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

# Integer Programming for Minimal Perturbation Problems in University Course Timetabling

**Antony E. Phillips · Cameron G. Walker · Matthias Ehrgott · David M. Ryan**

**Abstract** In this paper we present a general integer programming-based approach for the minimal perturbation problem in university course timetabling. This problem arises when an existing timetable contains hard constraint violations, or infeasibilities, which need to be resolved. The objective is to resolve these infeasibilities while minimising the disruption or perturbation to the remainder of the timetable. This situation commonly occurs in practical timetabling, for example when there are unexpected changes to course enrolments or available rooms.

Our method attempts to resolve each infeasibility in the smallest neighbourhood possible, and utilises the exactness of integer programming. Operating within a neighbourhood of minimal size keeps the computations fast, and does not permit large movements of course events, which cause widespread disruption to timetable structure. We demonstrate the application of this method using an example based on real data from the University of Auckland.

**Keywords** University Course Timetabling · Integer Programming · Decision Support Systems

## 1 Introduction

University course timetabling is a well-known problem in which a time period and a room are determined for each course event (e.g. a lecture). This may be conducted prior to the start of enrolment, or after enrolment data is known. The former case is referred to as *curriculum-based timetabling*, because time clashes between courses are determined by sets of courses known as *curricula*. The latter case is referred to

This research has been partially supported by the European Union Seventh Framework Programme (FP7-PEOPLE-2009-IRSES) under grant agreement #246647 and by the New Zealand Government as part of the OptALI project.

A. E. Phillips, C. G. Walker, D. M. Ryan
Department of Engineering Science, The University of Auckland
E-mail: antony.phillips@auckland.ac.nz

M. Ehrgott
Department of Management Science, Lancaster University

as *enrolment-based timetabling* because clashes can be determined and weighted by known enrolments for each course.

In a practical setting, both of these problems are applicable to some extent (Kingston, 2013a). The timetable is typically constructed significantly prior to the start of enrolments, and it will commonly need to be modified as enrolments take place. During each of these phases, the situation can arise where an existing timetable becomes infeasible due to changes in the underlying data. The minimal perturbation problem addresses how to modify an existing timetable so that feasibility is found with a minimal amount of perturbation (or disruption) to the structure of the timetable.

The minimal perturbation problem is first comprehensively addressed in the context of general dynamic scheduling (El Sakkout, Richards, and Wallace, 1998; El Sakkout and Wallace, 2000). El Sakkout and Wallace (2000) propose an algorithm based on constraint programming techniques, which leverages the efficiency of linear programming to solve part of the problem.

Barták, Müller, and Rudová (2004) are the first to study minimal perturbation problems in the context of university course timetabling, using a constraint satisfaction heuristic combined with a branch-and-bound process. The authors continue this work with a local search-based metaheuristic, known as "iterative forward search", which improves performance significantly (Müller, Rudová, and Barták, 2005). Finally, Rudová, Müller, and Murray (2011) present a summary of this approach as part of a broader course timetabling process, which is implemented at Purdue University, USA. This includes detailed results on the iterative forward search algorithm as applied to minimal perturbation problems, and is described in a practical setting.

Kingston (2013b) addresses a similar problem in the context of high school timetabling, proposing an ejection chain heuristic method.

In this paper we present a new general method for solving minimal perturbation problems which arise in practical timetabling. Around each infeasibility, we define a small *neighbourhood* of events, time periods, and rooms, which we are willing to perturb. Within this neighbourhood we solve an integer programme which attempts to resolve the infeasibility with minimal disruption to existing timetable structure. Utilising the exactness of integer programming, we will only expand the size or scope of the neighbourhood when we have certainty that the current neighbourhood is insufficient to resolve the infeasibility. This process aims to ensure the computational tractability of each integer programme. Limiting the neighbourhood size is also desirable because it prevents large movements of course events, which are seen as disruptive to the timetable structure.

We demonstrate the application of this method using an example based on real data from the University of Auckland. The expanding neighbourhood methodology has been successfully demonstrated in other real-world applications (Rezanova and Ryan, 2010).

