Dantzig-Wolfe Decomposition of the Daily Course Pattern Formulation for Curriculum-Based Course Timetabling

Bagger, Niels-Christian Fink; Sørensen, Matias; Stidsen, Thomas Jacob Riis

Published in:
European Journal of Operational Research

Link to article, DOI:
10.1016/j.ejor.2018.06.042

Publication date:
2018

Document Version
Peer reviewed version

Citation (APA):
Bagger, N-C. F., Sørensen, M., & Stidsen, T. J. R. (2018). Dantzig-Wolfe Decomposition of the Daily Course Pattern Formulation for Curriculum-Based Course Timetabling. *European Journal of Operational Research*, *272*(2), 430-446. https://doi.org/10.1016/j.ejor.2018.06.042

Dantzig-Wolfe Decomposition of the Daily Course Pattern Formulation for Curriculum-Based Course Timetabling

Niels-Christian F. Bagger, Matias Sørensen, Thomas R. Stidsen

PII: S0377-2217(18)30579-4
DOI: 10.1016/j.ejor.2018.06.042
Reference: EOR 15227

To appear in: *European Journal of Operational Research*

Received date: 6 February 2017
Revised date: 21 June 2018
Accepted date: 22 June 2018

Highlights

- A new Dantzig-Wolfe Decomposition and Column Generation algorithm is proposed.
- Novel preprocessing and inequality generation is applied to the pricing problem.
- Local Branching is applied when searching for columns with a negative reduced cost.
- The algorithm improves the best-known lower bounds for benchmark instances.

Dantzig-Wolfe Decomposition of the Daily Course Pattern Formulation for Curriculum-Based Course Timetabling

Niels-Christian F. Bagger[a,b,*], Matias Sørensen[a,b], Thomas R. Stidsen[a]

[a]*mORetime research group, Management Science, Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 426B, 2800 Kgs. Lyngby, Denmark, http://www.moretime.man.dtu.dk/*
[b]*MaCom A/S, Vesterbrogade 48, 1., 1620 København V, Denmark*

Abstract

In this paper, we considered the problem of Curriculum-Based Course Timetabling, i.e., assigning weekly lectures to a time schedule and rooms. We developed a Column Generation algorithm based on a pattern formulation of the time scheduling part of the problem by Bagger et al. (2016). The pattern formulation is an enumeration of all schedules by which each course can be assigned on each day; it is a lower bounding model. Pattern enumeration has also been considered in Burke et al. (2008), where the authors enumerated all schedules to which each curriculum can be assigned on each day. We applied the Dantzig-Wolfe reformulation, so each column corresponded to a schedule for an entire day.

We solved the reformulation with the Column Generation algorithm, where each pricing problem generated a full schedule for a single day. We provided a pre-processing technique that, on average, removed approximately 45% of the pattern variables in the pricing problems. We then extended the pre-processing technique into inequalities that we added to the model. Lastly, we describe how we applied Local Branching to the pricing problem by using the columns generated in previous iterations.

We compare the lower bounds we obtained, with other methods from literature, on 20 data instances of real-world applications. For 16 instances the optimal solutions are known, but the remaining four are still *open*. Our approach improved the best-known lower bound for all four open instances, and decreased the average gap from 24% to 11%.

Keywords: Timetabling, Integer Programming, Education, Column Generation, Local Branching

*Corresponding author
Email addresses: `nbag@dtu.dk,bagger.nc@gmail.com` (Niels-Christian F. Bagger), `ms@macom.dk,sorensen.matias@gmail.com` (Matias Sørensen), `thst@dtu.dk` (Thomas R. Stidsen)
URL: `orcid.org/0000-0003-4665-6761` (Niels-Christian F. Bagger)

1. Introduction

In this paper we focus on the Curriculum-Based Course Timetabling problem (CCT) as described by Di Gaspero et al. (2007). The problem has received much attention as it was used for the Second International Timetabling Competition in 2007 (ITC2007) (Di Gaspero et al., 2007; McCollum et al., 2010). Following the competition, a website (Bonutti et al., 2017) was created where researchers can upload data instances, solutions and bounds. Most of the work conducted on CCT is dominated by heuristic approaches (Asín Achá and Nieuwenhuis, 2014; Lübbecke, 2015). Heuristic methods have provided most of the best-known solutions according to Bonutti et al. (2017). For ITC2007, 21 data instances were provided arising from real-world applications. Four of these instances are still *open*, meaning that the best-known upper bound does not equal the best-known lower bound. The lower bounds are necessary, as the heuristics themselves do not provide a quality measurement. Our goal, in this work, is to strengthen the best-known lower bounds. We have done this by applying a Dantzig-Wolfe reformulation of a previous formulation, which is solved by Column Generation. We assume that the reader is familiar with the Dantzig-Wolfe reformulation, the Column Generation algorithm, and the terms used in it: (restricted) master problem, pricing problem, and reduced costs. Interested readers can refer to Martin (1999, chapter 11) and Desrosiers and Lübbecke (2010) for thorough and general descriptions. Column generation approaches have been considered before by Cacchiani et al. (2013), but it is still worthwhile to investigate such methods further (Lübbecke, 2015).

We have described the problem in detail in Section 1.1, and provided an overview of other methods in literature that have considered CCT in Section 1.2. We have described the pattern formulation suggested by Bagger et al. (2016) in Section 2, as this is the model which we have used in our Dantzig-Wolfe reformulation. In the pattern formulation, each column corresponds to an assignment of lectures for a course on a single day. In our reformulation, each column corresponds to the full assignment of the patterns on a single day of all the courses. We have described this reformulation in Section 3. We describe the pre-processing techniques we have applied in Section 4, as well as some inequalities we have derived followed by the framework we have used in the solution process: Local Branching (Fischetti and Lodi, 2003). We have reported the results of our computational experiments in Section 5. Lastly, we have provided the conclusion in Section 6.

