $X_{i,j,*}$, $1 \leq * \leq T$, and $Y_{i,j,*}$, $1 \leq * \leq T + 1$, variables, respectively. Constraints (6.4) ensure that placements must finish if they start, and (6.5) ensure that placements must start before they finish. Constraints (6.6) ensure that each apprentice's requirements are satisfied. Constraints (6.7) ensure that no apprentice is required to attend more than one placement at any given time. Constraints (6.8) ensure that apprentices spend the minimum amount of time in their assigned placements, and (6.9) and (6.10) ensure that each placement $j \in \mathcal{M}$ has either 0, or between a_j and b_j apprentices, respectively. The constraint (6.11) reduces the symmetry in the feasible region by forcing at least some assignments to happen at the beginning of the planning horizon. The above formulation was chosen over one with a more 'natural' encoding scheme, such as where $X_{i,j,t}$ is 1 if apprentice i attends placement j at time t or zero otherwise, in order to simplify the model with respect to the variable durations, without gaps, that apprentices can attend placements.

6.3 Complexity of Finding a Feasible Solution

In this section we show that even the problem of detecting the existence of a feasible solution when $T = 1$ and there exists only one placement group is NP-complete. It is easy to see that a feasible solution for the problem with $T = 1$ and a single placement group exists if and only if there is a solution for the following system of two inequalities:

$$a_1x_1 + \ldots + a_Mx_M \leq N \tag{6.15}$$

$$b_1x_1 + \ldots + b_Mx_M \geq N \tag{6.16}$$

where $x_i \in \{0, 1\}$ for all $1 \leq i \leq M$. Let c be a nonnegative integer. Denote by PL(c) the decision problem, requiring to answer whether or not there exists a solution for (6.15) - (6.16), where $a_1, \ldots, a_M$ and $b_1, \ldots, b_M$ satisfy the inequalities

$$b_i - a_i \geq c \qquad \text{for all } 1 \leq i \leq M. \tag{6.17}$$

Observe that c is not part of the input, i.e. the input is only positive integers N, $a_1, \ldots, a_M$ and $b_1, \ldots, b_M$, where all pairs a_i, b_i satisfy (6.17). In other words, all instances of PL(c) satisfy (6.17) for the same c.

In order to prove that, for any nonnegative integer c, PL(c) is NP-complete, consider the following decision problem which will be referred to as PARTITION:
Input: positive integers $u_1, \ldots, u_n$ such that $\sum_{i=1}^{n} u_i$ is even.
Question: does there exists a solution to the equation

$$u_1x_1 + \ldots + u_nx_n = \frac{1}{2}\sum_{i=1}^{n} u_i, \tag{6.18}$$

where $x_i \in \{0, 1\}$ for all $1 \leq i \leq n$?
It is well known that PARTITION is NP-complete [64]

Lemma 3. *For any positive integer c, PARTITION $\propto$ PL(c) and therefore PL(c) is NP-complete.*

Proof. Let $u_1, \ldots, u_n$ be an arbitrary instance of the PARTITION and let c be an arbitrary nonnegative integer. If $c = 0$, then PARTITION $\propto$ PL(c) is achieved by setting $M = n$, $N = \frac{1}{2}\sum_{i=1}^{n} u_i$, and

$$a_i = b_i = u_i \quad \text{for all } 1 \leq i \leq n.$$

If $c > 0$, then the corresponding instance of PL(c) is constructed as follows: $M = n$,

$$a_i = ncu_i \qquad \text{for all } 1 \leq i \leq n$$

$$b_i = a_i + c \qquad \text{for all } 1 \leq i \leq n. \tag{6.19}$$

and

$$N = \frac{nc}{2}\sum_{i=1}^{n} u_i.$$

Supposed that (6.18) holds for some $x'_1, \ldots, x'_n$. Then, by multiplying (6.18) by nc and taking into account that $M = n$,

$$a_1 x'_1 + \ldots + a_M x'_M = N$$

which by virtue of (6.19) gives also (6.16).

Supposed that (6.18) does not have a solution. Consider arbitrary $x'_1, \ldots, x'_n$ such that

$$a_1 x'_1 + \ldots + a_M x'_M \leq N.$$

By dividing by nc and taking into account that $M = n$,

$$u_1 x'_1 + \ldots + u_n x'_n \leq \frac{1}{2} \sum_{i=1}^{n} u_i.$$

Hence,

$$x'_1 + \ldots + x'_n < n. \tag{6.20}$$

Furthermore,

$$u_1 x'_1 + \ldots + u_n x'_n < \frac{1}{2} \sum_{i=1}^{n} u_i,$$

because of the assumption that (6.18) does not have a solution. Since the left-hand side and the right-hand side are integer,

$$\frac{1}{2} \sum_{i=1}^{n} u_i \geq u_1 x'_1 + \ldots + u_n x'_n + 1$$

which, by multiplying by nc and taking into account (6.20), $M = n$ and $N = \frac{nc}{2} \sum_{i=1}^{n} u_i$, gives

$$N \geq a_1 x'_1 + \ldots + a_M x'_M + Mc > a_1 x'_1 + \ldots + a_M x'_M + c(x'_1 + \ldots + x'_M)$$

$$= b_1 x'_1 + \ldots + b_M x'_M.$$

Hence, $x'_1, \ldots, x'_M$ is not a solution to (6.15)-(6.16).

Consequently, (6.18) has a solution if and only if the system (6.15)-(6.16) has a solution. □

6.4 Constructive and Improvement Approach

The problem is very challenging, and the required computational effort for straight-forward solution of (P) grows very rapidly as the problem size increases. Even for very small test cases, commercial IP solvers struggle to solve it on a modern desktop computer. Since real world problems are much larger, typically involving hundreds of apprentices, we propose two solution approaches: ***(i)*** A two-stage approach, consisting of a sequential construction stage to produce an initial feasible solution, followed by an improvement stage based on Simulated Annealing. ***(ii)*** A fix-and-optimise Large Neighbourhood Search matheuristic. The details of these approaches are given in the remainder of this section.

6.4.1 Sequential Construction

Solving (P) directly for hundreds of apprentices is not currently possible in practically acceptable time. It is, however, possible to incrementally construct a complete solution by scheduling a small set of apprentices $\hat{\mathcal{N}} \subseteq \mathcal{N}$ at a time by solving an IP model. A similar IP-based constructive approach was considered in Chapter 3.

In order to incrementally construct a schedule, we first introduce an augmented version of (P):

$$\text{(AP) min:} \quad \sum_{i \in \hat{\mathcal{N}}} \sum_{j=1}^{M} \sum_{t=1}^{T} c_{i,j}(X_{i,j,t} - Y_{i,j,t}) + \mu \sum_{j=1}^{M} \sum_{t=1}^{T} (U_{j,t} + V_{j,t}) \tag{6.21}$$

$$\text{s.t.} \quad (6.2) - (6.8)$$

$$\sum_{i \in \hat{\mathcal{N}}} (X_{i,j,t} - Y_{i,j,t}) + U_{j,t} \geq a_j Z_{j,t} \qquad j = 1, \ldots, M; t = 1, \ldots, T \tag{6.22}$$

$$\sum_{i \in \hat{\mathcal{N}}} (X_{i,j,t} - Y_{i,j,t}) - V_{j,t} \leq b_j Z_{j,t} \qquad j = 1, \ldots, M; t = 1, \ldots, T \tag{6.23}$$

$$(6.11) - (6.14) \quad (\text{with } i \in \hat{\mathcal{N}} \text{ instead of } i = 1, \ldots, N)$$

$$U_{j,t} \in \mathbb{Z}^+ \qquad j = 1, \ldots, M; t = 1, \ldots, T \tag{6.24}$$

$$V_{j,t} \in \mathbb{Z}^+ \qquad j = 1, \ldots, M; t = 1, \ldots, T \tag{6.25}$$

where the augmenting variables $U_{j,t}$ and $V_{j,t}$ represent the shortfall and excess apprentices in placement $j \in \mathcal{M}$ at time $t \in \mathcal{T}$, respectively. The objective function (6.21) penalises these values, weighted by a very large coefficient μ.

The (AP) model permits solutions that violate the constraints (6.9) and (6.10) from (P), albeit with considerable penalty. Allowing these violations is necessary when constructing a solution a small number of apprentices at a time. For example, if $\min_{j \in \mathcal{M}} a_j > |\hat{\mathcal{N}}|$, then no feasible solution can exist. Even when $\min_{j \in \mathcal{M}} b_j < |\hat{\mathcal{N}}|$, a feasible solution may not exist without the augmenting variables, even if a feasible solution exists for (P).