## 2 Minimal Perturbation Problems in University Course Timetabling

A complete solution to the university course timetabling problem is a timetable which specifies a time period and room for every course event. The solution can

be considered feasible if it does not include any violated hard constraints, or *infeasibilities*. Quality measures, or soft constraints, are desirable features of a feasible solution which may also be considered. For a coverage of commonly used hard and soft constraints we refer to the benchmarking paper by Bonutti, De Cesco, Di Gaspero, and Schaerf (2012).

University course timetabling is widely accepted to be a dynamic problem in practice, where data may continually change throughout construction and implementation of a timetable (McCollum, 2007; Kingston, 2013a). For complex timetabling at large universities, we broadly discuss how minimal perturbation problems can arise in each stage of the timetabling process. We draw on our own experiences at the University of Auckland, which bears many similarities to other large universities considered in the timetabling literature.

The early construction phase of timetabling occurs when most of the data has been gathered, and construction of a timetable is starting. Most time or room assignments are considered tentative, and may be changed relatively freely. At this stage, almost any changes to the data are possible e.g. new or removed courses, changes to staff employment status, room availabilities etc. Some infeasibilities may not need to be resolved until the data is more complete.

The late construction phase occurs when the timetable is close to being finalised for publication. This stage is the most similar to the curriculum-based timetabling problem addressed in the literature. At the University of Auckland, time assignments are determined in close collaboration with faculties, to satisfy their specific and complex requirements. In this case, changing the time period for an event would be disruptive, whereas the room assignment may be more freely perturbed. Major changes to the data are less likely at this stage, and all infeasibilities should be resolved. We note that infeasibilities may also arise due to the method of constructing a timetable, rather than solely due to changes in the data. For example, if faculties choose their own time-assignments independently (often "rolled-forward" from the previous year with changes), this can produce a timetable for which there is no feasible room assignment.

The enrolment phase of timetabling begins once the timetable is published, and students have started to select courses. At this stage, the most common changes to the data involve adjusting the expected course enrolments to actual course enrolments, which may cause some room assignments to be no longer suitable. In this phase, it can be disruptive to change either the time period or the room assignment for an event. However, the former is likely to be particularly disruptive, as students and staff may have external obligations affecting their personal timetable. We note that in the case of the University of Auckland, because of legal requirements it is not possible to have any excess or overflow of students in a room. Although not all events will be attended by every enrolled student (e.g. sickness, retention, recorded lectures), the first events of a semester are typically well attended.

In practice, the specific timing and cause of an infeasibility can significantly affect the choice of which perturbations we are willing to perform. The algorithm we present in this paper was motivated by two particular practical situations. The first arises in the late construction phase of timetabling, where faculties have approved a high quality time assignment for all their events, but we require perturbation in order for a feasible room assignment to exist. Secondly, we address the ubiquitous problem in the enrolment phase of timetabling, where unpredictable course enrolments have caused existing room assignments to become unsuitable.

## 3 Room Assignment

This section gives a brief overview of our room assignment algorithm. As mentioned in Section 2, many minimal perturbation problems in the construction phase and particularly the enrolment phase of timetabling, involve an infeasible room assignment. Although the room assignment algorithm is not the focus of this paper, attempting to solve a room assignment for an infeasible timetable will reveal important information about the nature of the infeasibilities. For a detailed coverage of the room assignment process, we refer to Phillips, Waterer, Ehrgott, and Ryan (2014).

The relationship between the room assignment and timetable perturbation process is shown in Figure 1. Each rectangular box corresponds to solving an integer programme to maximise the room assignment quality with respect to a particular objective. We maximise each objective sequentially, and fix its value in subsequent iterations, known as *lexicographic optimisation* (Ehrgott, 2005). A lexicographic approach implies a strict precedence of the importance of quality measures, which is appropriate in this case.