1.1. Curriculum-Based Course Timetabling

The CCT problem consists of the following entities: courses, days, time slots, lecturers, rooms and curricula. Each course is taught by exactly one lecturer and contains lectures that must all be scheduled in a weekly timetable and assigned to rooms. The week is divided into days and each day is divided into time slots of equal duration. A day and time slot pair is referred to as a period, so the total number of periods is the number of days multiplied by the number of time slots. The length of one lecture corresponds to one period. A curriculum is a set of courses, where for every pair of courses, there are students attending both courses.

The hard constraints are as follows: all lectures of a course must be scheduled, and they must be scheduled in distinct periods. If a lecture is not scheduled, then the Lectures (**L**) constraint is violated, and two lectures of the same course scheduled in the same period is also considered as a violation. A course can have specific periods defined as unavailable periods. Every lecture scheduled in such a period is a violation of the Availability (**A**) constraint. We can say that two courses are *conflicting* if they are taught by the same lecturer, or belong to the same curriculum, as scheduling two courses in the same periods would create a conflict. If there exists a period where two *conflicting* courses have a lecture scheduled, then the constraint Conflicts (**C**) is violated. A room can accommodate only one lecture in any given period. If more than one lecture is scheduled in the same room and the same period, then the constraint Room Occupancy (**RO**) is violated.

The problem contains four soft constraints: Room Capacity (**RC**), Room Stability (**RStab**), Minimum Working Days (**MWD**), and Isolated Lectures (**IL**). We are allowed to schedule any course in any room. However, a room is desired to be capable of accommodating as many students as possible when scheduling the courses into rooms. Every room has a capacity, and if the number of students attending a lecture is larger than the capacity of the room to which the lecture is assigned, the constraint Room Capacity (**RC**) is violated by the number of students minus the capacity. Furthermore, as the courses contain multiple lectures, it can also be an advantage that the lectures are all scheduled in the same room during the week. For every course, the constraint Room Stability (**RStab**) is violated by one for every distinct room to which the course is assigned minus one. For every course, it is preferred to spread the lectures across a predetermined number of days. This number is called *minimum working days*. If the lectures are scheduled in fewer days than the minimum working days, then the constraint Minimum Working Days (**MWD**) is violated by one for each day below the minimum working days. The last soft constraint is the Isolated Lectures (**IL**) constraint. If two periods belong to the same day and are in consecutive time slots, then we say that the periods are *adjacent*. Consider a curriculum and a course belonging to the curriculum. If the course has a lecture scheduled in a period, and no lecture from any of the courses belonging to the curriculum has been scheduled in an adjacent period, then we say that the lecture is *isolated*. For every curriculum, the constraint Isolated Lectures (**IL**) is violated by one for every isolated lecture.

The **IL** constraint is usually referred to as the *curriculum compactness* constraint in literature. We use the name *isolated lectures* as Bonutti et al. (2012) mentions different ways of defining *curriculum compactness*, and they use the name *isolated lectures* for the formulation used here and in ITC2007.

Any feasible timetable must fulfill all the hard constraints, i.e., a timetable is considered feasible if, and only if, no hard constraints are violated. The objective is to find a feasible timetable while minimizing the soft constraints. Each soft constraint has a weight associated with it, so that a single objective is defined by a weighted sum of all the soft constraints.

1.2. Related Work

In this section, we describe the approaches from literature that consider CCT. As our method is a lower bounding method, we focus on other lower bounding methods for CCT in literature. We refer to Bettinelli et al. (2015) for a comprehensive overview.

Burke et al. (2010a) introduced an exact mixed integer programming (MIP) model of CCT. They formulated the **IL** constraint by using a variable for each curriculum and each period. Burke et al. (2008) removed these variables and instead they used just one variable for each curriculum and each day. The value of this variable was then calculated by adding an exponential number of constraints. The constraints are generated by enumerating patterns for each curriculum and each day. In their study, Burke et al. (2012) kept a subset of the constraints from the enumerated patterns in Burke et al. (2008), and then added the remaining ones dynamically, whenever they were violated. Burke et al. (2010b) took the model from Burke et al. (2010a) and split it into two stages; first, the courses were scheduled into periods and then they were assigned to rooms. This approach was executed iteratively.

Splitting the problem into two stages was also considered by Lach and Lübbecke (2008, 2012): the first stage schedules the courses to periods and assigns them as per capacities, and the second stage then assigns the rooms with respect to the assigned capacities.

Hao and Benlic (2011) considered the first stage problem of Lach and Lübbecke (2012). They relaxed some of the constraints, so that the problem could be divided into sub-problems. Then, they computed a lower bound for each sub-problem and summed them up to get a lower bound for the overall problem.

Cacchiani et al. (2013) also computed lower bounds. They did this by splitting the problem into two parts: one part considered the time related constraints and the other part considered the room related constraints. A lower bound was then calculated by summing up the lower bounds for both parts. As some of the data instances were computationally time consuming to solve for the time related part, they applied a Dantzig-Wolfe reformulation, so that there was a pricing problem for each day and solved the model by column generation.

Asín Aschá and Nieuwenhuis (2014) proposed multiple reformulations of the decision variant as a propositional satisfiability test (SAT). They started by treating the soft constraints as hard constraints and solved the problem as a pure satisfiability problem. Then, they *relaxed* the constraints one by one, to move toward a weighted partial maximum satisfiability encoding.