Apprentices are dispatched (scheduled) one or more at a time, in a particular order. At each iteration, the considered apprentices $\hat{\mathcal{N}}$ are scheduled by solving (AP), and the solution is appended to a partial solution that becomes increasingly complete at each iteration with respect to the underlying problem. Each time (AP) is solved for a given set of apprentices, it is subject to the partial solution so far, i.e. the apprentices that have been scheduled in all previous iterations. A complete solution is produced once all apprentices have been scheduled. The order in which the apprentices are scheduled is likely to have a significant effect on the final constructed solution. We propose three basic ordering rules, and their relative performance is presented in Section 6.6:

1. Indexed order: Schedule apprentices in the order in which they are defined.

2. Random order: Schedule apprentices at random.

3. Average placement weighted penalty: Schedule apprentices in decreasing order of $\sum_{g \in G_j}(\sum_{j \in g}(l_j c_{i,j})/|g|)$.

While the sequential construction approach can produce a solution with substantially less computational effort than by directly solving (P), the solution quality is not guaranteed to be good. The produced solution is, however, a good starting point for some other solution approaches, such as local search metaheuristics.

6.4.2 Simulated Annealing

The solution obtained from the sequential constructive approach may be improved by a local search metaheuristic such as Simulated Annealing (SA). A straight forward implementation of SA is proposed.

A solution is represented by the set of all apprentices, each of whom is characterised by: ***(i)*** a list of placements they will attend, ***(ii)*** a list of durations corresponding to their attended placements, and ***(iii)*** a list of starting offsets—the starting offset of a given placement for an apprentice is the unallocated time between the start of the placement and end the of the previous placement.

The neighbourhood function produces at each iteration a candidate neighbour solution σ', based on the current solution σ, by applying one of the following operations to a random apprentice:

- swapping the order of two placements at random,
- substituting one random placement for another from the same placement group,
- incrementing or decrementing the amount of time spent at a random placement, or
- incrementing or decrementing the starting offset for a random placement.

Selection of the apprentice and neighbourhood operation is at random with uniform probability. The first and second types of operations listed above will always produce a feasible solution to the augmented (AP) model. The third and fourth types of operations should be applied in such a way that the minimum placement duration is not violated, and that starting offsets are always non-negative.

6.5 Fix-and-Optimise Large Neighbourhood Search Approach

Since the problem considered in this chapter has not yet been considered in the literature, we propose an alternate solution approach in order to compare with the procedure described in Section 6.4. We propose a fix-and-optimise [67, 130] Large Neighbourhood Search (LNS) [141, 129] algorithm.

In the iterative fix-and-optimise LNS procedure, each iteration solves a subproblem consisting of a subset of the original variables, with the remaining variables being “fixed” as constants. The neighbourhood to search at each iteration is the entire set of feasible solutions to the considered subproblem. Different subsets of variables to optimise over are chosen for each iteration in attempt to gradually improve the solution. Since the previous solution must be feasible in the subsequent iteration, each iteration therefore results in a solution no worse than the previous.

To start with, the fix-and-optimise LNS heuristic must begin with an initial feasible solution. A feasible solution to (AP) simply satisfies apprentice placement requirements, with the restrictions on minimum and maximum placement sizes relaxed. We can therefore easily construct an initial feasible solution by assigning each apprentice to any one placement for each of their required placement groups for the minimum required duration, ignoring the number of apprentices in each placement at any given time. Alternatively, an initial feasible solution can be obtained using the constructive heuristic described in Section 6.4.1.

At each iteration, decisions need to be made about which variables will fixed (replaced by constants). We fix the variables associated with the following two solution features:

1. The complete timetable for a given apprentice,

2. The complete timetable for a given non-empty placement.

In the former case, one or more apprentices are selected. All X and Y variables associated with those apprentices have their values replaced by constants corresponding to the solution at the previous iteration. In the latter case, one or more placements are selected, and all X and Y variables corresponding to those placements are replaced by constants. It is possible to fix both a subset of placements and a subset of apprentices in a given iteration.

Naturally, the more features that are fixed, the faster the subproblem can be solved, however it is less likely that larger improvements can be made. Conversely, the fewer features that are fixed, the greater the opportunity for improvement on average, however significantly more computational effort is likely to be required per iteration. We propose to initially fix a large number of apprentices and placements, perhaps as high as 90% for larger problem sizes, in order to keep computation times short per iteration. When the algorithm fails to improve the solution after a certain number of iterations, the number of apprentices and placements to fix should be decreased at a geometric rate.

We propose to deterministically select apprentices and placements to fix. For each apprentice i, we determine what their contribution is to the objective value defined by $\sum_{j=1}^{N}\sum_{t=1}^{T} c_{i,j}(X_{i,j,t} - Y_{i,j,t})$. Similarly, for each placement j, we determine what their contributions are to the objective value defined by $\sum_{i=1}^{M}\sum_{t=1}^{T} c_{i,j}(X_{i,j,t} - Y_{i,j,t}) + \mu\sum_{t=1}^{T}(U_{j,t} + V_{j,t})$. The list of apprentices and placements are each sorted by their contributions to the objective value. In the event that two apprentices or two placements have the same contribution, they are ordered according to their index. If α apprentices and β placements are to be fixed, then $\frac{\alpha}{2}$ apprentices are selected from the start of the list, and $\frac{\alpha}{2}$ apprentices are selected from the end of the list, and likewise $\frac{\beta}{2}$ placements are selected from the start of the list, and $\frac{\beta}{2}$ placements are selected from the end of the list.

In order to prevent the algorithm getting stuck in a cycle, a list of previously fixed sets of variables and their corresponding values must be kept, and the algorithm is not permitted to revisit these neighbourhoods. If the selection procedure described above would call for a previously visited neighbourhood to be revisited, then the set of apprentices and/or placements to fix should be shifted inwards. For example, instead of fixing apprentices $\{1, \ldots, \frac{\alpha}{2}\}$ and $\{M - \frac{\alpha}{2}, \ldots, M\}$, apprentices $\{2, \ldots, \frac{\alpha}{2} + 1\}$ and $\{M - \frac{\alpha}{2} - 1, \ldots, M - 1\}$ should be fixed, and likewise for placements.

When the number of features to fix reaches a minimum threshold, when the algorithm reaches a time limit, or when the algorithm cannot find an unvisited neighbourhood, the procedure is terminated and the best solution is returned.

6.6 Computational Results

In order to test the proposed solution approaches, a set of 36 test cases were randomly generated with similar characteristics to real-world cases found at Australia's largest electricity distributor (by number of connections and number of employees). All test cases and corresponding solution files can be downloaded from `https://goo.gl/c8yt8H`. The testing system was an Intel i7-4790K quad core CPU with 16GB RAM, running Microsoft Windows 10 64-bit. Code was written in C#, and we used IBM ILOG CPLEX 12.5.0.0 64-bit using the ILOG Concert API to solve the mathematical programming models, including a straight-forward "direct" approach of the (AP) models for comparison. CPLEX was given a two-hour time limit for the direct approach.

We set $T = 1.5 \times \max_{i \in \mathcal{N}} \left\{ \sum_{g \in G_i} \max_{j \in g} \{l_j\} \right\}$ in order to approximate the required length of the timeline. We constructed timetables with the sequential heuristic using all three list permutations mentioned in Section 6.4.1. In each case, we scheduled apprentices one at a time. We then chose the timetable with the best objective value and used that as a starting point for the SA metaheuristic mentioned in Section 6.4.2.

Neighbour solutions in SA were chosen randomly with uniform probability. We set the initial temperature to 10^6, the termination temperature to 10^{-3}, and used a geometric cooling rate of 0.99, which was applied at each iteration. These values were determined experimentally, using trial-and-error on some small, medium, and large test cases. It is a matter of future research to determine what parameter values perform best in general.