Firstly we would like to maximise the number of events which are assigned to a feasible room, or *event hours*. Due to the exactness of integer programming, if we do not find a room for all events, we can be certain that no such complete room assignment exists. This means we will need to perform a timetable perturbation in order to find a feasible room assignment.

However, we would first like to improve the value of other objectives. We next maximise the *seated student hours* which aims to assign larger events to a room, in preference to smaller events. This means the unassigned events will be of the smallest size possible. Thirdly, we maximise the *seat utilisation* where events are placed in rooms which "fit" well, i.e. most of the seats are occupied. This means that any rooms which are left unused (in each time period) will be the largest possible. Finally we maximise the *building preference* which favours holding events in rooms which are geographically close to the associated teaching department.

The resulting partial room assignment provides us with essential information. Firstly, it shows which time periods contain unassigned events (infeasibilities), and therefore require solving the minimal perturbation problem. Secondly, the partial room assignment contains information which helps to focus our neighbourhood search to restore feasibility (see Section 4.4).

## 4 Minimal Perturbation

### 4.1 Expanding Neighbourhood Algorithm

Solving the minimal perturbation problem requires assigning both a time period and a room for each event, rather than handling these problems separately. However, building a model with variables indexed over all events, time periods and rooms could easily result in millions of variables (Burke, Mareček, Parkes, and Rudová, 2008), which would be intractable. As a result, we would like to build a model which resolves each infeasibility in as small a *neighbourhood* as possible. The neighbourhood around an infeasibility is defined by a restricted set of events which can be moved, and subsets of time periods and rooms to which events can

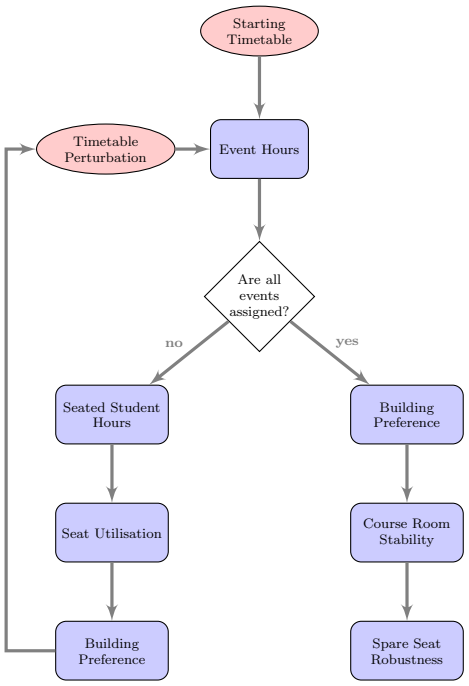


**Fig. 1** University of Auckland Timetabling Process 2010

be moved. All events outside this neighbourhood are *fixed*. Based on information from the partial room assignment, the specific composition of a neighbourhood is heavily customised to the nature of the infeasibility.

In many universities, it is common for rooms to be utilised in approximately 50% of available time periods (Beyrouthy, Burke, Landa-Silva, McCollum, McMullan, and Parkes, 2007). Therefore, it is very likely that a feasible timetable will exist where all events are held in a suitable time period and room. Furthermore, whether the infeasibility arises from rolling forward an old timetable with changes, or if there are unexpected post-enrolment changes, it is likely that the infeasible timetable will be "close" to feasibility, i.e. only a small number of events will need to change time period or room.

We will initially generate a very small neighbourhood, where we are only willing to move events of a similar size to our unassigned event(s), and only to/from the time periods one hour before and after their current time (for example). Within this neighbourhood we solve an integer programme (IP) to reassign all neighbourhood events in the least disruptive way that removes the infeasibility. If this is not possible, we will expand the scope of the neighbourhood, and solve again until the infeasibility can be resolved. Although several iterations of this process may be required, each IP model will be relatively small due to the optimistic methodology of starting with a small neighbourhood. Because we use an exact algorithm to attempt resolving the infeasibility, we know with certainty whether a feasible reassignment exists in this neighbourhood or not. This is an advantage over using a

heuristic method, where it can be difficult to determine whether a given problem is infeasible or whether this heuristic is unable to find a solution.

