Solving the Classroom Assignment Problem Using Integer Programming

Antony Phillips, David Ryan
Department of Engineering Science
University of Auckland
New Zealand
antony.phillips@auckland.ac.nz

Matthias Ehrgott
Department of Management Science
Lancaster University
United Kingdom

Abstract

In this paper we present an integer programming method for solving the Classroom Assignment Problem in University Course Timetabling. We introduce a novel formulation of the problem which generalises existing models and maintains tractability even for large instances.

We also address how the structure of different classroom assignment problems can affect the relative difficulty of finding and confirming an optimal solution. For situations where a high quality solution is required quickly, we develop a method based on constraint branching which significantly reduces the problem size while attempting to retain the best solutions.

Our model and methods are validated through computational results based on our experiences at the University of Auckland, and on instances from the 2007 International Timetabling Competition.

Key words: University course timetabling, classroom assignment, integer programming, constraint branching

1 Introduction

University course timetabling is a large resource allocation problem, in which both times and rooms are determined for each class meeting. Significant research has been devoted to Curriculum-Based Course Timetabling (CB-CTT) in particular, because of its importance for universities worldwide.

Due to the difficulty and size of modern timetabling problems, much of the literature proposes purely heuristic solution methods. However, in recent years, integer programming (IP) methods have been the subject of increased attention.

We refer the reader to a recent survey which covers the most popular IP methods and heuristic paradigms (MirHassani & Habibi, 2013).

One decomposition used in course timetabling is to generate a timetable first, followed by a classroom assignment second. This is commonly used in practice because the time elements of a timetable involve complex institution-specific requirements, over which experienced administrators and teaching staff would like to maintain control. In some institutions, the classroom assignment problem is the only part of constructing a timetable which uses computer-aided decision making.

Most older formulations of the classroom assignment problem use a simple measure of quality which allows each time period to be considered independently (e.g. Ferland & Roy, 1985). These formulations can be modelled as an assignment problem, which can be solved in polynomial time. More recent formulations are able to address complex measures of quality which cause interdependencies between time periods, such as providing the same room for all classes from the same course (e.g. Lach & Lübbecke, 2012). However, this causes the problem to be NP-complete (Carter & Tovey, 1992).

In this paper we present a novel integer programming based method for the classroom assignment problem of university course timetabling. We particularly focus on applying the method of constraint branching to improve the tractability of the most difficult NP-complete instances of classroom assignment. Our method is validated through computational results on data from the University of Auckland and from the 2007 International Timetabling Competition (ITC).

The remainder of this paper is organised as follows. Section 2 outlines an integer programming model for solving general classroom assignment problems. Section 3 provides an insight into the matrix structure of the integer programme and demonstrates how fractions can arise in the linear programming relaxation. In Section 4 we present a strategy for solving the most difficult room assignment problems based on constraint branching, with validating results in Section 5. Finally, Section 6 outlines the main conclusions of our work, and future research directions.

2 A Pattern-Based Set Packing Model

In this section we model (class)room assignment as a set packing problem which is solvable using integer programming.

2.1 Notation

A teaching *event* e is a meeting between staff and students (e.g. a lecture), which requires a room for the duration of one time period in the timetabling domain (typically one week). Let E denote the set of events. A *course* c is a set of events which require a room of size at least $size_c$ (measured by the number of seats) and possessing at least the room attributes att_c. Let C denote the set of all courses.

A meeting *pattern* p is defined to be a subset of events for a given course that will be assigned the same room. For course c, let P_c denote the set of all its patterns, the power set of course events. Let $length_c$ and $length_p$ denote the number of events in a course and pattern respectively. As a power set, P_c will feature $2^{length_c} - 1$ elements, which potentially could be large. However, in practice, the number of events per course is usually quite small (for example, averaging between 2 and 3 at

the University of Auckland). Let P denote the set of all patterns, i.e. $P = \bigcup_{c \in C} P_c$. Note that while each pattern p uniquely identifies a set of events, an event is usually in more than one pattern. Let P_e denote the set of all patterns which contain event e, i.e. $P_e = \{p \in P \colon e \in p\}$.

Let R denote the set of *rooms* in the pool of common teaching space, where $size_r$ and att_r correspond to the room size and set of room attributes for a room r. R_c represents the set of rooms which are suitable for events of course c, i.e. $R_c = \{r \in R \colon size_r \geq size_c, att_r \supseteq att_c\}$. Note that any pattern p of course c will have the set of feasible rooms for this pattern, R_p, equal to R_c.