For the fix-and-optimise LNS heuristic, we started with an initial feasible solution created by assigning apprentices to one placement from each of their required placement groups, ignoring placement

Case	$\vert\mathcal{N}\vert$	G	$\vert\mathcal{M}\vert$	ρ	Case	$\vert\mathcal{N}\vert$	G	$\vert\mathcal{M}\vert$	ρ
01	100	4	16	2	19	200	4	16	2
02	100	4	16	4	20	200	4	16	4
03	100	4	32	2	21	200	4	32	2
04	100	4	32	4	22	200	4	32	4
05	100	4	64	2	23	200	4	64	2
06	100	4	64	4	24	200	4	64	4
07	100	8	16	2	25	200	8	16	2
08	100	8	16	4	26	200	8	16	4
09	100	8	32	2	27	200	8	32	2
10	100	8	32	4	28	200	8	32	4
11	100	8	64	2	29	200	8	64	2
12	100	8	64	4	30	200	8	64	4
13	100	16	16	2	31	200	16	16	2
14	100	16	16	4	32	200	16	16	4
15	100	16	32	2	33	200	16	32	2
16	100	16	32	4	34	200	16	32	4
17	100	16	64	2	35	200	16	64	2
18	100	16	64	4	36	200	16	64	4

Table 6.1: The number of apprentices, placement groups, and placements for the 36 test cases

bounds as described in Section 6.5. At each iteration, we initially allowed 75% of both apprentices and placements to be fixed, and these values were squared each time no improvement could be made. We gave each iteration a time limit of half an hour, and a time limit of two hours for the entire procedure. The procedure would also terminate if no apprentices or placements would be fixed.

Table 6.1 outlines the 36 test cases that were tested. The tables outline the test case identifier (Case), the number of apprentices ($|\mathcal{N}|$), the total number of placement groups (G), the number of placements ($|\mathcal{M}|$), and the number of groups per apprentice (ρ).

Tables 6.2 and 6.3 show the time taken, in minutes, for each procedure tested. The columns are as follows: The test case (Case); the time taken for sequential construction using the indexed order (Idx), random order (Rnd), and average placement weighted penalty order (Pty); the time taken for the SA algorithm to converge (SA), the time for the fix-and-optimise LNS algorithm to converge (LNS), and the time taken for the IP to terminate (IP).

The sequential construction procedures took about 3.75 minutes on average for the test cases with

Case	Idx	Rnd	Pty	SA	LNS	IP
01	**0.3**	0.3	0.3	1.5	23.4	1.2
02	**0.7**	0.8	0.8	4.6	33.3	19.3
03	**0.6**	0.6	0.6	1.4	23.2	10.8
04	**3.2**	3.9	3.7	26.6	120.0	120.5
05	**1.6**	1.7	1.6	3.8	120.2	120.0
06	**7.6**	8.1	8.2	8.6	120.1	120.1
07	**0.3**	0.3	0.3	4.1	120.2	120.2
08	3.0	**1.2**	1.3	15.8	120.1	120.0
09	3.5	2.7	**2.6**	2.8	26.4	6.9
10	7.6	8.6	7.6	**3.9**	120.1	120.0
11	5.5	5.5	5.5	**2.5**	120.3	120.0
12	**14.1**	14.6	14.5	83.7	120.3	120.1
13	1.0	0.8	0.8	1.6	23.0	**0.1**
14	2.5	2.5	2.7	7.4	24.1	**2.4**
15	1.7	2.2	1.6	4.1	23.9	**0.9**
16	4.6	4.8	4.9	**2.8**	23.0	4.5
17	3.2	3.1	3.1	**1.8**	39.7	30.1
18	6.7	6.6	6.5	**5.7**	120.2	120.0

Table 6.2: The time taken, in minutes, for each tested procedure (1 of 2).

Case	Idx	Rnd	Pty	SA	LNS	IP
19	**1.9**	**1.9**	**1.9**	2.8	23.6	4.2
20	9.7	8.9	9.4	**8.2**	120.0	120.0
21	4.3	4.3	4.5	**3.3**	34.4	21.1
22	16.6	15.5	**14.9**	32.0	120.2	120.1
23	6.2	6.1	6.1	**4.3**	120.3	120.0
24	**27.4**	29.1	30.7	37.2	120.3	120.0
25	1.6	1.5	**1.4**	7.0	22.4	2.5
26	13.2	6.6	6.1	**3.3**	27.6	23.0
27	3.4	3.2	3.6	3.3	24.3	**3.1**
28	11.7	12.2	**10.5**	58.0	120.2	120.0
29	5.4	5.5	5.6	**5.1**	83.7	71.4
30	21.8	**21.5**	22.8	26.9	120.3	120.0
31	1.2	1.3	1.1	2.9	24.7	**0.2**
32	5.7	**5.2**	5.7	17.0	30.9	7.9
33	2.2	2.2	**2.1**	8.9	22.8	7.5
34	7.4	7.5	7.5	**5.6**	120.0	120.0
35	2.1	**2.1**	2.4	30.5	120.2	120.6
36	**18.4**	19.5	19.8	88.0	120.3	120.0

Table 6.3: The time taken, in minutes, for each tested procedure (2 of 2).

100 apprentices, and about 8.71 minutes on average for the test cases with 200 apprentices. Simulated Annealing took about 10.15 minutes on average to converge for the test cases with 100 apprentices, 19.12 minutes to converge for the test cases with 200 apprentices. The fix-and-optimise LNS algorithm took about 79.1 minutes on average to converge for the test cases with 100 apprentices, and 76.5 minutes on average to converge for the test cases with 200 apprentices. The LNS algorithm required a minimum of about 22.4 minutes to converge, even on test cases where SA and IP were around or under one minute. This is due to the fix-and-optimise LNS framework not detecting that the problem can be efficiently solved to optimality without fixing any variables, and spending a lot of time performing unnecessary iterations.

The IP approach terminated at the two hour time limit for half of the test cases, although these were spread evenly between those cases with 100 and those with 200 apprentices. The average time overall for the IP approach, including for those that terminated at the time limit, was 66 minutes, and there was no significant difference in average time between the test cases with 100 and with 200 apprentices. For the test cases where CPLEX reported an optimal solution before reaching the time limit, the average solve time was 12 minutes, and again there was no significant difference in the cases where there were 100 or 200 apprentices.

Table 6.4 shows the total number of lower- and upper-bound violations of placement size, given by $\sum_{j=1}^{M}\sum_{t=1}^{T}(U_{j,t}+V_{j,t})$, for the solutions produced by each procedure. The columns are as follows: The test case (Case); the number of bound violations produced by sequential construction for indexed order (Idx), random order (Rnd), and average placement weighted penalty order (Pty); the number of bound violations for the solution produced by the SA algorithm (SA), the number of bound violation for the solution produced by the fix-and-optimise LNS algorithm (LNS), and the number of bound violations for the solution produced by the IP approach (IP). CPLEX did not report any integer solutions for test cases `08`, `24`, and `30` prior to terminating at the time limit.

Each of the sequential construction procedures produced violation-free solutions for 10 of the 36 test cases. For 6 of those 10, all three list permutations were able to produce violation-free solutions, whereas for the other 4, some list permutations produced violation-free solutions while other permutations did not. The average number of bound violations were 32, 30, and 31, respectively, for the three tested permutations.

Both SA and IP produced 23 violation-free solutions for the 36 test cases. For 16 of those 23 cases, SA and IP both produced violation-free solutions. For test case `20`, SA produced a solution with no bound violations, while the IP approach produced 1076 violations after terminating at the two hour time limit. For test case `22`, SA again produced a solution with no bound violations, while the IP approach produced a solution with 845 bound violations.

Case	Idx	Rnd	Pty	SA	LNS	IP	Case	Idx	Rnd	Pty	SA	LNS	IP
01	29	29	31	**0**	0	**0**	19	20	19	19	**0**	0	0
02	4	20	18	**0**	0	**0**	20	61	10	10	**0**	5	1076
03	**0**	0	**0**	0	**0**	0	21	**0**	0	1	**0**	0	0
04	12	23	28	**0**	0	465	22	30	13	18	**0**	14	845
05	**0**	1	**0**	0	**0**	2	23	**0**	0	**0**	0	**0**	0
06	**0**	0	**0**	0	**0**	83	24	4	4	4	1	**0**	-
07	122	118	114	18	**0**	5	25	118	120	120	18	15	0
08	78	40	72	5	**0**	-	26	2	4	2	**0**	2	0
09	9	7	4	**0**	0	**0**	27	12	27	43	**0**	0	0
10	1	**0**	0	**0**	0	46	28	219	214	232	1	**0**	474
11	1	2	**0**	0	**0**	0	29	**0**	0	**0**	0	**0**	0
12	8	3	2	2	**0**	47	30	1	1	**0**	0	**0**	-
13	11	3	9	8	**0**	0	31	111	100	92	**0**	0	0
14	13	20	10	3	**0**	0	32	214	194	210	101	**0**	0
15	4	14	1	3	**0**	0	33	58	55	46	1	**0**	0
16	**0**	0	4	**0**	0	**0**	34	**0**	0	2	**0**	0	0
17	**0**	0	**0**	0	**0**	0	35	14	18	9	5	**0**	0
18	**0**	0	**0**	0	**0**	0	36	**1**	29	8	2	6	365

Table 6.4: $\sum_{j=1}^{M} \sum_{t=1}^{T} (U_{j,t} + V_{j,t})$ for solutions produced by each tested procedure.