When we need to expand the neighbourhood, this is done in a similar way to how the starting neighbourhood is constructed, in the sense that the expansion rules will prioritise adding the most promising variables or options first. Once we have found a feasible timetable and room assignment within this neighbourhood, we can apply this solution to the timetable, and proceed to resolve any other infeasibilities. This algorithm is presented as Algorithm 1.

**Algorithm 1** Expanding Neighbourhood Algorithm

```
for all Infeasible Time Periods do
  N ← GenerateInitialNeighbourhood()
  searching ← True
  while searching do
    IP ← BuildNeighbourhoodIP(N)
    IP.Solve()
    if IP is Feasible then
      Timetable.Update()
      searching ← False
    else
      N.Expand()
    end if
  end while
end for
```

Finally, it is worth recalling that we are not only looking for a feasible solution; we are also aiming to minimise the timetable disruption. Although the latter is our objective function within the neighbourhood, in some cases a feasible timetable and room assignment can be found within a given neighbourhood, but only through significant disruption to the underlying timetable. In this case, it may be possible to expand the neighbourhood further, and find a solution with a lower disruption. This is desirable, if it is computationally feasible to explore the larger neighbourhood size.

### 4.2 Event-based Neighbourhood IP

Within a given neighbourhood, we solve an integer programme to attempt to find a feasible and low disruption assignment of events to time periods and rooms. As mentioned in Section 4.1, significant attention is paid to the scope of the neighbourhood, in terms of which (re)assignments we will consider. All sets defining a neighbourhood are listed in Table 1, where variables are generated over the sets $E$, $T_e$ and $R_{et}$. Many of the complex constraints relating to either the time or room assignments (e.g. clashes between courses, staff requirements etc.), are implicitly represented by the variable generation.

Control over the neighbourhood sets offers significant modelling power. In particular, for many courses it is required that a maximum of one event is held per day. If a *fixed* event (i.e. a 'non-neighbourhood' event) from a course $c$ is held on day $d$, then time periods on this day will be excluded from $T_e$ for any other events of this course $e \in E_c$. Similarly, a *curriculum* is a group of courses which cannot be

held in the same time period (as they are taken by a common group of students). If an event from curriculum *curr* is fixed in a time period within the neighbourhood, then no events from courses in this curriculum can be moved to this time period. If we are solving a problem in the enrolment phase of timetabling, the curricula can be determined by actual student enrolments. It is important that no enrolled student has a timetable which becomes infeasible after the minimal perturbation problem is solved.

Timetable elements which span multiple consecutive time periods (e.g. two-hour lectures) are generalised as sets of events known as *long events*. If a long event lies only partially in the neighbourhood, it is not permitted to be moved to ensure the time stability. In many situations, it is also required to ensure room stability across all time periods of a long event. In this case, if the long event is partially in the neighbourhood and already has a room, it is excluded. If it is partially in the neighbourhood but does not have a room assigned, the neighbourhood must expand to include the full long event. In our model we have assumed that both time and room stability are required.

| | | | |
|---|---|---|---|
| $E$ | events in neighbourhood | $T_e$ | time periods suitable for event $e$ |
| $C$ | courses | $T$ | time periods in neighbourhood |
| $CU$ | curricula | $D$ | days of neighbourhood time periods |
| $E_c$ | events of course $c$ | $T_d$ | time periods on day $d$ |
| $E_{curr}$ | events of courses in curriculum *curr* | $R_{et}$ | rooms suitable for event $e$ and available in time period $t$ |
| $E_F$ | events which are single-period or the first of a long event | | |
| $E_{tr}$ | events suitable for assignment to time period $t$ and room $r$ | | |