Let A denote the set of all room attributes, i.e. $A = \bigcup_{r \in R} att_r$.

Let T denote the set of all usable time *periods* in the timetabling domain, which are of a common duration (often one hour) and are non-overlapping. Each event $e \in E$ occurs during a prescribed time period T_e given by the timetable. Classes which are two or more periods long (e.g. tutorials or labs) require one event $e \in E$ for each time period they are held in. To enforce a stable room across the long class, the set of patterns for this course, P_c, is pruned to only include patterns which contain all or none of these events. Because all events of a pattern are assigned to the same room, this enforces the contiguous room stability requirement.

Finally, let P_{rt} denote the set of all patterns which include an event in time period t, and for which room r is suitable, i.e. $P_{rt} = \{p \in P \colon r \in R_p, (\exists e \in p \colon t = T_e)\}$.

2.2 Integer Programming Formulation

Using the notation defined in Section 2.1, we present an integer programming formulation of a pattern-based set packing model for room assignment. In this formulation, the binary variables x_{pr} are indexed by feasible pattern-to-room assignments. Specifically, let the variable x_{pr} take the value 1 if pattern $p \in P$ is to be held in room $r \in R_p$. For a given objective function w (representing some measure of solution quality), an optimal assignment of patterns to rooms can be determined by solving the following integer programme (1)—(5).

$$\text{maximise} \quad \sum_{p \in P} \sum_{r \in R_p} w_{pr} x_{pr} \tag{1}$$

$$\text{subject to} \quad \sum_{p \in P_{rt}} x_{pr} \leq 1, \qquad r \in R,\ t \in T \tag{2}$$

$$\sum_{p \in P_e} \sum_{r \in R_p} x_{pr} = 1, \qquad e \in E \tag{3}$$

$$\sum_{p \in P_c} x_{pr} \leq 1, \qquad c \in C,\ r \in R_c \tag{4}$$

$$x_{pr} \in \{0, 1\}, \qquad p \in P,\ r \in R_p \tag{5}$$

Constraints (2) ensure that at most one event is assigned to each room in each period, while constraints (3) ensure that exactly one room is assigned for each event. Constraints (4) ensure that each course uses at most one pattern per room, i.e. all events from a course that are assigned to a room, should be part of the same (maximal) pattern.

The model is referred to as pattern-based because P contains all patterns of events for each course. However, depending on the objective function w, it is possible

to formulate the model with a restricted set $\bar{P} \subseteq P$ of patterns.

If $\bar{P}$ is restricted to patterns which correspond to a single event, i.e. $\bar{P} = E$, then the *event-based* model is obtained. This can be used for any solution quality measure which relates to the suitability of a room for a particular event i.e. event-based measures. This is contrasted to pattern-based measures which relate to the suitability of a room for a set of course events. For an event-based model we must omit constraints (4) which are only valid when events belong to more than one pattern.

2.3 Measures of Solution Quality

There are many, sometimes conflicting, measures of solution quality which can be either event- or pattern-based. We define two common quality measures; *room preference* which is event-based and *course room stability* which is pattern-based. If we need to optimise a pattern-based measure, a pattern-based model is required. Event-based measures, however, can apply to either an event- or pattern-based model. Note that each event-based objective coefficient includes the term $length_p$, which provides linear scaling for when p contains more than one event.

The following definitions give the objective coefficients w_{pr} for all patterns p and all rooms r. For these definitions, assume that c corresponds to the course of which pattern p is a part.

Room preference (RP) Maximise the total course-to-room preference over all events assigned a room. This preference may be determined at the department-to-building level i.e. all courses from each department have the same preference for all rooms from each building. At the University of Auckland, the preference is derived from the building's location relative to the teaching department's offices, and may take the value -1, 0 or 1 to indicate undesirability, indifference, or preference.

$w_{pr} = length_p * Pref(c, r)$

Course room stability (RS) Minimise the total number of different rooms, assigned to each course, over all courses. The disruption to the room stability of a course by one of its patterns is given by $(length_c - length_p)/length_c$. In a feasible room assignment, the sum of these fractions by patterns of a course will sum to the number of additional rooms used, relative to the target '1 room per course'. For example, a course with 3 events could use just 1 pattern (all events in the same room), 2 patterns (2 events in the same room, 1 in a different room), or 3 patterns (each event in a different room). Using the disruption formula, the first case with 1 pattern causes a disruption of zero since no additional rooms are used. The second case will cause a disruption of 1/3 for the larger pattern, and 2/3 for the smaller pattern, summing to 1 additional room. The 3 patterns of the final case disrupt stability by 2/3 each, summing to 2 additional rooms.