In their study, Bagger et al. (2017) considered decomposition of the problem similar to Lach and Lübbecke (2012) and Burke et al. (2010b) where the problem was split into a time scheduling model and a room allocation model. The two models were then reconnected by an underlying flow problem to get an exact formulation.

In Bagger et al. (2016)'s research, the time scheduling part of the problem was considered, i.e., the room assignment was disregarded. Here a pattern formulation was suggested where each variable corresponds to a time schedule for one course on one day. The main difference between the formulation by Burke et al. (2008) and Bagger et al. (2016), was that Burke et al. (2008) added the patterns as constraints, whereas Bagger et al. (2016) added the patterns as variables. In the time schedule, it is possible to ensure that all the hard constraints, **L**, **A**, **C**, and **RO**, are fulfilled, which has also also been noted by Lach and Lübbecke (2012). For the soft constraints, Bagger et al. (2016) only accommodated the constraints **MWD** and **IL**, meaning that the model was a lower bounding model for the overall problem.

In this paper, we have applied the Dantzig-Wolfe decomposition to the formulation presented by Bagger et al. (2016). We have then solved the reformulation by a column generation algorithm. We have decomposed the model, so that there is one pricing problem per day. Note that this is similar to the work by Cacchiani et al. (2013). One of the main differences is that we have included the **RO** constraints, which means that we can guarantee that there exists a feasible room assignment for any integer solution we obtain. Another difference is the pricing problems. In each pricing problem, we have a binary variable for each course and each feasible pattern that the course can be assigned for an entire day, whereas Cacchiani et al. (2013) considered a formulation where they had a binary variable for each course and each period. The benefit of the pattern formulation is that when we apply the pre-processing by Bagger et al. (2016), we remove patterns from the pricing problems, which will never be included in any columns. In a formulation similar to Cacchiani et al. (2013), the patterns that are removed in our formulation can potentially be generated by the pricing problems. This makes the pattern-based formulation stronger, in the sense that the objective value of the *Linear Programming* (LP) relaxation of our master problem will be at least as large as the LP relaxation of the master problem by Cacchiani et al. (2013).

2. Pattern Formulation

In this section, we have provided an overview of the pattern formulation provided by Bagger et al. (2016). A pattern represents a schedule of lectures of a course to periods for an entire day. Burke et al. (2008, 2012) reported that there are an exponential number of patterns. However, as most of the data instances, used in the literature, have five or six time slots for each day, then the number of patterns for each course and each day is only 32 or 64. For each course and each day, we have a binary variable for each feasible schedule (pattern) of lectures of the given course on the given day. Before we describe the pattern formulation, we start by providing the notation used throughout this paper. The set of courses, days and time slots are denoted as $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{T}$ respectively. The combination of a day $d \in \mathcal{D}$ and time slot $t \in \mathcal{T}$ is known as a period. For a time slot $t \in \mathcal{T}$, the time slot that is right before t is denoted by $t-1$ and the time slot right after t is denoted by $t+1$. The set of curricula is denoted by $\mathcal{Q}$, and for each curriculum $q \in \mathcal{Q}$, the set $\mathcal{C}_q \subseteq \mathcal{C}$ is the set of courses that belongs to the curriculum q.

For each course $c \in \mathcal{C}$, the number of lectures to schedule is given by the parameter L_c, and the requested minimum number of working days is given by the parameter $D_c^{\min}$. For each curricula

$q \in \mathcal{Q}$, we define the parameter L_q as the total number of lectures that must be scheduled for the courses $\mathcal{C}_q$, i.e., $L_q = \sum_{c \in \mathcal{C}_q} L_c$. The total number of rooms available is denoted by R. Lastly, for each course $c \in \mathcal{C}$, day $d \in \mathcal{D}$ and time slot $t \in \mathcal{T}$ the parameter $F_{c,d,t}$ is one if the course is available in the corresponding period; otherwise, it is zero. If time slot $t \in \mathcal{T}$ is the first time slot, then the parameter $F_{c,d,t-1}$ is defined as zero and likewise if t is the last time slot, the parameter $F_{c,d,t+1}$ is zero for each course $c \in \mathcal{C}$ and day $d \in \mathcal{D}$.

As all periods are uniform, it is only necessary to generate different patterns that are possible for the set of time slots $\mathcal{T}$ once and then apply them to each course and day. An example of all the patterns is illustrated in Table 1 when $|\mathcal{T}| = 4$.

Table 1: Illustration of all the patterns for $|\mathcal{T}| = 4$. Each column corresponds to a pattern, and each row corresponds to a time slot. The symbol "$\times$" indicates whether or not a pattern schedules a lecture in the corresponding time slot.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		$\times$				$\times$	$\times$	$\times$				$\times$	$\times$	$\times$		$\times$
1			$\times$			$\times$			$\times$	$\times$		$\times$	$\times$		$\times$	$\times$
2				$\times$			$\times$		$\times$		$\times$	$\times$		$\times$	$\times$	$\times$
3					$\times$			$\times$		$\times$	$\times$		$\times$	$\times$	$\times$	$\times$

The set of all the patterns is denoted by $\mathcal{K}$. For each pattern $k \in \mathcal{K}$ and time slot $t \in \mathcal{T}$, the parameter a_t^k is set to one if k contains a lecture in t. If $t \in \mathcal{T}$ is the first time slot, then a_{t-1}^k is defined as zero and likewise if t is the last time slot, then a_{t+1}^k is defined to be zero. The number of lectures contained in pattern $k \in \mathcal{K}$ is denoted as L_k, i.e., $L_k = \sum_{t \in \mathcal{T}} a_t^k$. For each course $c \in \mathcal{C}$ and day $d \in \mathcal{D}$, the set $\mathcal{K}_{c,d} \subseteq \mathcal{K}$ denotes the set of patterns that c can be assigned on day d. A pattern $k \in \mathcal{K}$ is feasible for a course $c \in \mathcal{C}$ and day $d \in \mathcal{D}$ if assigning course c to the pattern on day d does not schedule c in any unavailable periods, i.e., if $a_k^t \leq F_{c,d,t}$, $\forall t \in \mathcal{T}$, and if $L_k \leq L_c$. Bagger et al. (2016) explain pre-processing techniques used to decrease the sizes of the sets $\mathcal{K}_{c,d}$. See Section 2.2.