**Table 1** Notation for Neighbourhood Sets

Using the notation defined in Table 1, we present an integer programming formulation of an *event-based* neighbourhood perturbation model. In this formulation, the binary variables $x_{etr}$ are indexed by suitable event-time-room assignments. Specifically, let the variable $x_{etr}$ take the value 1 if event $e \in E$ is to be held at time $t \in T_e$ in room $r \in R_{et}$. For a given weighting of penalties, $v_{etr}$, an optimal assignment of events to time periods and rooms can be determined by solving the following integer programme (1)–(7). The formulation is considered event-based because the penalty for reassigning an event is independent of the other neighbourhood events from the same course. The disruption penalties for an event can vary depending on the number of time periods it moves, whether the room changes, and how this relates to any fixed events from this course. With sufficient available data, precise penalties can be specified for each disruption.

$$\text{minimise} \quad \sum_{e \in E} \sum_{t \in T_e} \sum_{r \in R_{et}} v_{etr} * x_{etr} \tag{1}$$

$$\text{subject to:} \sum_{e \in E_{tr}} x_{etr} \leq 1 \qquad t \in T, r \in R_t \tag{2}$$

$$\sum_{t \in T_e} \sum_{r \in R_{et}} x_{etr} = 1 \qquad e \in E \tag{3}$$

$$\sum_{\substack{e \in \\ (E_c \cap E_F)}} \sum_{\substack{t \in \\ (T_e \cap T_d)}} \sum_{r \in R_{et}} x_{etr} \leq 1 \qquad c \in C, d \in D \tag{4}$$

$$\sum_{e \in E_{curr}} \sum_{r \in R_{et}} x_{etr} \leq 1 \qquad curr \in CU, t \in T \tag{5}$$

$$x_{etr} - x_{(e-1)(t-1)r} = 0 \qquad e \in (E \setminus E_F), t \in T_e, r \in R_{et} \tag{6}$$

$$x_{etr} \in \{0, 1\} \qquad e \in E, t \in T_e, r \in R_{et} \tag{7}$$

The objective function (1) minimises the total timetable *disruption* between the proposed timetable solution, and the initial (infeasible) timetable.

Constraints (2) ensure that each room in each time period is occupied by a maximum of one event, while constraints (3) ensure that all events are assigned to exactly one room in one time period. Constraints (4) ensure that two events from the same course cannot move to any time period on the same day. Because long events are represented as more than one individual event $e \in E$, only the first event $e$ in any long event is included in each constraint. Constraints (5) ensure that two events from the same curriculum cannot move to the same time period. Lastly, constraints (6) enforce the strict time stability and room stability on the events of a long event.

If room stability is not required for a long event (i.e. it is acceptable to change rooms between the individual event-hours), constraints (6) can be altered such that each constraint is summed over all suitable rooms, rather than applied as one constraint per room.

### 4.3 Course-based Neighbourhood IP

Although the event-based formulation is versatile in terms of representing penalties, it is not able to model *time stability* for a course. This is a common quality measure where it is considered desirable to schedule all weekly events from a course at the same time of day. To build upon the simpler event-based formulation (1)–(7), we define the following: $C_{stab}$ is the set of courses which desire time stability, $H$ is the set of hours from all neighbourhood time periods, $c_e$ is the course with which event $e$ is associated, and $h_t$ is the hour of the day for time period $t$. Let the variable $y_{ch}$ take the value 1 if *any* event of course $c$ is held in hour $h$ in the timetable.

For a given weighting of time stability penalties, $w_{ch}$, we can solve the following course-based integer programme.

$$\text{minimise} \quad (1) + \sum_{c \in C} \sum_{h \in H} w_{ch} * y_{ch} \tag{8}$$

subject to: (2)–(7)

$$x_{etr} - y_{c_e h_t} \leq 0 \qquad e \in E, t \in T_e, r \in R_{et} \tag{9}$$

$$y_{ch} \in \{0, 1\} \qquad c \in C_{stab}, h \in H \tag{10}$$

The additional term in the objective function (8) penalises each course for each unique hour of the day it uses for any of its events. Constraints (9) appropriately tie the values of the $y_{ch}$ variables to the $x_{etr}$ variables.

This formulation requires a different starting neighbourhood and set of expansion rules to the event-based model. Because entire courses are required to move together, the initial neighbourhood should not focus around a single infeasible time period, but rather all weekly infeasible time periods for an hour of the day.