$w_{pr} = -(length_c - length_p)/length_c$

3 Computational Difficulty

The simplest class of room assignment problems are those which can be formulated with an event-based model ($\bar{P} = E$). For any objective function (1), the constraint matrix defined by (2)—(3) (since (4) is invalid) of this problem is known to be totally unimodular. Because there are no interdependencies between periods, each period may be solved as an assignment problem in polynomial time (Carter & Tovey, 1992).

The more difficult class of room assignment problems are those which require a pattern-based model ($\bar{P} = P$), because they consider a pattern-based quality measure such as *course room stability*. This problem is shown to be NP-complete in general (Carter & Tovey, 1992). To model course room stability as a quality measure, for each course we must generate a pattern for each subset of course events (as per Section 2.1), for each room.

A minimal example of a difficult pattern-based problem is shown as Example 1. This problem is specified in Table 1, where for each course c, a tick indicates which of the three time periods it features in, and feasible rooms are given as R_c. For this problem, it is easy to find a feasible room assignment, however it is not possible to offer a stable room to all courses. For our formulation defined by (1)—(5), the fractionating part of the constraint matrix is shown in Figure 1. Only constraints (2) (identified by the period & room) and variables which relate to 'pattern 3' (both events) for each course are shown. The other patterns, corresponding to the first event only ('pattern 1') and the second event only ('pattern 2'), are not shown.

Example 1. A minimal pattern-based problem featuring fractionating 3-order 2-cycles

c	t_1	t_2	t_3	R_c
A	✓	✓		1,2
B		✓	✓	1,2
C	✓		✓	1,2

Table 1: Time Periods and Feasible Rooms

$$
\begin{array}{c}
 \\
t_1r_1 \\
t_2r_1 \\
t_3r_1 \\
t_1r_2 \\
t_2r_2 \\
t_3r_2 \\
\dots \\
\\
x
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
Ap_3r_1 & Bp_3r_1 & Cp_3r_1 & Ap_3r_2 & Bp_3r_2 & Cp_3r_2 & \dots
\end{array} \\
\left[\begin{array}{cccccccc}
1 & & 1 & & & & \\
1 & 1 & & & & & \\
 & 1 & 1 & & & & \\
 & & & 1 & & 1 & \\
 & & & 1 & 1 & & \\
 & & & & 1 & 1 & \\
 & & & & & &
\end{array}\right] \\
\begin{array}{cccccccc}
0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 &
\end{array}
\end{array}
\leq
\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \dots \end{bmatrix}
$$

Figure 1: Partial Constraint Matrix and LP Relaxation Solution

Solving the IP (with partial constraint matrix shown in Figure 1) to maximise the course room stability, will find an optimal solution with quality of -1 (a penalty of 1). However, the LP relaxation is able to find an optimal solution with quality of

0 (no penalty), with each 'pattern 3' variable taking the value 0.5 (those shown in Figure 1), and other variables taking the value 0.

For this problem, integer solutions incur a penalty because they require at least one of the courses to use the undesirable 'pattern 1' and 'pattern 2' variables, which assign two events from the same course to different rooms. The constraint matrix in Figure 1 shows how the desirable 'pattern 3' variables of each course can form odd-order 2-cycles (shaded), which are known to permit fractional solutions to occur (Berge, 1972; Ryan & Falkner, 1988).

The requirements for this type of cycle (Figure 1) to form are that there must be three courses which share two common feasible rooms and each course must feature in a different two of three time periods. To induce an integrality gap with respect to the course room stability objective, there must also be a relative shortage of available feasible rooms for these courses, in the particular time periods. This could be due to a generally high utilisation rate over all rooms, or because particular sizes and types of rooms are in shortage. If a third room was introduced into Example 1 which was feasible for even one of the courses, the same cycles would exist, yet there would no longer be an integrality gap.