Let $x_{c,d}^k$ be a binary variable taking value one if course $c \in \mathcal{C}$ is assigned pattern $k \in \mathcal{K}_{c,d}$ for day $d \in \mathcal{D}$. The following constraints ensure that every course selects exactly one pattern for each day, that all lectures are scheduled and that no more than one lecture is scheduled in one room in one period:

$$\sum_{k \in \mathcal{K}_{c,d}} x_{c,d}^k = 1, \quad \forall c \in \mathcal{C}, d \in \mathcal{D} \tag{1}$$

$$\sum_{d \in \mathcal{D}, k \in \mathcal{K}_{c,d}} L_k x_{c,d}^k = L_c, \quad \forall c \in \mathcal{C} \tag{2}$$

$$\sum_{c \in \mathcal{C}, k \in \mathcal{K}_{c,d}} a_t^k x_{c,d}^k \leq R, \quad \forall d \in \mathcal{D}, t \in \mathcal{T} \tag{3}$$

The constraints (1) – (3) ensure that the constraints **A**, **L** and **RO** are not violated. To model constraint **C**, a *pattern conflict graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed. For every course $c \in \mathcal{C}$, day $d \in \mathcal{D}$ and pattern $k \in \mathcal{K}_{c,d}$ there is a node $v^k_{c,d} \in \mathcal{V}$ corresponding to the variable $x^k_{c,d}$. If course $c_1 \in \mathcal{C}$ chooses pattern $k_1 \in \mathcal{K}_{c_1,d_1}$ for day $d_1 \in \mathcal{D}$ and course $c_2 \in \mathcal{C}$ chooses pattern $k_2 \in \mathcal{K}_{c_2,d_2}$ for day $d_2 \in \mathcal{D}$ and this results in a conflict, then there is an edge $e \in \mathcal{E}$ between the two nodes $v^{k_1}_{c_1,d_1}$ and $v^{k_2}_{c_2,d_2}$. Since each course $c \in \mathcal{C}$ must choose exactly one pattern for each day $d \in \mathcal{D}$, we also consider $v^{k_1}_{c,d}$ and $v^{k_2}_{c,d}$ to be conflicting for every pair of patterns $k_1 \in \mathcal{K}_{c,d}$ and $k_2 \in \mathcal{K}_{c,d} \backslash \{k_1\}$. Furthermore, as each course $c \in \mathcal{C}$ must be assigned to L_c periods, we also consider $v^{k_1}_{c,d_1}$ and $v^{k_2}_{c,d_1}$ to be conflicting if $L_{k_1} + L_{k_2} > L_c$, for day $d_1 \in \mathcal{D}$, day $d_2 \in \mathcal{D}\backslash\{d_1\}$, pattern $k_1 \in \mathcal{K}_{c,d_1}$, and pattern $k_2 \in \mathcal{K}_{c,d_2}$. Bagger et al. (2016) identify more conflicts by extending the pre-processing techniques to add more edges to the graph. We have described these extensions in Section 2.2.

Let Θ be a set of cliques that covers all edges, i.e., for each edge there is at least one clique in Θ where both endpoints of the edge are included. For each clique $\theta \in \Theta$ in the graph let $\mathcal{V}_\theta$ be the set of nodes in the clique. Adding the following constraints ensures that the **C** constraints are not violated:

$$\sum_{v^k_{c,d} \in \mathcal{V}_\theta} x^k_{c,d} \leq 1, \quad \forall \theta \in \Theta \tag{4}$$

To generate the clique edge cover, Bagger et al. (2016) ran the heuristic by Kou et al. (1978), that is trying to minimize the cardinality of the set Θ.

Let w_c be an integer variable calculating how much the soft constraint **MWD** is violated. The value of these variables can be calculated by the following constraints:

$$\sum_{d \in \mathcal{D}, k \in \mathcal{K}_{c,d} : L_k \geq 1} x^k_{c,d} + w_c \geq D^{\min}_c, \quad \forall c \in \mathcal{C} \tag{5}$$

To calculate the violation of the soft constraint **IL** the parameter $\overline{a}^k_t$ is defined for each pattern $k \in \mathcal{K}$ and time slot $t \in \mathcal{T}$:

$$\overline{a}^k_t := \begin{cases} 1, & \text{if } a^k_t = 1 \wedge a^k_{t-1} = a^k_{t+1} = 0 \\ -1, & \text{if } a^k_t = 0 \wedge \left(a^k_{t-1} = 1 \vee a^k_{t+1} = 1\right) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The variable $s_{q,d,t}$ is introduced for each curriculum $q \in \mathcal{Q}$, day $d \in \mathcal{D}$ and time slot $t \in \mathcal{T}$. The variable $s_{q,d,t}$ is a binary variable that takes value one if q has an isolated lecture in time slot t for day d:

$$\sum_{c \in \mathcal{C}_q, k \in \mathcal{K}_{c,d}} \overline{a}^k_t x^k_{c,d} \leq s_{q,d,t}, \quad \forall q \in \mathcal{Q}, d \in \mathcal{D}, t \in \mathcal{T} \tag{7}$$

Due to the constraints (4), the left-hand-side of (7) is less than or equal to one.