### 4.4 Limiting the Neighbourhood

As mentioned in Section 4.1, it is desirable to keep the neighbourhood as small as possible, as determined by the sizes of the sets in Table 1. This necessitates a selective definition of the neighbourhood at each stage of expansion to include the most promising variables (i.e. event-time-room allocations) in the model.

The initial definition and expansion rules for a neighbourhood are guided by the existing partial room assignment, which can offer substantial insight into the specific cause of the infeasibility. By maximising the *seated student hours*, we ensure that any unassigned events will be as small as possible. Therefore, if we observe that large events remain unassigned, we can infer that the associated time periods are in shortage of large rooms. As a result, when expanding the neighbourhood we know to focus on events which currently occupy large rooms, and ideally we can expand into time periods with vacant large rooms. Without maximising the seated student hours, unassigned large events could be due to a general lack of rooms of any size.

The room assignment process also maximises the *seat utilisation*, where it is favourable to assign events to rooms which are closely matched in size. This optimisation is important, particularly for time periods which do not contain an unassigned event themselves, but are adjacent or near to time periods with unassigned events. In the case of a full room assignment for a particular time period, the previous maximisations (of event hours and seated student hours) will permit assigning small events in larger rooms than necessary, as long as it is still possible to assign all events. Maximising the seat utilisation has two useful outcomes for the neighbourhood search. Firstly, where possible, it will leave larger rooms vacant as they are more flexible. Secondly, by assigning events into closely fitting rooms, the neighbourhood expansion can delay incorporating an event and room assignment which "fit" well, since the room is already well utilised. The last quality measure in the room assignment is the *building preference*, which can also be useful when prioritising which event-to-room assignments should enter the neighbourhood first.

Finally, we note that for some practical situations, infeasibilities may be caused by a shortage of a particular type of room (e.g. computer laboratories) or room

attribute (e.g. piano, fume cupboard), rather than a specific size of room. In this situation, the neighbourhood definition will include rooms from other time periods with this specific feature, even if they are presently occupied by a closely fitting event. The room assignment process may even be altered to include a quality measure which favours assigning events with specific room requirements. This helps identify which room features are in shortage, and is analogous to maximising seated student hours to identify the critical room sizes.

## 5 Practical Example

To demonstrate our algorithm, we consider a scenario based on the University of Auckland's timetabling data from Semester 2 in 2010. The University of Auckland timetable features 2131 events, 72 rooms, and 50 weekly time periods (8am to 6pm, from Monday to Friday). Like many universities, the average room is utilised in approximately 60% of time periods. However, the utilisation is above 80% in "peak" time periods, which are typically between 10am and 3pm. The utilisation of the largest rooms is also above average, due to a long-term increase in student enrolments.

Computational experiments are run using Gurobi 5.0 running on 32-bit Ubuntu 12.04, with a quad-core 3.33GHz processor (Intel i5-660).

### 5.1 Over-enrolment Example

In this example, we analyse the problem of unexpectedly high enrolment numbers in the enrolment phase of timetabling. We assume that the revised enrolment numbers are received prior to the start of semester, but clearly after the timetable is originally constructed. As a result, it is considered disruptive to make changes to the time at which events are held, although changing room assignments is more acceptable. Specific penalties are given for each of the approaches in Sections 5.2 and 5.3.

The example scenario is given in Table 2, where the enrolment numbers for an introductory sociology and law course are substantially larger than expected. The courses have two and three events respectively, in the time periods listed. These are the type of courses which are likely to have an unpredictable enrolment, as they are available to new-entrant students and may be taken as electives by students from many academic programmes. In cases when enrolments are only marginally larger than expected, ideally the existing rooms will still be suitable. The room assignment process (Figure 1) maximises the *spare seat robustness* objective, for this purpose.