Case	Idx	Rnd	Pty	SA	LNS	IP
01	-	-	-	23466.74	**12218.00**	**12218.00**
02	-	-	-	31456.12	19588.86	**19539.15**
03	11380.88	11380.88	11380.88	**8950.06**	11380.88	**8950.06**
04	-	-	-	58500.79	**19681.93**	-
05	30538.88	-	29993.70	26038.54	**5738.52**	-
06	20413.69	20413.69	20413.69	20413.69	**10901.95**	-
07	-	-	-	-	**28249.83**	-
08	-	-	-	-	**50089.23**	-
09	-	-	-	34522.71	14464.76	**14452.74**
10	-	29381.33	29379.06	29379.06	**28132.44**	-
11	-	-	20402.50	20402.50	**9906.64**	9927.31
12	-	-	-	-	**21096.81**	-
13	-	-	-	-	**32044.13**	32044.13
14	-	-	-	-	**44299.37**	44299.37
15	-	-	-	-	**19000.53**	19000.53
16	48001.84	48003.71	-	48001.45	**47292.00**	**47292.00**
17	25266.19	25267.24	25267.24	23561.66	**15840.75**	**15840.75**
18	23908.51	23910.29	23911.83	23908.51	**22838.19**	22841.66

Table 6.5: $\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{t=1}^{T} c_{i,j}(X_{i,j,t} - Y_{i,j,t})$ for solutions with no bound violations produced by each tested procedure (1 of 2).

The fix-and-optimise heuristic was able to produce violation-free solutions for all but five of the test cases. Interestingly, the approach could not produce violation-free solutions for test case 26, whereas both the SA and IP approach could. Overall, the fix-and-optimise LNS approach was much more successful than the SA and IP approaches on average, with respect to producing solutions free of bound violations.

The quality of solutions, with respect to the number of bound violations, produced by SA appear to be of roughly similar quality to the solutions produced by the IP approach. The SA approach, however, did not have a time limit, and took on average about 14.6 minutes to converge, whereas CPLEX was given a two hour limit for the IP approach. It is reasonable to assume that if the SA procedure was tuned to run for a longer duration, it would have produced better solutions.

Tables 6.5 and 6.6 show the solution quality, given by $\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{t=1}^{T} c_{i,j}(X_{i,j,t} - Y_{i,j,t})$, for the solutions free of bound violations produced by each procedure. The columns are as follows: The

Case	Idx	Rnd	Pty	SA	LNS	IP
19	-	-	-	33116.71	**22657.04**	**22657.04**
20	-	-	-	**77746.94**	-	-
21	28490.52	28484.20	-	27858.08	**15261.23**	**15261.23**
22	-	-	-	**81712.63**	-	-
23	12898.13	12899.39	12898.53	12898.13	**11401.68**	11411.07
24	-	-	-	-	**25942.54**	-
25	-	-	-	-	-	**37379.71**
26	-	-	-	89275.82	-	**83455.80**
27	-	-	-	36460.34	**27029.42**	**27029.42**
28	-	-	-	-	**67014.59**	-
29	41506.05	41644.64	40808.95	35225.88	16741.77	**16731.11**
30	-	-	45126.83	45126.83	**34255.88**	-
31	-	-	-	70340.18	**69970.71**	**69970.71**
32	-	-	-	-	**119561.60**	**119561.60**
33	-	-	-	-	**38290.64**	**38290.64**
34	102606.22	102603.19	-	102603.19	**100582.43**	**100615.49**
35	-	-	-	-	**25535.05**	25512.38
36	-	-	-	-	-	-

Table 6.6: $\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{t=1}^{T} c_{i,j}(X_{i,j,t} - Y_{i,j,t})$ for solutions with no bound violations produced by each tested procedure (2 of 2).

test case (Case); the solution quality produced by sequential construction for indexed order (Idx), random order (Rnd), and average placement weighted penalty order (Pty); the solution quality for the solution produced by the SA algorithm (SA), the solution quality for the solution produced by the fix-and-optimise LNS algorithm (LNS), and solution quality for the solution produced by the IP approach (IP). We do not report the solution quality where the number of bound violations is nonzero.

For the 16 cases where both the SA and the IP approach were able to produce solutions free of bound violations, the solutions produced by SA were, on average, 25.35% worse than the solutions produced by the IP approach. CPLEX reported optimal solutions to 12 of these 16 cases. The total solve time for these 16 cases, on average, was 3.35 minutes for SA, and 42.23 minutes for the IP approach.

The fix-and-optimise LNS approach produced the best solution for 26 of the 36 test cases, and the uniquely best solution, i.e. where the solution was strictly better than the other tested approaches, for 9 of the test cases.

Overall, each approach has its advantages and disadvantages. All three approaches require a general purpose solver, however the two stage construction and improvement approach from Section 6.4 can be modified to use an initial solution by a different means, allowing for a solver-free implementation. The IP approach is the simplest to implement and performs similar on average to the two-stage (SA) approach, however the use of a solver is unavoidable. The SA and IP approaches were very similar in performance to one another, with the SA approach performing slightly better with respect to bound violations, and IP performing slightly better when looking at the quality of violation-free solutions.

For the tested cases with the tested parameters, the fix-and-optimise LNS approach appears to be the overall winner. It produced the most good-quality, violation-free solutions. The drawbacks of the fix-and-optimise heuristics are that the implementation is the most complex by far, using a solver is unavoidable, and the computational time is quite high for the easier test cases.

6.7 Conclusions

This chapter considers the problem of assigning apprentices to practice placements. It is shown that even the problem requiring the answer to the question: "Does there exist a feasible solution?", is NP-complete. The chapter presents two solution approaches. The first is n optimisation procedure comprised of a sequential application of Integer Linear Programming, and a Simulated Annealing metaheuristic. The second is a fix-and-optimise Large Neighbourhood Search approach.

The proposed heuristics were tested by means of computational experiments on a number of randomly generated test cases, with similar characteristics to relevant data at Australia's largest electricity distributor (by number of connections and number of employees). The straight forward approach of solving the IP model subject to a two-hour time limit produced good solutions in most cases, however this approach is not suitable when the size of the problem grows large. The approach of sequentially constructing a solution one apprentice at a time by solving an augmented version of the IP produced solutions quickly, but with many violations on the lower and upper bounds on the permissible number of apprentices in each placement. The approach of improving a sequentially constructed schedule using Simulated Annealing was shown to produce solutions of similar quality to the approach of solving the IP, but required substantially less time. The approach of iteratively fixing certain variables and solving the remainder of the problem performed the best overall, both with respect to bound violations and solution quality.

Future work can further investigate different apprentice ordering rules, as well as mapping the performance versus quality trade-off of considering a greater or fewer number of apprentices at each iteration of the sequential constructive stage. For the SA implementation, applying the operations in the neighbourhood function in a non-uniform manner, in order to better direct the optimisation away from poor solutions should be investigated. For the fix-and-optimise LNS implementation, the effect of selecting features to fix using different selection rules should be investigated.

Chapter 7

Conclusions

This thesis has examined several related problems arising in practice at an Australian electricity distributor from both analytical and practical perspectives. For each of the problems discussed, current knowledge in the respective areas has been advanced, and several new questions for further research have been prompted.

7.1 Summary

This section briefly summaries the problems discussed in Chapters 3 to 6, including the main results and conclusions.

7.1.1 Demand-Based Course Timetabling

Chapter 3 presented the NP-hard problem of producing a demand-based class timetable and trainer roster. This problem shares many similarities with high school and university class timetabling, however the presence of a number of distinguishing features, summarised in Section 2.3, make the existing approaches from the literature not directly applicable.

The problem was modelled by means of Integer Programming (IP), where IP models ware constructed for the class timetabling subproblem, and for the trainer rostering subproblem. This decomposition is justified by the computational effort required. Even with the decomposition, the problem remains unsolvable in practically acceptable time even for very small test cases.

A three-stage heuristic approach was presented. First, an initial timetable is constructed class-by-class by means of solving a restricted version of the class timetabling IP model. Next, the timetable is improved by means of an iterative fix-and-optimise Large Neighbourhood Search (LNS) procedure, in which small subsets of variables are carefully selected to be included when solving the class timetabling IP model, and the rest replaced by constants. Finally, the trainer rostering subproblem, subject to the solution from the class timetabling subproblem, is solved by direct application of IP.