Because these cycles can be formed over *any* three time periods, there are many opportunities for such cycles to occur in general room assignment problems. The courses are even not required to have the same number of events as one another, because the pattern-based model generates variables for every subset of a course's events. As a result, courses with a large number of events contribute significantly to the number of cycles which can occur.

The relationship between courses with many events and an increased potential for fractions can also be understood through the *subsequences* of the constraint (2) rows. The subsequences for constraint $t_x r_z$ (occupancy of room r_z in period t_x), are all constraints on the occupancy of the same room (r_z) in a later time period t_y, and for which there is at least one course holding an event in each period.

For a room in the first time period, the subsequence count will be the number of other periods which contain an event from any course also in the first time period. In this sense, the subsequence count can be viewed as the connectivity (through courses of multiple events) of a time period to other time periods. If all rows of a matrix have at most one subsequence, this matrix is said to have unique subsequence, and will be totally unimodular i.e. the LP relaxation is naturally integer. Conversely, the higher the subsequence count, the more likely we are to see fractional solutions to the LP relaxation (Ryan & Falkner, 1988). Consequently, it is clear that large courses with many events contribute to the subsequence count, degree of fractionality, and ultimately the difficulty of the problem.

4 Constraint Branching Techniques

As demonstrated in Section 3, a high degree of connectivity between time periods causes odd order cycle-induced fractions in the LP relaxation of a room assignment IP. Traditional variable branching may be ineffective in such a case because the zero side of the branch simply removes one pattern-to-room assignment, which is unlikely to impact the objective when there are many possibilities for the odd order cycles to reappear.

A method to reduce the fractionality of the LP is to use a *constraint branch*

which involves branching on whether one of a particular set of variables will satisfy a pair of constraints (Ryan & Foster, 1981). In this case we are applying constraint branches as a *dive* i.e. we only explore one side of the branch. The aim is ultimately to find a reduced subset of the original problem variables which is substantially more tractable to solve, and is likely to include many high quality solutions.

The use of a constraint branch is demonstrated in Example 2, an expanded version of Example 1, with an optimal IP objective of -3. In this case, the optimal LP solution has an objective value of 0, and corresponds to all maximal-pattern variables (denoted p_F) equally spread in quarters across all 4 rooms. The 5-order cycle which permits this is shown in Figure 2 (rooms 3 to 5 are not shown). For this solution, there are several possible constraint branches which can be applied. Considering the first two constraints t_1r_1 and t_2r_1, we note that the values of those variables which feature in both constraints sum to 0.75. This number, referred to as the 'sum of fractions' suggests that these two constraints could be satisfied together i.e. room r_1 will be occupied by events from a common course in each of t_1 and t_2. This branch removes not just the Ap_Fr_1 and Bp_Fr_1 variables shown, but another 44 variables corresponding to non-maximal patterns for all five courses for room r_1. These variables can be efficiently removed from the LP so that the solver only needs to perform a small number of fast iterations to find a new optimal feasible solution. Repeating this process (on this example) of finding a constraint branch with a high sum of fractions will eventually lead to a solution with most variables at integer values except one small cycle of fractional variables similar to that in Example 2. In this case, the highest sum of fractions is 0.5 and so there is no clear candidate branch. At this stage, the reduced problem is solved using an IP solver (e.g. Gurobi) which is rapidly able to explore the few possibilities for fractions remaining and return an optimal solution.

Example 2. 5 course constraint branch

c	t_1	t_2	t_3	t_4	t_5	R_c
A		✓	✓	✓	✓	1,2,3,4
B	✓		✓	✓	✓	”
C	✓	✓		✓	✓	”
D	✓	✓	✓		✓	”
E	✓	✓	✓	✓		”

