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

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

Oliver Czibula · Hanyu Gu · Aaron Russell · Yakov Zinder

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

Keywords Timetabling · Rostering · Integer Programming · Heuristic

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

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

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

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

1 Introduction

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

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

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

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

of students, a module-room pair has a maximum number of students given by the minimum of these two values. While different modules of a course can be assigned to different rooms, it is important that all the modules of a course are assigned to a single location.

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

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

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

The robustness of the training plan is of paramount importance to Ausgrid. Since people are expected to book themselves into classes when needed, a good timetable should exhibit certain characteristics that maximise the likelihood that people will be able to find a class at a suitable time and place. For example, it is desirable for courses to be distributed uniformly throughout the planning horizon. Moreover, a timetable should exhibit robustness by minimising the impact of unforeseen events that may affect the running of courses. One undesirable element in a timetable, and source of uncertainty, is a room swap which happens when consecutive modules of the same course instance are assigned to different rooms. We consider it highly

desirable if all the modules of a course can be run in a single room allowing the students and trainer to remain in one place, uninterrupted, for the whole duration of the course. Conversely, requiring the students and trainer to pack up and migrate to a nearby room is not only disruptive to the flow of the course, but it also detracts from the robustness of the timetable as the room may be unexpectedly unavailable, jeopardising the whole course if a substitute room cannot be found.

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

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

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

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

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

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

2 Literature Review

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

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

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

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

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

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

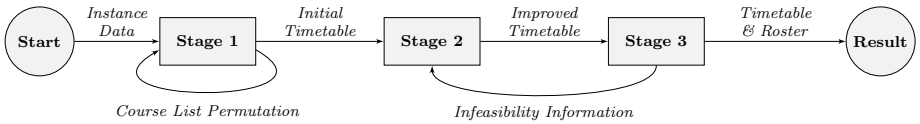

Fig. 1 A high-level view of the three-stage approach.

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

3 Optimisation Procedure

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

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

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

3.1 First stage: Initial timetable construction

In the first stage, all course instances are placed in an ordered list and then arranged on the timetable one at a time. As more courses are placed on the timetable,

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

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

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

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

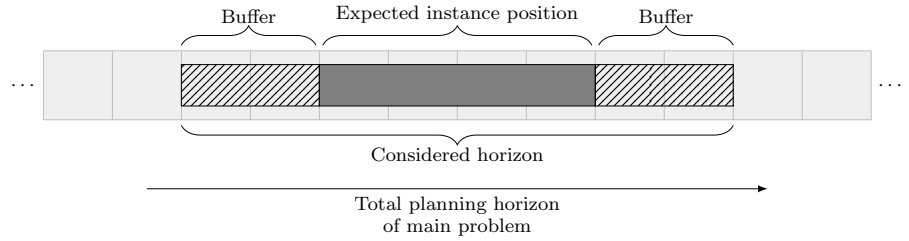

Fig. 2 A buffer added to either end of the reduced timeline.

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

3.2 Second stage: Timetable improvement

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

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

A feasible solution is guaranteed at each iteration, which must be no worse than that of the previous iteration. At iteration $i-1$, most courses were considered as constants in the model, while some were variables that were optimised over the objective function to produce the solution $S^{(i-1)}$. In the subsequent iteration i, a different set of courses are considered constants and a different set are variables, however the model is optimised over the same objective function. Since the solution $S^{(i-1)}$ is feasible (and optimal given the model at iteration $i-1$), $S^{(i-1)}$ must also be feasible at iteration i and will necessarily have the same objective value. However, it is possible that $S^{(i)} \neq S^{(i-1)}$ and then it is possible that the objective

value at iteration i will be better than it was at $i-1$, however it can never be worse.

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

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

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

3.3 Third stage: Rostering

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

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

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

4 Timetabling Model

4.1 Time Discretisation

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

4.2 Input Data Set-up

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

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

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

4.3 List of Symbols

Sets of primary objects:

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

Sets of derived objects:

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

Primary decision variables:

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

Auxiliary variables:

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

$\bar{Y}_{c,i}$ 1 if course $c \in C$ instance $i \in I_c$ runs, or 0 otherwise.

$\tilde{Y}_{c,i,\omega,\xi}$ The number of students expected to sit in course $c \in C$ instance $i \in I_c$ during time window $\omega \in \Omega$ in region $\xi \in \Xi$.

$u_{c,\omega,\xi}$ The number of students not accommodated for course $c \in C$ during window $\omega \in \Omega$ in region $\xi \in \Xi$.

t_m The new room flag for module $m \in \hat{M}$. If the room type for this module is that same type of room as for the previous module, if applicable, then $t_m = 0$, otherwise $t_m = 1$.

$\rho_{r,\lambda}$ The number of rooms of type $r \in R$ occupied during rental window $\lambda \in \Lambda$.

Z_i The ith goal term in the objective function.

Constants:

σ_t The quantity available of resource $t \in T$.