The results showed that the proposed approach could produce good quality solutions to the considered problem in practically acceptable time, whereas direct application of mathematical programming could not be shown to make the same claim. The approach that was presented can be generalised and applied to many different problems.

7.1.2 Single-Machine Batch Scheduling with Incompatible Job Families and an Ordered Bi-Criteria Objective

Chapter 4 discussed a problem with largely the same motivation as the problem studied in Chapter 3. This problem, however, was presented from an entirely different perspective. Chapter 4 is concerned with a class timetabling problem for a single bottleneck classroom. In this study, individual students are to be assigned to classes, whereas individual student assignments were not considered in Chapter 3.

The problem was discussed in the context of batch scheduling, which is an area of research within the field of machine scheduling. Within the batch scheduling framework, each student's requirement for a particular course is represented by a job, and the student's expiry date for that qualification corresponds to the job's due date. The bottleneck classroom is represented by a batching machine, and the size of the classroom corresponds to the number of jobs the machine can process simultaneously at one time. A class is composed of students who are attending the same course as one another, therefore each job has a family that corresponds to the course, and the problem is a batch scheduling problem with incompatible job families. The problem has an ordered bicriteria objective; the goal is to find a solution that minimises the secondary objective over the set of solution that minimises the primary objective.

For the objective of minimising the number of tardy jobs, the NP-hardness in the strong sense was proven by reduction from the single machine total weighted tardiness problem. The objective of minimising an arbitrary, regular maximum cost function of job completion time, an $O(n^2)$ algorithm was presented.

An extensive computational study was carried out on the bicriteria case, where the primary objective is the maximum weighted tardiness and the secondary objective is the total weighted tardiness, which is justified by the problem's real world application. Both Simulated Annealing (SA) and Genetic Algorithm (GA) metaheuristics are proposed. Both algorithms outperformed direct application of IP, with GA making slower but more consistent progress than SA, especially for the larger test cases.

7.1.3 Partitioning of Students into Classes

Chapter 5 discussed a partitioning problem, which has to do with deciding which students should attend which class. Each student is of exactly one type, and must be assigned to exactly one class. Classes have minimum and maximum capacity of students and of student types; classes may remain empty at no cost. Students have due dates for their training, and classes are distributed along the timeline. Each student must be assigned to exactly one class, and the goal is to minimise the total penalty associated with assigning particular students to particular classes, and the total penalty associated with combining different student types within the same class.

The problem was shown to share many similarities with previously studied problems, such as the Graph Partitioning Problem and the related Edge Partitioning Problem, the Quadratic Multiple Knapsack Problem, and the Quadratic Assignment Problem. The problem was shown to be NP-hard in the strong sense by reduction from the clique problem.

Four solution approaches were presented. The first approach is based on the Lagrangian Relaxation, whereby trends are identified in the intermediate solutions of the subgradient algorithm and the trends inform us about reasonable restrictions to impose on the problem to significantly reduce the solution space. The second approach is based on Column Generation, whereby partial solutions corresponding to student assignments for each class are generated. Three strategies were proposed for finding integer feasible solutions. The third approach incorporates the Column Generation approach within a fix-and-optimise Large Neighbourhood Search (LNS) framework. The fourth approach was a Genetic Algorithm, which incorporates an IP subroutine to avoid the problem of infeasibility.

Extensive computational experimentation showed that the fix-and-optimise Large Neighbourhood Search approach performed the best overall, followed by the Column Generation-based approaches. The Genetic Algorithm was the least successful of the four proposed approaches, however all four approaches significantly outperformed direct application of mathematical programming.

7.1.4 Timetabling of Practice Placements

Chapter 6 discussed a problem concerned with the timetabling of practice placements. This problem is often encountered by students who must complete a variety of supervised practical training placements—also known as rotations—in addition to their theoretical studies. The problem is perhaps most notable in the area of clinical education, where for example nursing students must complete a set of practice placements (also known as clinical placements) following their theoretical studies in order to graduate. The study of this problem was motivated by the problem of timetabling electrical apprentices at the electricity distributor into practice placements.

In the studied problem, there are a set of placement groups, each comprised of a set of placements.

Each placement in a particular group exposes the apprentice to the same core set of skills, however they are otherwise distinct. Each placement belongs to exactly one group, and has a minimum and maximum capacity of apprentices, however the placement can also remain empty. Each apprentice, depending on the type of apprenticeship they are enrolled in, has a set of placement groups they must attend. They must attend exactly one placement from each of their required groups, in any order, and for a minimum duration. There is a penalty for assigning particular apprentices to particular placements, and this penalty is accumulated per unit of time they spend at the placement. The objective is to produce a timetable in which all students are assigned to exactly one placement for each of their required placement groups for at least a minimum duration, where at any given time placements are either empty or have between a minimum and maximum number of apprentices, such that the total weighted assignment penalty is minimised. The problem was shown to be NP-hard by reduction from the partition problem.

The problem is formulated using IP, and two solution approaches were presented. The first approach constructs an initial timetable by means of successively solving a restricted and augmented version of the IP model in a similar manner to what was proposed in Section 3.2.1, followed by an improvement stage based on Simulated Annealing. The second approach is a fix-and-optimise Large Neighbourhood Search procedure to produce an improved timetable.

Extensive computational experimentation was carried out on a number of randomly generated test cases with similar characteristics to real-world cases at the electricity distributor. The results showed that direct application of IP was not practical as the problem size grew larger. The approach of using SA to improve a sequentially constructed timetable was shown to produce results of similar quality to IP, but required substantially less time. The fix-and-optimise LNS approach performed the best overall.

7.2 Future Work

This section outlines a number of recommended directions for future research for each of the problems discussed in this thesis.

7.2.1 Demand-Based Course Timetabling

The demand-based class timetabling and trainer rostering problem in Chapter 3 was modelled by means of two IP models: one for the class timetabling subproblem and one for the trainer rostering subproblem. This was done not only to significantly simplify the modelling process, but also to increase the tractability of the considered problem. Alternatively, one could choose to formulate the problem as a single, integrated IP model that covers both subproblems. A complete solution produced using the optimal solutions from the class timetabling and trainer rostering subproblems will not necessarily have an objective value as low as the optimal solution from the integrated approach. Moreover, since the polytope for the trainer rostering subproblem depends on the solution from the class timetabling subproblem, one cannot even say that the best strategy is to produce the best possible class timetable. In fact, it was shown in Section 3.6 that in some cases, the combined objective values of the optimal timetable and the optimal roster given the optimal timetable was poorer than the combined objective values of a suboptimal timetable and the optimal roster given the suboptimal timetable. Future work should investigate a single, integrated model for the class timetabling and trainer rostering problem. As a starting point, such a model could introduce an additional subscript t to the X variables, corresponding to the assignment of a particular trainer t assigned to teach module m in room r during period p. In the worst case, this change alone represents an $O(2^t)$ increase in the number of variables in the integrated model, compared with the class timetabling model; there would be additional constraints also. Given that the decomposed approach from Chapter 3 presented a significant computational challenge, an integrated approach would very likely require far greater computational effort to solve. This increase in difficulty together with the increase in the size of the search space means that the fix-and-optimise LNS approach may not be as attractive. This is because more iterations would likely be required to explore the search space thoroughly, and each iteration would likely take much longer. For this reason, more efficient modelling approaches and alternative solution approaches should be investigated.

The approach of Column Generation for the integrated class timetabling and trainer roster problem should be investigated. The integrated problem and be expressed by means of a set covering formulation. A *pattern* would describe the assignment of particular classes to rooms, and trainers to modules. Each pattern would be feasible with respect to the timetabling and rostering constraints, so the responsibility of the set covering model would be to decide which patterns should be used, and when and where the patterns should be run. Additional constraints in the set covering model would prevent a particular trainer from being assigned to two different places at the same time by two patterns. Naturally, the number of possible patterns is far too large to enumerate, even for small problems. A Column Generation approach would make use of the dual variables from the linear relaxation to the reduced master problem (set covering formulation), and generate new patterns using another IP model. A similar approach was explored in Section 5.6.