| Course Name | Time Periods | Planned Enrolment | Revised Enrolment |
|---|---|---|---|
| SOCIO 100 | Mon 12pm, Thu 2pm | 320 | 500 |
| LAW 121G | Mon 12pm, Wed 12pm, Fri 12pm | 269 | 500 |

**Table 2** Scenario changes to course enrolments

Initially, the state of having five events (from two courses) with no assigned room is identified as the infeasibility in the timetable. In order to resolve this infeasibility, we will first re-solve the room assignment for this timetable, as explained in Section 3. This will only change the room assignment for a very small number of events, which are held in the same time periods as the infeasibilities. If there are suitable vacant rooms in these time periods, or if it is possible to re-assign other events to "free up" such rooms, we will have a feasible room assignment without needing to perturb the time assignments at all. For this problem, maximising the event hours proves it is not possible to leave fewer than five events unassigned, without perturbing the timetable. This is because the events are held in highly congested time periods where all large rooms are presently occupied. However, by maximising the seated student hours, we are able to change our partial room assignment so that the five unassigned events are those which have the smallest number of students. The five events which remain unassigned are given in Table 3, where it can be seen that three of the events from the "SOCIO 100" and "LAW 121G" courses have been replaced by smaller events from other courses.

| Time Period | Course Name | Enrolment |
|---|---|---|
| Mon 12pm | SOCIO 100 | 500 |
| Mon 12pm | THEOL 101 | 490 |
| Wed 12pm | LAW 121G | 500 |
| Thu 2pm | POLIT 113 | 347 |
| Fri 12pm | BIOSCI 203 | 360 |

**Table 3** Events without a room after room assignment

### 5.2 Event-based Perturbation

We first attempt to solve the problem of unassigned events (Table 3) using an event-based IP formulation (1)–(7) within the expanding neighbourhood algorithm (Algorithm 1). Around each of the four time periods, we initialise a neighbourhood to include similar sized events in one time period before and after. If this is insufficient to resolve the infeasibility, we expand the neighbourhood to include events from two time periods before and after. We set a simple penalty (for each event) of one unit for each hour moved and two units for each day moved from the original time period. There is a very small penalty for moving rooms within the same time period, so that this can occur, but only when necessary.

The solution to this problem is given in Table 4, where events are relocated to nearby time periods as shown, for a total penalty of 7. The events for "SOCIO100", "LAW121G" and "POLIT113" are able to move to other time periods with suitable vacant rooms in their respective neighbourhoods. The events for "THEOL101" and "BIOSCI203" are able to stay in their time periods and move into the rooms vacated by "ECON151G" and "PSYCH204" respectively. This manoeuvre commonly occurs in solutions to these problems, where one event changes time periods to free its room for another event. This is because each event has a unique set of time periods to which it can move, as determined by curricula and

other requirements. Some events inevitably have a greater flexibility or a different set of suitable time periods.

| | Monday | Tuesday | Wednesday | Thursday | Friday |
|---|---|---|---|---|---|
| 10am | | | | | |
| 11am | | | | | |
| 12pm | SOCIO100<br>THEOL101<br>ECON151G | | LAW121G | | POLIT113 |
| 1pm | | | | | |
| 2pm | | | | BIOSCI203<br>PSYCH204 | |
| 3pm | | | | | |

**Table 4** Event-based solution

Solving the 6 IPs required for this problem was very rapid, as they all featured less than 500 binary variables, and terminated optimally in less than one second.

### 5.3 Course-based Perturbation

Because many faculties appreciate time stability for their courses, we also demonstrate the course-based IP formulation (1)–(10). In the previous solution (Table 4), perturbations to the events from several courses (e.g. "ECON151G", "LAW121G", and "POLIT113") incurred an unmeasured disruption to time stability. For the course-based model we use a different starting neighbourhood which includes all time periods at this same time of day across the week. This neighbourhood then expands to include all time periods one hour before and after. Because the first expansion causes the neighbourhood to include a total of 15 time periods, it is important to be selective about which events are included in each time period. To keep the model size manageable, we only include large events and rooms.