$\delta_{t,l,k}$ The time required (in periods) for a unit of resource $t \in T$ to move from location $l \in L$ to location $k \in L$.

$\theta_{l,d}$ The number of trainers normally allocated to location $l \in L$ on day $d \in D$.

$\theta^{\max}$ The maximum number of additional trainers permitted to any location on any given day.

$\theta^{\min}$ The maximum number of subtracted trainers permitted from any location on any given day.

$Q_{r,p}$ The quantity of room type $r \in R$ available at period $p \in P$, or 0 otherwise.

$d_{r,\lambda}$ The expected revenue from renting out a unit of room type $r \in R$ during rental window $\lambda \in \Lambda$.

l_c The length (in periods) of course $c \in C$.

b_c The minimum class size (in students) required to justify running an instance of course $c \in C$.

π_c The length (in periods) of the rolling time window used to compute the minimum and maximum number of times a course $c \in C$ should be run.

π_c^+ The maximum number of times a course $c \in C$ should be run in any given time window of length π_c.

π_c^- The minimum number of times a course $c \in C$ should be run in any given time window of length π_c.

$s_{c,\omega,\xi}$ The demand (in students) for course $c \in C$ during window $\omega \in \Omega$ in region $\xi \in \Xi$.

α_i The coefficient of the ith goal in the objective function.

4.4 Core Timetabling Constraints

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

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

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

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

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

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

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

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

4.5 Characteristic Constraints

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

4.5.1 Module Positioning Constraints

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

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

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

4.5.2 Capacity Constraints

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

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

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

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

4.6 Trainer Movement Constraints

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

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

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

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

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

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

4.7 Resource Movement Constraints

A network formulation can be leveraged to represent the flow of resources between locations across time in a convenient way. Resources, in this context, refer to mobile pieces of equipment that are required for teaching particular modules. One such

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

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

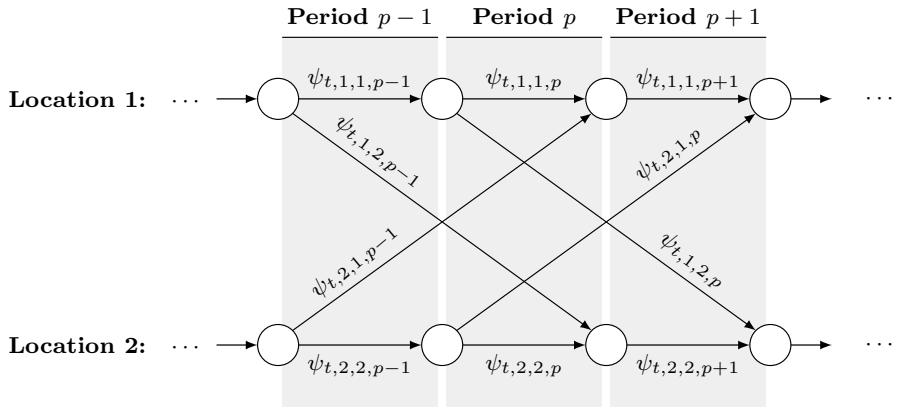

Fig. 3 A sample flow network for some resource t about period p with 2 locations.

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

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

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

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

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

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

4.8 Spreading Constraints

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

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

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

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

4.9 Objective Function

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Rostering Model

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

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

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

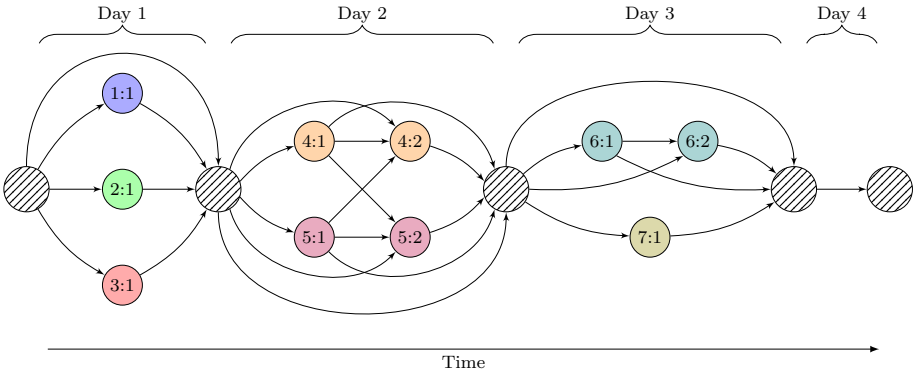

Fig. 5 The flow network corresponding to the sample timetable shown in figure 4. (Home nodes are hatched, and activity nodes are solid)

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

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

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

We introduce the following symbols for the rostering model:

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

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

The flow balance equations for the network are as follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6 Implementation

6.1 Pre-Processing

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

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

6.2 Symmetry Breaking

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

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

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

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

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

7 Computational Experiments

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

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

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

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

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

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

Table 6 Results for the existing software system.

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

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

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

Table 7 Results for the three-stage heuristic.

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

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

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

8 Conclusion(s)

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

References

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

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