Another approach that may be promising would be to make use of a quadratically constrained model, which would require significantly fewer variables than the linearly constrained models discussed in Chapter 3. For example, currently $X_{m,r,p}$ is 1 if module m runs in room r starting at period p or 0 otherwise, requiring up to $|M| \times |R| \times |P|$ binary variables. If instead we replace $X_{m,r,p}$ by $X_{m,r}X_{m,p}$, where $X_{m,r}$ is 1 if module m runs in room r or 0 otherwise, and $X_{m,p}$ is 1 if module m starts at time p or 0 otherwise, then only up to $|M| \times (|R| + |P|)$ binary variables are needed. Moreover, in an integrated model, $X_{m,r,t,p}$ which would be 1 if module m running in room r taught by trainer t starting at period p, or 0 otherwise, can be replaced by $X_{m,r}X_{m,t}X_{m,p}$, where $X_{m,t}$ is 1 if trainer t teaches module m or 0 otherwise. This would require up to $|M| \times (|R| \times |T| \times |P|)$ variables instead of up to $|M| \times |R| \times |T| \times |P|$.

7.2.2 Single-Machine Batch Scheduling with Incompatible Job Families and an Ordered Bi-Criteria Objective

Further study of the approaches proposed for the problem in Chapter 4 should thoroughly explore, by computational experimentation, the parameter space in order to determine the parameter values that perform best in general, as well as for specific problem instances.

Further study should be also be focussed on producing approximation algorithms for the NP-hard cases together with performance guarantees. Typically, such performance guarantees provide an upper bound (for a minimisation objective) on either the ratio $(\frac{f(\sigma)}{f(\sigma^*)})$ or the difference $(f(\sigma) - f(\sigma^*))$ between the heuristic solution quality σ and optimal solution quality σ^* given some objective function f.

Algorithms based on Dynamic Programming (DP) [115] can be developed to solve the problem considered, however the NP-hard cases will require an exponential number of operations on the problem size. Further research can be directed towards finding a Polynomial Time Approximation Scheme (PTAS) for the problem. A PTAS is a polynomial time algorithm that can find a solution with an objective value no greater than $1 + \frac{1}{\epsilon}$ times the optimal for any fixed $\epsilon > 1$.

To address some of the uncertainties involved in problem, Stochastic Dynamic Programming [134] can be an area for further research. Some sources of uncertainty include students not being able to attend classes, or classes needing to be cancelled, for example if the trainer is unavailable and no substitute can be found.

7.2.3 Partitioning of Students into to Classes

The problem studied in Chapter 5 was formulated by means of Quadratic Programming (more specifically, a Linearly Constrained Quadratic Program), and a linearisation was given, resulting in an Integer Programming formulation. The addition of new variables was necessitated by this linearisation, increasing the size of the model. The computational experiments showed, however, that the linearised model performed better than the quadratic model despite the additional variables. It is unclear whether this is the case in general, or simply a result of the properties of the test cases that were studied. Further study is needed to identify the conditions under which IP outperforms QP and vice-versa.

For the Column Generation approach, it is not strictly necessary to find the incoming column (pattern) with the most negative reduced cost as is described in Section 5.6. In fact, *any* column with a negative reduced cost can be introduced. Further research should investigate how to effectively balance the higher computational cost for the greater reward of introducing columns that are more likely to be beneficial, versus the lower computational cost for the lesser reward of introducing columns with less negative reduced costs. Related to this, approximation methods for the generation of columns should be investigated, for example, Sol and Savelsbergh [144] discuss a fast heuristic approach for generating new columns.

The reduced master heuristic, whereby the reduced master problem is solved, subject to the available columns, with integer constraints became intractable as the number of available columns grew very large. It is easy to see, however, that at most N columns will be active (selected), since there are only N classes, and at most one column can be selected for each class. While it may not be possible to efficiently determine which N columns will lead to an optimal solution, further research should investigate how some columns can be eliminated from the pool to improve the performance of the reduced master heuristic. Strategies related the selection of columns to eliminate are discussed in Vanderbeck [156].

The study in Chapter 5 investigated Lagrangian Relaxation and Column Generation approaches, among others, but did not mention Row Generation. Further research should investigate whether Row Generation is also a suitable approach for this problem. Row Generation has been successfully combined with Column Generation with good results in Nemhauser and Park [116] and in Clarke and Gong [33], and this direction should be investigated for the problem studied in Chapter 5 also.

7.2.4 Timetabling of Practice Placements

The three ordering rules for the initial construction stage, discussed in Section 6.4.1, exhibited similar performance with respect to solution quality. Further research should investigate other ordering rules, and try to determine which rules are likely to work best for a given problem instance.

Further research should investigate more sophisticated operators for the Simulated Annealing metaheuristic, in particular the neighbourhood operator. One possible direction to focus attention is the application of Kempe Chains [92, 113], which was developed for graph colouring problems but has been successfully applied to neighbourhood operators in Simulated Annealing for timetabling problems [151, 152].

Approaches based on decomposition should also be investigated for the practice placement timetabling problem. In particular, future research can be directed towards Column Generation. Columns representing timetables for individual students should be compared with columns representing timetables for individual placements. Due to the restrictive nature of the constraints, it is likely that finding integer feasible solutions to the master problem will be very challenging. In Chapter 5, one of the proposed approaches for finding an integer feasible solution involved identifying certain features of the partial solutions represented by the columns, and reducing the problem's search space by imposing corresponding restrictions to the underlying problem in accordance with those trends. (See Section 5.6.3). A similar approach should be investigated here. Possible features for each given student that are worth investigating include which placement in particular should be attended from each of their groups, and the sequence in which the placement groups are attended. A possible feature for each given placement that is worth investigating include whether the placement is or is not being used by apprentices at particular times.

Bibliography

[1] D. Abramson, M. K. Amoorthy, and H. Dang. Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research*, 16(1):1, 1999.

[2] E. A. Akkoyunlu. A linear algorithm for computing the optimum university timetable. *The Computer Journal*, 16(4):347–350, 1973.

[3] K. Andreev and H. Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[4] J. E. Aronson. The multiperiod assignment problem: A multicommodity network flow model and specialized branch and bound algorithm. *European Journal of Operational Research*, 23(3):367–381, 1986.

[5] M. Azizoglu and S. Webster. Scheduling a batch processing machine with incompatible job families. *Computers & Industrial Engineering*, 39(3):325–335, 2001.

[6] H. Babaei, J. Karimpour, and A. Hadidi. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 2014.

[7] N. Balakrishnan and R. T. Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20(1):25–42, 1990.

[8] J. F. Bard and L. Wan. Workforce design with movement restrictions between workstation groups. *Manufacturing & Service Operations Management*, 10(1):24–42, 2008.

[9] T. Barnett, M. Cross, E. Jacob, L. Shahwan-Akl, A. Welch, A. Caldwell, and R. Berry. Building capacity for the clinical placement of nursing students. *Collegian*, 15(2):55–61, 2008.

[10] M. S. Bazaraa and H. D. Sherali. Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem. *Naval Research Logistics Quarterly*, 27(1):29–41, 1980.

[11] J. Beliën and E. Demeulemeester. Scheduling trainees at a hospital department using a branch-and-price approach. *European Journal of Operational Research*, 175(1):258–278, 2006.

[12] J. Beliën and E. Demeulemeester. On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. *Annals of Operations Research*, 155(1):143–166, 2007.

[13] C.-E. Bichot. Metaheuristics versus spectral and multilevel methods applied on an air traffic control problem. In *International Federation of Automatic Control*. Citeseer, 2006.

[14] T. Birbas, S. Daskalaki, and E. Housos. Course and teacher scheduling in hellenic high schools. In *4th Balkan Conference on Operational Research, Thessaloniki, Greece*, 1997.

[15] A. Bölte and U. W. Thonemann. Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research*, 92(2):402–416, 1996.

[16] P. Bruker, A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. Potts, T. Tautenhahn, and S. Van De Velde. Scheduling a batching machine. *Journal of Scheduling*, 1:31–54, 1998.

[17] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. pages 117–158, 2016.

[18] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics (SIAM), 2012.

[19] E. 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.

[20] E. K. Burke, G. Kendall, et al. *Search Methodologies*. Springer, 2005.

[21] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová. A branch-and-cut procedure for the udine course timetabling problem. *Annals of Operations Research*, 194(1):71–87, 2012.

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

[23] E. Burke, D. Elliman, and R. Weare. A university timetabling system based on graph colouring and constraint manipulation. *Journal of Research on Computing in Education*, 27(1):1–18, 1994.

[24] A. Caprara, D. Pisinger, and P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2):125–137, 1999.

[25] R. Carlson and G. Nemhauser. Scheduling to minimize interaction cost. *Operations Research*, 14(1):52–58, 1966.

[26] M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, pages 373–383, 1996.