Let $W^{\mathbf{MWD}}$ and $W^{\mathbf{IL}}$ be the non-negative weights of the soft constraints **MWD** and **IL** respectively. Then the objective function to minimize can be formulated as follows:

$$\sum_{q\in\mathcal{Q},d\in\mathcal{D},t\in\mathcal{T}} W^{\mathbf{IL}} s_{q,d,t} + \sum_{c\in\mathcal{C}} W^{\mathbf{MWD}} w_c \tag{8}$$

Bagger et al. (2016) show that when $D_c^{\min} = L_c$ or $D_c^{\min} = 2$ for a course $c \in \mathcal{C}$, then the following substitutions can be made:

$$w_c = \sum_{\substack{d\in\mathcal{D},\\ k\in\mathcal{K}_{c,d}:L_k\geq 2}} (L_k - 1)\, x_{c,d}^k, \quad \forall c \in \mathcal{C} : D_c^{\min} = L_c \tag{9}$$

$$w_c = \sum_{\substack{d\in\mathcal{D},\\ k\in\mathcal{K}_{c,d}:L_k=L_c}} x_{c,d}^k, \quad \forall c \in \mathcal{C} : D_c^{\min} = 2 \tag{10}$$

We replace the variable w_c in the objective function with the right-hand side of either (9) or (10) and remove the associated constraints (5) from the model.

Consider a curriculum $q \in \mathcal{Q}$, a day $d \in \mathcal{D}$ and time slot $t \in \mathcal{T}$. Let the set of courses $\mathcal{C}_{q,d,t}$ be defined as follows:

$$\mathcal{C}_{q,d,t} := \left\{ c \in \mathcal{C}_q \;\middle|\; \sum_{t'\in\{t-1,t,t+1\}} F_{c,d,t'} \geq 1 \right\} \tag{11}$$

Bagger et al. (2016) make the following substitution when the number of courses in $\mathcal{C}_{q,d,t}$ equals one:

$$s_{q,d,t} = \sum_{c\in\mathcal{C}_q,k\in\mathcal{K}_{c,d}:\overline{a}_t^k=1} x_{c,d}^k, \quad \forall q \in \mathcal{Q}, d \in \mathcal{D}, \in \mathcal{T} : |\mathcal{C}_{q,d,t}| = 1 \tag{12}$$

We replace the variable $s_{q,d,t}$ with the right-hand side of (12) in the objective function and remove the associated constraints (7) from the model.

Bagger et al. (2016) also described valid inequalities that are added to the model. We have provided an overview of these inequalities in Section 2.1.

2.1. Valid inequalities

In this section, we have provided an overview of the valid inequalities described by Bagger et al. (2016). We have defined a *working day* of a course to be a day where at least one lecture is scheduled. We can calculate the minimum and maximum number of working days that course

$c \in \mathcal{C}$ can have, which leads to the following valid inequalities:

$$\min_{\mathcal{D}' \subseteq \mathcal{D}} \left\{ |\mathcal{D}'| : \sum_{\substack{d \in \mathcal{D}', \\ t \in \mathcal{T}}} F_{c,d,t} \geq L_c \right\} \leq \sum_{\substack{d \in \mathcal{D}, \\ k \in \mathcal{K}_{c,d}: \\ L_k \geq 1}} x^k_{c,d} \leq \min \left\{ L_c, \left| \left\{ d \in \mathcal{D} : \sum_{t \in \mathcal{T}} F_{c,d,t} \geq 1 \right\} \right| \right\} \tag{13}$$

For the next inequalities, we have introduced the *course cliques*. Construct a graph, that contains a node for each course. For two *conflicting* courses, connect the corresponding nodes with an edge. The maximal cliques are enumerated in the graph, using the algorithm by Bron and Kerbosch (1973). We denote this set of *course cliques* Γ. For each clique $\gamma \in \Gamma$, we have denoted the set of courses $\mathcal{C}_\gamma$. Let L_γ be the total number of lectures to be scheduled for all the courses $\mathcal{C}_\gamma$: $L_\gamma := \sum_{c \in \mathcal{C}_\gamma} L_c$. For course clique $\gamma \in \Gamma$, and for $i \in \left\{2, 3, \dots, \left\lfloor \frac{L_\gamma}{2} \right\rfloor + 1\right\}$, the following inequalities are valid:

$$\sum_{\substack{c \in \mathcal{C}_\gamma, d \in \mathcal{D}, \\ k \in \mathcal{K}_{c,d}: L_k \geq i}} x^k_{c,d} \leq \left\lfloor \frac{L_\gamma}{i} \right\rfloor \tag{14}$$

For each course $c \in \mathcal{C}$, day $d \in \mathcal{D}$, and pattern $k \in \mathcal{K}_{c,d}$, let $\overline{L}^k_{c,d} := \max_{k' \in \mathcal{K}_{c,d}} \{L_{k'}\} - L_k$. For each course clique $\gamma \in \Gamma$, let $\overline{L}_\gamma := \sum_{c \in \mathcal{C}_\gamma, d \in \mathcal{D}} \max_{k \in \mathcal{K}_{c,d}} \{L_k\} - \sum_{c \in \mathcal{C}_\gamma} L_c$. For $i \in \left\{2, 3, \dots, \left\lfloor \frac{\overline{L}_\gamma}{2} \right\rfloor\right\}$, the following inequalities are valid:

$$\sum_{\substack{c \in \mathcal{C}_\gamma, d \in \mathcal{D}, \\ k \in \mathcal{K}_{c,d}: \overline{L}^k_{c,d} \geq i}} x^k_{c,d} \leq \left\lfloor \frac{\overline{L}_\gamma}{i} \right\rfloor \tag{15}$$

The inequalities (14) and (15) are valid for any set of courses. However, as there are $2^{|\mathcal{C}|}$ potential sets of courses, we only add the inequalities for the maximal course cliques, as the number of maximal course cliques is at most $3^{|\mathcal{C}|/3}$ (Moon and Moser, 1965).