Table 2: Time Periods and Feasible Rooms

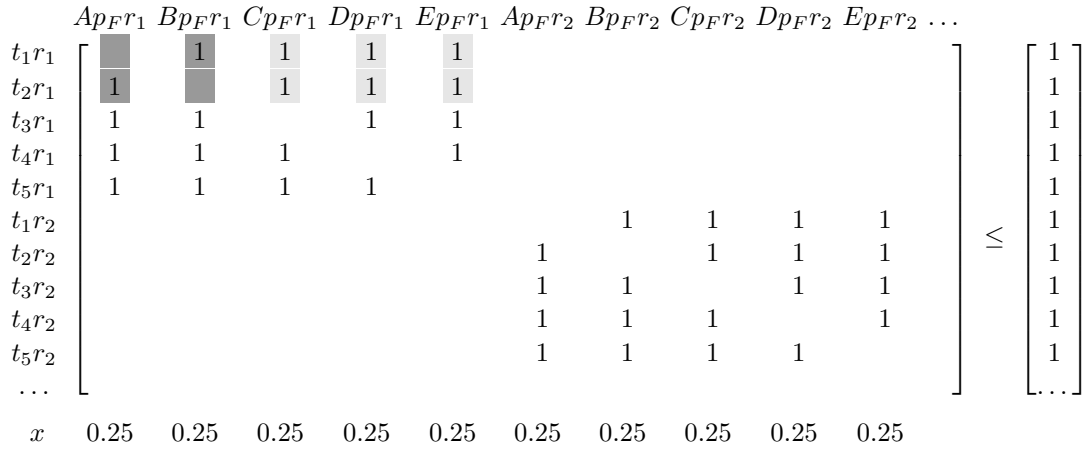

Figure 2: Set Packing Partial Constraint Matrix

In more complex problems than Example 2, there may be many candidates for which constraints to branch on. Rather than exclusively relying on the sum of fractions value, we can apply our own understanding of the problem in practice, to help improve our branching decisions. One such method is to examine the connectivity between the two time periods in question and see specifically how many of these connecting courses could feasibly be held in each room. A very high degree of connectivity between two constraints, coupled with a high sum of fractions, suggests that this room will be able to host the same course in these two periods in an optimal solution. For our computational tests in Section 5, we use a strategy where we branch on the constraint pair with the highest connectivity, provided the sum of fractions is greater than 0.75 (and less than 1). Once there are no more candidate branches meeting this criteria, we solve the resulting reduced IP.

At each LP solution, we will also find constraint pairs with a sum of fractions of 1. For these, we can add a branch to ensure they are satisfied together. Although this is not a branch in the sense that it does not stop affect the current LP solution, it does aid in reducing the number of potential variables.

5 Results

In this section we present the computational results of our constraint branching method on a pattern-based model minimising the course room stability penalty, for comparison against solving the full IP directly. The constraint branching strategy used is outlined in Section 4.

The problems titled *UOA-S1* and *UOA-S2* are from the University of Auckland, where the room assignment is solved for faculty-prepared timetables from Semesters 1 and 2 in 2010. The remaining 20 problems are from the ITC in 2007, which are the main benchmark problems addressed in the university course timetabling literature (Bonutti, De Cesco, Di Gaspero & Schaerf, 2012). Timetables for the ITC problems were generated using a simple IP which ensured a feasible room assignment would exist, but made no provisions for a high quality room assignment.

All computational experiments were run using Gurobi 5.1 running on 64-bit Debian 7, with a quad-core 3.33 GHz processor (Intel i5-2500K). The time limit was set at 3600 seconds.

Problem			Full IP			Reduced IP			
Name	Util	Conn	Pen	Time_{END}	Time_{BST}	%Reduc	Pen	Time_{END}	Time_{BST}
UOA-S1-2010	0.66	0.33	94	130.0	130.0	70	97	13.0	13.0
UOA-S2-2010	0.62	0.34	82	490.9	490.9	67	87	7.0	5.0
comp02	0.71	0.62	0	2.7	2.7	55	1	0.7	0.7
comp03	0.63	0.58	0	1.6	1.6	44	0	0.2	0.2
comp04	0.64	0.66	3	20.28	19.0	68	4	1.0	1.0
comp05	0.47	0.15	3	0.1	0.1	79	3	0.0	0.0
comp06	0.80	0.71	8	3600.0	1109.0	67	9	5.5	4.0
comp07	0.87	0.76	13	3600.0	2179.0	69	13	191.8	18.0
comp08	0.72	0.71	4	3600.0	1626.0	78	8	6.5	3.0
comp09	0.62	0.68	0	0.8	0.8	56	1	5.0	5.0
comp10	0.82	0.69	2	3600.0	883.0	60	5	22.6	19.0
comp11	0.72	0.33	8	0.4	0.4	96	9	0.0	0.0
comp12	0.55	0.20	2	0.2	0.2	42	2	0.1	0.1
comp13	0.65	0.71	1	3600.0	277.0	71	4	5.7	2.0
comp14	0.65	0.56	0	1.0	1.0	64	2	0.1	0.1
comp15	0.63	0.58	0	1.9	1.9	66	1	0.5	0.5
comp16	0.73	0.75	1	354.2	61.0	58	3	8.1	8.1
comp17	0.80	0.73	2	3600.0	613.0	34	3	3600.0	13.0
comp18	0.43	0.14	2	0.1	0.1	61	2	0.0	0.0
comp19	0.69	0.60	0	3.2	3.2	53	1	299.6	299.6
comp20	0.82	0.70	17	3600.0	121.0	55	12	2077.6	168.0
comp21	0.73	0.70	0	10.7	10.7	57	1	1.5	1.5