[27] M. W. Carter and C. A. Tovey. When is the classroom assignment problem hard? *Operations Research*, 40(1-Supplement-1):S28–S39, 1992.

[28] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[29] Y. Chen and J.-K. Hao. Iterated responsive threshold search for the quadratic multiple knapsack problem. *Annals of Operations Research*, 226(1):101–131, 2015.

[30] T. E. Cheng, M. Y. Kovalyov, and A. V. Tuzikov. Single machine group scheduling with two ordered criteria. *Journal of the Operational Research Society*, pages 315–320, 1996.

[31] S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993.

[32] R. Cipriano, L. Di Gaspero, and A. Dovier. Hybrid approaches for rostering: a case study in the integration of constraint programming and local search. In *International Workshop on Hybrid Metaheuristics*, pages 110–123. Springer, 2006.

[33] L. W. Clarke and P. Gong. Capacitated network design with column generation. *ISE Published Research Papers Georgia Institute of Technology*, 1996.

[34] T. B. Cooper and J. H. Kingston. *The complexity of timetable construction problems*, pages 281–295. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[35] J.-F. Cordeau, G. Laporte, F. Pasin, and S. Ropke. Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, 13(4):393–409, 2010.

[36] T. H. Cormen. *Introduction to Algorithms*. MIT press, 2009.

[37] O. Czibula, H. Gu, A. Russell, and Y. Zinder. A multi-stage IP-based heuristic for class timetabling and trainer rostering. *Annals of Operations Research*, pages 1–29, 2014.

[38] O. Czibula, H. Gu, and Y. Zinder. Timetabling of workplace training: A combination of mathematical programming and simulated annealing. In *International Conference of the Practice and Theory of Automated Timetabling*, 2016.

[39] O. Czibula, H. Gu, Y. Zinder, and A. Russell. A multi-stage IP-based heuristic for class timetabling and trainer rostering. In *International Conference of the Practice and Theory of Automated Timetabling*, 2014.

[40] O. G. Czibula, H. Gu, F.-J. Hwang, M. Y. Kovalyov, and Y. Zinder. Bi-criteria sequencing of courses and formation of classes for a bottleneck classroom. *Computers & Operations Research*, 65:53–63, 2016.

[41] O. G. Czibula, H. Gu, and Y. Zinder. A Lagrangian relaxation-based heuristic to solve large extended graph partitioning problems. In *WALCOM: Algorithms and Computation*, pages 327–338. Springer, 2016.

[42] O. G. Czibula, H. Gu, and Y. Zinder. Scheduling personnel retraining: Column generation heuristics. In *International Symposium on Combinatorial Optimization*, pages 213–224. Springer, 2016.

[43] O. G. Czibula, H. Gu, and Y. Zinder. Lagrangian relaxation versus genetic algorithm based matheuristic for a large partitioning problem. *Theoretical Computer Science*, 2017.

[44] G. B. Dantzig. Letter to the editor-a comment on edie's traffic delays at toll booths. *Journal of the Operations Research Society of America*, 2(3):339–341, 1954.

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

[46] J. Desrosiers and M. E. Lübbecke. *A Primer in Column Generation*. Springer, 2005.

[47] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[48] M. Dimopoulou and P. Miliotis. Implementation of a university course and examination timetabling system. *European Journal of Operational Research*, 130(1):202–213, 2001.

[49] G. Dobson and R. S. Nambimadom. The batch loading and scheduling problem. *Operations Research*, 49(1):52–65, 2001.

[50] Á. P. Dorneles, O. C. de Araújo, and L. S. Buriol. A fix-and-optimize heuristic for the high school timetabling problem. *Computers & Operations Research*, 52:29–38, 2014.

[51] J. Dreo, A. Petrowski, P. Siarry, and E. Taillard. *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer Science & Business Media, 2006.

[52] Z. Drezner. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 15(3):320–330, 2003.

[53] T. El-Sakka. University course timetable using constraint satisfaction and optimization. *International Journal of Computing Academic Research*, 2015.

[54] M. Elshafei and H. K. Alfares. A dynamic programming algorithm for days-off scheduling with sequence dependent labor costs. *Journal of Scheduling*, 11(2):85–93, 2008.

[55] G. Erdoğan, E. Erkut, A. Ingolfsson, and G. Laporte. Scheduling ambulance crews for maximum coverage. *Journal of the Operational Research Society*, 61(4):543–550, 2010.

[56] 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.

[57] N. Fescioglu-Unver and M. M. Kokar. Self controlling tabu search algorithm for the quadratic assignment problem. *Computers & Industrial Engineering*, 60(2):310–319, 2011.

[58] M. Fischetti and P. Widmayer. Towards solving very large scale train timetabling problems by lagrangian relaxation. In *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, volume 9, 2008.

[59] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12_supplement):1861–1871, 2004.

[60] C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. *Quadratic Assignment and Related Problems*, 16:173–187, 1994.

[61] L. S. Franz and J. L. Miller. Scheduling medical residents to rotations: solving the large-scale multiperiod staff assignment problem. *Operations Research*, 41(2):269–279, 1993.

[62] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[63] C. García-Martínez, F. J. Rodriguez, and M. Lozano. Tabu-enhanced iterated greedy algorithm: A case study in the quadratic multiple knapsack problem. *European Journal of Operational Research*, 232(3):454–463, 2014.

[64] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of NP-completeness, 1979.

[65] J. W. Gavett and N. V. Plyter. The optimal assignment of facilities to locations by branch and bound. *Operations Research*, 14(2):210–232, 1966.

[66] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics*, 10(2):305–313, 1962.

[67] V. Gintner, N. Kliewer, and L. Suhl. Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. *OR Spectrum*, 27(4):507–523, 2005.

[68] F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[69] O. Goldschmidt, D. S. Hochbaum, A. Levin, and E. V. Olinick. The sonet edge-partition problem. *Networks*, 41(1):13–23, 2003.

[70] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, 23(4):665–679, 1976.

[71] C. Gotlieb. The construction of class-teacher timetables. In *Proc. IFIP Congress*, volume 62, pages 73–77, 1963.

[72] M. Guignard. Lagrangean relaxation. *Top*, 11(2):151–200, 2003.

[73] A. Gunawan, K. Ng, and K. Poh. A hybrid algorithm for the university course timetabling problem. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[74] M. Günther and V. Nissen. Combined working time model generation and personnel scheduling. In *Advanced Manufacturing and Sustainable Logistics*, pages 210–221. Springer, 2010.

[75] A. Hiley and B. A. Julstrom. The quadratic multiple knapsack problem and three heuristic approaches to it. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 547–552. ACM, 2006.

[76] H. Hoogeveen. Multicriteria scheduling. *European Journal of Operational Research*, 167(3):592–623, 2005.

[77] IBM. IBM User's Manual for CPLEX. `ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf`, 2015.

[78] IBM. IBM CPLEX Optimizer. `http://www.ibm.com/software/commerce/optimization/cplex-optimizer/`, 2016.

[79] Y. Ikura and M. Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2):61–65, 1986.

[80] ISCO 2016. The 4th International Symposium on Combinatorial Optimization. `http://www.isco2016.it/`, 2016.

[81] H. Isermann. Linear lexicographic optimization. *OR Spectrum*, 4(4):223–228, 1982.

[82] A. Janiak and M. Y. Kovalyov. Single machine group scheduling with ordered criteria. *Annals of Operations Research*, 57(1):191–201, 1995.

[83] A. Janiak, Y. M. Shafransky, and A. Tuzikov. Sequencing with ordered criteria, precedence and group technology constraints. *Informatica, Lith. Acad. Sci.*, 12(1):61–88, 2001.

[84] E. L. Johnson, A. Mehrotra, and G. L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62(1-3):133–151, 1993.

[85] F. Jolai. Minimizing number of tardy jobs on a batch processing machine with incompatible job families. *European Journal of Operational Research*, 162(1):184–190, 2005.

[86] F. Jolai Ghazvini. *Ordonnancement sous Contrainte de Groupage.* PhD thesis, 1998.

[87] B. A. Julstrom. Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 607–614. ACM, 2005.

[88] M. N. M. Kahar and G. Kendall. The examination timetabling problem at universiti malaysia pahang: Comparison of a constructive heuristic with an existing software solution. *European Journal of Operational Research*, 207(2):557–565, 2010.

[89] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu. *VLSI Physical Design: from Graph Partitioning to Timing Closure.* Springer Science & Business Media, 2011.

[90] R. M. Karp. *Reducibility Among Combinatorial Problems.* Springer, 1972.