The last valid inequalities by Bagger et al. (2016) are to consider the pattern variables that are *within the support* of the constraints (7). We say that a variable is *within the support* of a constraint, if the coefficient of the variable is non-zero in the constraint. The variables are split into two sets:

$$\mathcal{V}^+_{q,d,t} := \left\{ v^k_{c,d} \in \mathcal{V} : c \in \mathcal{C}_q, k \in \mathcal{K}_{c,d}, \overline{a}^k_t = 1 \right\} \tag{16}$$

$$\mathcal{V}^-_{q,d,t} := \left\{ v^k_{c,d} \in \mathcal{V} : c \in \mathcal{C}_q, k \in \mathcal{K}_{c,d}, \overline{a}^k_t = -1 \right\} \tag{17}$$

Let $\mathcal{H}_{q,d,t} \subseteq \mathcal{V}$ be a clique, where every node in $\mathcal{H}_{q,d,t}$ is a neighbour of every node in $\mathcal{V}^-_{q,d,t}$, i.e., $\left(v^k_{c,d}, v^{k'}_{c',d'}\right) \in \mathcal{E}, \forall v^k_{c,d} \in \mathcal{V}^-_{q,d,t}, v^{k'}_{c',d'} \in \mathcal{H}_{q,d,t}$ and $\left(v^k_{c,d}, v^{k'}_{c',d'}\right) \in \mathcal{E}, \forall v^k_{c,d}, v^{k'}_{c',d'} \in \mathcal{H}_{q,d,t} : v^k_{c,d} \neq v^{k'}_{c',d'}$.

Then the following is a valid inequality:

$$\sum_{v^k_{c,d} \in \mathcal{V}^+_{q,d,t}} x^k_{c,d} + \sum_{v^k_{c,d'} \in \mathcal{H}_{q,d,t}} x^k_{c,d'} - s_{q,d,t} \leq 1 \tag{18}$$

For each day $d \in \mathcal{D}$, we let $\mathcal{V}_d \subseteq \mathcal{V}$ denote the nodes that correspond to day d. Bagger et al. (2016) consider cliques $\mathcal{H}_{q,d,t}$, that are a subset of $\mathcal{V}$, which means that there may be nodes $v^{k'}_{c',d'} \in \mathcal{H}_{q,d,t}$ where $v^{k'}_{c',d'} \notin \mathcal{V}_d$. We restrict the cliques $\mathcal{H}_{q,d,t}$ to be a subset of $\mathcal{V}_d$ in our implementation. The reason we do this is to make it possible to include these inequalities in our pricing problems, as we consider a pricing problem for each day $d \in \mathcal{D}$.

2.2. Pre-processing and Conflict Detection

In this section, we have provided an overview of the pre-processing techniques, and the detection of conflicts in the pattern graph by Bagger et al. (2016). The pre-processing techniques remove nodes from the pattern graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Since each node $v^k_{c,d} \in \mathcal{V}$ corresponds to a variable $x^k_{c,d}$ in the pattern formulation, we remove the variable also if the node is removed. All these variables are put into the pricing problems in our Dantzig-Wolfe decomposition, thus, removing variables in the pattern formulation corresponds to removing variables from the pricing problems. The detection of conflicts is an extension of the pre-processing techniques, so we go through the conflict detection as we go through the pre-processing.

Consider a course $c \in \mathcal{C}$, day $d \in \mathcal{D}$, and pattern $k \in \mathcal{K}_{c,d}$. The course c can only be assigned to pattern k on day d if the following holds:

$$\sum_{d' \in \mathcal{D}\setminus\{d\}} \min_{k' \in \mathcal{K}_{c,d'}} \{L_{k'}\} \leq L_c - L_k \leq \sum_{d' \in \mathcal{D}\setminus\{d\}} \max_{k' \in \mathcal{K}_{c,d'}} \{L_{k'}\} \tag{19}$$

If (19) is not fulfilled then pattern k is removed from the set $\mathcal{K}_{c,d}$, along with the corresponding node $v^k_{c,d} \in \mathcal{V}$ and variable $x^k_{c,d}$. Note that when we remove patterns from the sets $\mathcal{K}_{c,d}$, then the sums in condition (19) are changed for other days and patterns. So, whenever a pattern is removed from a set $\mathcal{K}_{c,d}$, condition (19) is checked again for the other patterns. The condition (19) can be extended to identify a conflict, by considering a course $c \in \mathcal{C}$, day $d_1 \in \mathcal{D}$, day $d_2 \in \mathcal{D}\setminus\{d_1\}$, pattern $k_1 \in \mathcal{K}_{c,d_1}$, and pattern $k_2 \in \mathcal{K}_{c,d_2}$. The course c can only be assigned to pattern k_1 on day d_1 and to pattern k_2 on day d_2, at the same time, if the following holds:

$$\sum_{d \in \mathcal{D}\setminus\{d_1,d_2\}} \min_{k \in \mathcal{K}_{c,d}} \{L_k\} \leq L_c - L_{k_1} - L_{k_2} \leq \sum_{d \in \mathcal{D}\setminus\{d_1,d_2\}} \max_{k \in \mathcal{K}_{c,d}} \{L_k\} \tag{20}$$

If (20) does not hold, then we add an edge to $\mathcal{E}$ between the nodes, $v^{k_1}_{c,d_1}$ and $v^{k_2}_{c,d_1}$.