Table 3: Comparison of methods for maximising course room stability

The results are listed in Table 3, where the first three columns give the problem name, the room utilisation (number of events per room availability), and the average time period connections (as a percentage of all connectable time periods). Columns 4, 5 and 6 give the course room stability penalty (lower is better), time to optimality (or timeout) in seconds, and the time to find the best incumbent solution, for solving the full integer programme (1)—(5). Column 7 relates to the reduced IP obtained by constraint branching, and gives the percentage reduction in the number of variables relative to the full IP. Columns 8, 9 and 10 give the same three quantities for the reduced IP as those listed for the full IP.

In general, the reduced IP is significantly faster to solve, yet often gives a solution with a higher penalty. A closer look shows that the 7 hardest problems, which take the full 3600 seconds for the full IP to solve, are solved well by the constraint branch scheme and a high quality solution found rapidly in all cases. The problems which the full IP takes an intermediate length of time to solve (e.g. between 10 seconds and 10 minutes) show a fair result for the reduced IP with a typically much faster solve time and slightly greater penalty. The problems which are rapidly solved by the full IP are also quickly solved with the reduced IP, but for a greater penalty. However, these problems are clearly not the use-case for our constraint branching scheme, if the full IP can be rapidly solved.

These results are what we would have expected, as the constraint branching method is essentially a tool to make large and difficult problems more tractable. The majority of solve times for the reduced problem are approximately on the scale which a human would be willing to wait while using automated timetabling software (e.g. less than 5 minutes), although the solve process could be terminated early if required. The critical remaining question to answer is whether we can predict how difficult a given problem will be before solving, and so decide whether to tackle the full IP directly or use constraint branching. Fortunately, we have accurate means of obtaining such an *a priori* prediction, based principally on the room utilisation and the time period connectivity. As demonstrated in Section 3, these factors ultimately cause difficult integer programmes, which our empirical results appear to verify.

6 Concluding Remarks and Future Directions

We have introduced a novel formulation for room assignment problems, where our pattern-based model is sufficiently versatile to model many different measures of solution quality. Through analysis of the integer programme matrix we are able to identify the situations where fractional solutions can arise in the LP relaxation, causing the problems to become significantly more difficult. To deal with the most difficult instances of room assignment, we have proposed a constraint branching method which is able to significantly reduce the problem size and run-time to find a high quality solution. This approach has been validated on problems from the University of Auckland, and from the ITC.

The results are promising both in terms of identifying when a problem is likely to be difficult, and then how to efficiently reduce a problem in this case. The latter of these in particular is the subject of continued research, such as additional factors to aid selection of the best branch, and more sophisticated methods such as dynamically changing the branching criteria.

References

Berge, C. (1972). Balanced matrices. *Mathematical Programming*, *2*(1), 19–31.

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.

Carter, M. W. & Tovey, C. A. (1992). When is the classroom assignment problem hard? *Operations Research*, *40*, S28–S39.

Ferland, J. A. & Roy, S. (1985). Timetabling problem for university as assignment of activities to resources. *Computers & Operations Research*, *12*(2), 207–218.

Lach, G. & Lübbecke, M. E. (2012). Curriculum based course timetabling: new solutions to Udine benchmark instances. *Annals of Operations Research*, *194*(1), 255–272.

MirHassani, S. & Habibi, F. (2013). Solution approaches to the course timetabling problem. *Artificial Intelligence Review*, *39*(2), 133–149.

Ryan, D. M. & Falkner, J. C. (1988). On the integer properties of scheduling set partitioning models. *European Journal of Operational Research*, *35*(3), 442–456.

Ryan, D. M. & Foster, B. A. (1981). In A. Wren (Ed.), *An integer programming approach to scheduling*, Computer scheduling of public transport urban passenger vehicle and crew scheduling (pp. 269–280). North Holland, Amsterdam.