[91] A. B. Keha, K. Khowala, and J. W. Fowler. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56(1):357–367, 2009.

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

[93] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[94] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[95] U. Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. *Geoinformation und Mobilität-von der Forschung zur praktischen Anwendung*, 22:219–230, 2004.

[96] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963.

[97] E. L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

[98] N. L. Lawrie. An integer linear programming model of a school timetabling problem. *The Computer Journal*, 12(4):307–316, 1969.

[99] S. M. Lee. *Goal Programming for Decision Analysis.* Auerbach Philadelphia, 1972.

[100] J. Leigh, N. Withnell, N. Finnigan, H. Winder, S. O'Flanagan, S. Buliptt, N. Fishburn, S. Drury, and S. Dean. Innovative placement allocation model for pre-registration student nurses. *Journal of Education and Training Studies*, 2014.

[101] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.

[102] H. Li, G. W. Rosenwald, J. Jung, and C.-C. Liu. Strategic power infrastructure defense. *Proceedings of the IEEE*, 93(5):918–933, 2005.

[103] J. Li and C.-C. Liu. Power system reconfiguration based on multilevel graph partitioning. In *PowerTech, 2009 IEEE Bucharest*, pages 1–5. IEEE, 2009.

[104] C.-F. Liaw. Applying simulated annealing to the open shop scheduling problem. *IIE Transactions*, 31(5):457–465, 1999.

[105] L. Liu, C. Ng, and T. E. Cheng. Bicriterion scheduling with equal processing times on a batch processing machine. *Computers & Operations Research*, 36(1):110–118, 2009.

[106] D. Luxen and D. Schieferdecker. Candidate sets for alternative routes in road networks. *Journal of Experimental Algorithmics (JEA)*, 19:2–7, 2015.

[107] D. Marx. Graph coloring problems and their applications in scheduling. In *in Proc. John von Neumann PhD Students Conference.* Citeseer, 2004.

[108] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[109] N. K. Mehta. The application of a graph coloring method to an examination scheduling problem. *Interfaces*, 11(5):57–65, 1981.

[110] S. V. Mehta and R. Uzsoy. Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE Transactions*, 30(2):165–178, 1998.

[111] Minerals Council of Australia. Training and Education Activity in the Minerals Sector. `http://www.minerals.org.au/file_upload/files/reports/Final_Report_Minerals_Council_2013.pdf`, 2013.

[112] S. Mohan. Scheduling part-time personnel with availability restrictions and preferences to maximize employee satisfaction. *Mathematical and Computer Modelling*, 48(11):1806–1813, 2008.

[113] C. A. Morgenstern and H. D. Shapiro. *Chromatic number approximation using simulated annealing*. Department of Computer Science, College of Engineering, University of New Mexico, 1986.

[114] S. C. Murray and G. R. Williamson. Managing capacity issues in clinical placements for pre-registration nurses. *Journal of Clinical Nursing*, 18(22):3146–3154, 2009.

[115] G. L. Nemhauser. *Introduction to Dynamic Programming*. Wiley, 1966.

[116] G. L. Nemhauser and S. Park. A polyhedral approach to edge coloring. *Operations Research Letters*, 10(6):315–322, 1991.

[117] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*, volume 18. Wiley New York, 1988.

[118] L. Nepomuceno, E. C. Baptista, A. R. Balbo, and E. M. Soler. Coevolutionary genetic algorithm based on the augmented lagrangian function for solving the economic dispatch problem. *IEEE Latin America Transactions*, 13(10):3277–3286, 2015.

[119] G. Neufeld and J. Tartar. Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM*, 17(8):450–453, 1974.

[120] K. Nonobe. Inrc2010: An approach using a general constraint optimization solver. *The First International Nurse Rostering Competition (INRC 2010)*, 2010.

[121] C. E. Nugent, T. E. Vollmann, and J. Ruml. An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research*, 16(1):150–173, 1968.

[122] S. Orero and M. Irving. A combination of the genetic algorithm and lagrangian relaxation decomposition techniques for the generation unit commitment problem. *Electric Power Systems Research*, 43(3):149–156, 1997.

[123] I. H. Osman and J. P. Kelly. *Meta-Heuristics: Theory and Applications.* Springer Science & Business Media, 2012.

[124] K. Papoutsis, C. Valouxis, and E. Housos. A column generation approach for the timetabling problem of greek high schools. *Journal of the Operational Research Society*, 54(3):230–238, 2003.

[125] PATAT 2014. The 10th international conference on the practice and theory of automated timetabling. `http://www.patatconference.org/patat2014/`, 2014.

[126] PATAT 2016. The 11th international conference on the practice and theory of automated timetabling. `http://www.patatconference.org/patat2016/`, 2016.

[127] N. Pillay. A survey of school timetabling research. *Annals of Operations Research*, 218(1):261–293, 2014.

[128] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Science & Business Media, 2012.

[129] D. Pisinger and S. Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

[130] Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming.* Springer Science & Business Media, 2006.

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

[132] A. P. Punnen and K. Nair. Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem. *Discrete Applied Mathematics*, 55(1):91–93, 1994.

[133] A. Qualizza and P. Serafini. A column generation scheme for faculty timetabling. In *Practice and theory of automated timetabling V*, pages 161–173. Springer, 2005.

[134] S. M. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, 2014.

[135] R. Ruiz and A. Allahverdi. No-wait flowshop with separate setup times to minimize maximum lateness. *The International Journal of Advanced Manufacturing Technology*, 35(5-6):551–565, 2007.

[136] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[137] T. Saraç and A. Sipahioglu. A genetic algorithm for the quadratic multiple knapsack problem. In *Advances in Brain, Vision, and Artificial Intelligence*, pages 490–498. Springer, 2007.

[138] A. Schaerf. A survey of automated timetabling. *Artificial intelligence review*, 13(2):87–127, 1999.

[139] K. Schimmelpfeng and S. Helber. Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum*, 29(4):783–803, 2007.

[140] G. Schmidt and T. Ströhlein. Timetable construction–an annotated bibliography. *The Computer Journal*, 23(4):307–316, 1980.

[141] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998.

[142] A. Singh and A. S. Baghel. A new grouping genetic algorithm for the quadratic multiple knapsack problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 210–218. Springer, 2007.

[143] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

[144] M. Sol and M. W. P. Savelsbergh. *A branch-and-price algorithm for the pickup and delivery problem with time windows*. Eindhoven University of Technology, Department of Mathematics and Computing Science, 1994.

[145] M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[146] É. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4):443–455, 1991.

[147] E.-G. Talbi. *Metaheuristics: From Design to Implementation*, volume 74. John Wiley & Sons, 2009.

[148] Z. C. Taşkın, J. C. Smith, S. Ahmed, and A. J. Schaefer. Cutting plane algorithms for solving a stochastic edge-partition problem. *Discrete Optimization*, 6(4):420–435, 2009.

[149] D. M. Tate and A. E. Smith. A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, 22(1):73–83, 1995.

[150] The Australian Society for Operations Research. ASOR Recent Advances in Operations Research 2015. `http://www.asor.org.au/files/ra-program_2015-02-08.pdf`, 2015.

[151] J. M. Thompson and K. A. Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research*, 63(1):105–128, 1996.

[152] J. M. Thompson and K. A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25(7):637–648, 1998.

[153] Ö. Ülker, E. Özcan, and E. E. Korkmaz. Linear linkage encoding in grouping problems: applications on graph coloring and timetabling. In *Practice and Theory of Automated Timetabling VI*, pages 347–363. Springer, 2007.

[154] R. Uzsoy. Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33(10):2685–2708, 1995.

[155] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[156] F. Vanderbeck. *Decomposition and Column Generation for Integer Programs.* 1994.

[157] WALCOM 2016. The 10th international workshop on algorithms and computation. `http://walcom2016.aitm.edu.np/`, 2016.

[158] D. J. 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.

[159] M. R. Wilhelm and T. L. Ward. Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, 19(1):107–119, 1987.

[160] L. A. Wolsey. *Integer Programming.* Wiley, 1998.

[161] X. Yao. A new simulated annealing algorithm. *International Journal of Computer Mathematics*, 56(3-4):161–168, 1995.

[162] X. Zhang, J. Zhao, and X. Chen. A hybrid method of lagrangian relaxation and genetic algorithm for solving UC problem. In *2009 International Conference on Sustainable Power Generation and Supply*, pages 1–6. IEEE, 2009.

[163] A. V. Zykina. A lexicographic optimization algorithm. *Automation and Remote Control*, 65(3):363–368, 2004.