Consider the periods that the lectures are scheduled for when assigning the courses to the patterns. We introduce the notation of all feasible periods $\mathcal{P}_c$ for course $c \in \mathcal{C}$, and all the periods

$\mathcal{P}^k_d$ that have lectures scheduled if a course is assigned to the pattern $k \in \mathcal{K}$, on day $d \in \mathcal{D}$:

$$\mathcal{P}_c := \{(d,t) \in \mathcal{D} \times \mathcal{T} \,:\, F_{c,d,t} = 1\} \tag{21}$$

$$\mathcal{P}^k_d := \{d\} \times \left\{t \in \mathcal{T} \,:\, a^k_t = 1\right\} \tag{22}$$

A course $c \in \mathcal{C}$ cannot be assigned to pattern $k \in \mathcal{K}_{c,d}$ on day $d \in \mathcal{D}$, if there exists a course $c' \in \mathcal{C}\backslash\{c\}$, that is conflicting with c, where the following is not satisfied:

$$L_{c'} \leq \left|\mathcal{P}_{c'} \backslash \mathcal{P}^k_d\right| \tag{23}$$

Condition (23) implies that the number of feasible periods for c', that is not overlapping with the periods in the selected pattern, must be able to accommodate all the lectures of c'.

Consider a clique $\gamma \in \Gamma$, and a course $c \in \mathcal{C}_\gamma$. We cannot assign c to a pattern $k \in \mathcal{K}_{c,d}$, on day $d \in \mathcal{D}$, if the periods that are not occupied by c, are not enough to schedule all the lectures of the remaining courses in the clique, i.e., if the following is not satisfied:

$$\sum_{c' \in \mathcal{C}_\gamma \backslash \{c\}} L_{c'} \leq \left| \bigcup_{c' \in \mathcal{C}_\gamma \backslash \{c\}} \mathcal{P}_{c'} \backslash \mathcal{P}^k_d \right| \tag{24}$$

The *period coverage* conditions (23) and (24) can be extended to identify pattern conflicts. Consider a course $c_1 \in \mathcal{C}$, and a course $c_2 \in \mathcal{C}\backslash\{c_1\}$, which is conflicting with c_1. Assume that c_1 is assigned to pattern $k_1 \in \mathcal{K}_{c_1,d_1}$ on day $d_1 \in \mathcal{D}$, and that c_2 is assigned to pattern $k_2 \in \mathcal{K}_{c_2,d_2}$ on day $d_2 \in \mathcal{K}_{c_2,d_2}$. The remaining lectures of c_2 must be scheduled in periods that do not belong to d_2 as only one pattern can be assigned per day. Thus, the pair of assignments is infeasible if the remaining lectures of c_2 cannot be covered by the periods, not belonging to d_2, that are not occupied by c_1, i.e., an edge is added between $v^{k_1}_{c_1,d_2}$ and $v^{k_2}_{c_2,d_2}$ if the following does not hold:

$$L_{c_2} - L_{k_2} \leq \left|\mathcal{P}_{c_2} \backslash \left(\mathcal{P}_{d_2} \cap \mathcal{P}^{k_1}_{d_1}\right)\right| \tag{25}$$

Consider two courses again, $c_1 \in \mathcal{C}$ and $c_2 \in \mathcal{C}$, but this time the courses do not need to be conflicting, nor do they need to be distinct. Let $c_3 \in \mathcal{C}\backslash\{c_1, c_2\}$ be a course that is conflicting with both c_1 and c_2. Assume that c_1 is assigned to $k_1 \in \mathcal{K}_{c_1,d_1}$ on day $d_1 \in \mathcal{D}$, and that c_2 is assigned to $k_2 \in \mathcal{K}_{c_2,d_2}$ on day $d_2 \in \mathcal{D}$. If $c_1 = c_2$, then we only consider $d_1 \neq d_2$. As c_3 is conflicting with both c_1 and c_2, then all the lectures of c_3 must be scheduled in periods that are not occupied by c_1 or c_2. If the following is not valid, then the pair of assignments are conflicting, and we add and edge between $v^{k_1}_{c_1,d_2}$ and $v^{k_2}_{c_2,d_2}$:

$$L_{c_3} \leq \left|\mathcal{P}_{c_3} \backslash \left(\mathcal{P}^{k_1}_{d_1} \cup \mathcal{P}^{k_2}_{d_2}\right)\right| \tag{26}$$

The condition (26) can be extended by considering another course $c_4 \in \mathcal{C}\backslash\{c_1, c_2, c_3\}$, which is conflicting with all three courses, c_1, c_2, and c_3:

$$L_{c_3} + L_{c_4} \leq \left| (\mathcal{P}_{c_3} \cup \mathcal{P}_{c_4}) \setminus \left(\mathcal{P}^{k_1}_{d_1} \cup \mathcal{P}^{k_2}_{d_2} \right) \right| \tag{27}$$