The solution to this problem is given in Table 5, where events are relocated to nearby time periods as shown for a total *event-based penalty* of 8. This is a greater penalty than in the event-based solution, however we are now able to resolve the infeasibility with no penalty incurred from disruption to the time stability.

Solving the 3 IPs required for this problem was again rapid, although there were up to 5000 binary variables in the largest case. However, due to the near-naturally integer property of these models, they still terminated optimally in less than one second.

## 6 Conclusion and Future Work

We have proposed a general integer programming-based approach for minimal perturbation problems which arise in practical university course timetabling. This

| | Monday | Tuesday | Wednesday | Thursday | Friday |
|---|---|---|---|---|---|
| 10am | | | | | |
| 11am | | | | | |
| 12pm | SOCIO100<br>THEOL101<br>ECON151G | | LAW121G<br>ECON151G | | LAW121G<br>POLIT113 |
| 1pm | | | | | |
| 2pm | | | | BIOSCI203 | |
| 3pm | | | | | |

**Table 5** Course-based solution

approach is versatile, as there are substantial possibilities for customisation in the way the neighbourhoods are constructed and expanded. We have shown an example application of this process on real data from the University of Auckland.

An interesting extension to the proposed formulations would be the incorporation of multi-objective optimisation techniques, where additional quality measures are considered explicitly rather than implicitly in terms of disruption. This also leads to the question of whether it is possible (and desirable) to disrupt a given feasible timetable in order to improve the room assignment. Finally, we would like to consider equity and fairness across faculties and courses when making timetabling perturbations.

## References

Barták R, Müller T, Rudová H (2004) A new approach to modeling and solving minimal perturbation problems. In: Apt KR, Fages F, Rossi F, Szeredi P, Vncza J (eds) Recent Advances in Constraints, Lecture Notes in Computer Science, vol 3010, Springer Berlin, pp 233–249

Beyrouthy C, Burke EK, Landa-Silva D, McCollum B, McMullan P, Parkes AJ (2007) Towards improving the utilization of university teaching space. Journal of the Operational Research Society 60(1):130–143

Bonutti A, De Cesco F, Di Gaspero L, Schaerf A (2012) Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Annals of Operations Research 194(1):59–70

Burke EK, Mareček J, Parkes AJ, Rudová H (2008) Uses and abuses of MIP in course timetabling. Poster at the workshop on mixed integer programming, MIP2007, Montréal, 2008, available online at `http://cs.nott.ac.uk/jxm/timetabling/mip2007-poster.pdf`

Ehrgott M (2005) Multicriteria optimization, 2nd edn. Springer Berlin

El Sakkout H, Wallace M (2000) Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 5(4):359–388

El Sakkout H, Richards T, Wallace M (1998) Minimal perturbation in dynamic scheduling. In: Prade H (ed) Proceedings of the 13th European Conference on Artifical Intelligence, ECAI-98

Kingston JH (2013a) Educational timetabling. In: Uyar AS, Ozcan E, Urquhart N (eds) Automated Scheduling and Planning, Studies in Computational Intelligence, vol 505, Springer Berlin, pp 91–108

Kingston JH (2013b) Repairing high school timetables with polymorphic ejection chains. Annals of Operations Research pp 1–16

McCollum B (2007) A perspective on bridging the gap between theory and practice in university timetabling. In: Burke EK, Rudová H (eds) Practice and Theory of Automated Timetabling VI, Lecture Notes in Computer Science, vol 3867, Springer Berlin, pp 3–23

Müller T, Rudová H, Barták R (2005) Minimal perturbation problem in course timetabling. In: Burke EK, Trick M (eds) Practice and Theory of Automated Timetabling V, Lecture Notes in Computer Science, vol 3616, Springer Berlin, pp 126–146

Phillips A, Waterer H, Ehrgott M, Ryan DM (2014) Integer programming methods for large scale practical classroom assignment problems. Submitted to Computers & Operations Research

Rezanova NJ, Ryan DM (2010) The train driver recovery problem - a set partitioning based model and solution method. Computers & Operations Research 37(5):845–856

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