The condition (27) can be extended even further, by using the course cliques Γ. Consider a clique $\gamma \in \Gamma$, where the courses c_1 and c_2, may or may not be part of $\mathcal{C}_\gamma$, but where every node in $\mathcal{C}_\gamma \backslash \{c_1, c_2\}$ is conflicting with both c_1 and c_2. We add an edge between $v^{k_1}_{c_1,d_2}$ and $v^{k_2}_{c_2,d_2}$, if the following does not hold:

$$\sum_{c \in \mathcal{C}_\gamma \backslash \{c_1,c_2\}} L_c \leq \left| \bigcup_{c \in \mathcal{C}_\gamma \backslash \{c_1,c_2\}} \mathcal{P}_c \backslash \left(\mathcal{P}^{k_1}_{d_1} \cup \mathcal{P}^{k_2}_{d_2} \right) \right| \tag{28}$$

Next, Bagger et al. (2016) used a directed graph and solved a series of *Maximum Flow Problems* (MFP), as a pre-processing technique and for conflict detection. The graph contains a source node ($\mathfrak{s}$), a sink node ($\mathfrak{t}$), and a *dummy* source node ($\mathfrak{s}'$). For each period $p \in \mathcal{P}$, there is a node (p), and for each course $c \in \mathcal{C}$, there is a node (c). From the source node ($\mathfrak{s}$), there is an outgoing arc to the *dummy* source node ($\mathfrak{s}'$), and to the node (p) for each period $p \in \mathcal{P}$. For each courses $c \in \mathcal{C}$, there is an outgoing arc from (c) to the sink ($\mathfrak{t}$), and an ingoing arc from (p) for each period $p \in \mathcal{P}$. The graph is illustrated in Figure 1.

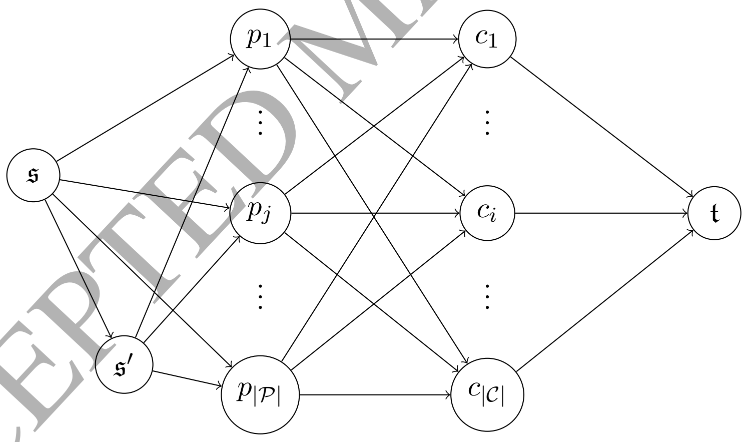

Figure 1: Illustration of the maximum flow graph, that is used for removing patterns, and detecting cliques.

Consider course clique $\gamma \in \Gamma$. For each course $c \in \mathcal{C}$, set the capacity of the arc $(c, \mathfrak{t})$ to L_c if $c \in \mathcal{C}_\gamma$; otherwise, set it to zero. Consider now course $c \in \mathcal{C}_\gamma$, day $d \in \mathcal{D}$, and pattern $k \in \mathcal{K}_{c,d}$. Set the capacity of the arc $(\mathfrak{s}, \mathfrak{s}')$ to $L_\gamma - L_k$. For each period $p \in \mathcal{P}$, if $p \in \mathcal{P}^k_d$ then set the capacity of the arc $(\mathfrak{s}, p)$ to one and the capacity of $(\mathfrak{s}', p)$ to zero, and if $p \notin \mathcal{P}^k_d$ then set the capacity of $(\mathfrak{s}, p)$ to zero and the capacity of $(\mathfrak{s}', p)$ to one. For each period $p \in \mathcal{P}$ and course $c' \in \mathcal{C}$, set the capacity of the arc (p, c') to one if $c' = c \wedge p \in \mathcal{P}_{c'} \backslash \left(\mathcal{P}_d \backslash \mathcal{P}^k_d \right)$ or if $c' \neq c \wedge p \in \mathcal{P}_{c'} \backslash \mathcal{P}^k_d$; otherwise, set it to zero. Now, solve the MFP, and if the value of the flow is less than L_γ, then assigning c to $k \in \mathcal{K}_{c,d}$

on day $d \in \mathcal{D}$ is infeasible, and we remove the node $v_{c,d}^k$ from $\mathcal{V}$ and the corresponding variable.

The graph can also be used for identifying a conflict. Consider course clique $\gamma \in \Gamma$. For each course $c \in \mathcal{C}$, set the capacity of the arc $(c, \mathfrak{t})$ to L_c if $c \in \mathcal{C}_\gamma$; otherwise, set it to zero. Consider now course $c_1 \in \mathcal{C}_\gamma$, day $d_1 \in \mathcal{D}$, pattern $k_1 \in \mathcal{K}_{c_1,d_1}$, course $c_2 \in \mathcal{C}_\gamma$, day $d_2 \in \mathcal{D}$, and pattern $k_2 \in \mathcal{K}_{c_2,d_2}$. If $c_1 = c_2$ then we only consider $d_1 \neq d_2$. Set the capacity of the arc $(\mathfrak{s}, \mathfrak{s}')$ to $L_\gamma - L_{k_1} - L_{k_2}$. For each period $p \in \mathcal{P}$, if $p \in \mathcal{P}_{d_1}^{k_1} \cup \mathcal{P}_{d_2}^{k_2}$ then set the capacity of the arc $(\mathfrak{s}, p)$ to one and the capacity of $(\mathfrak{s}', p)$ to zero, and if $p \notin \mathcal{P}_{d_1}^{k_1} \cup \mathcal{P}_{d_2}^{k_2}$ then set the capacity of $(\mathfrak{s}, p)$ to zero and the capacity of $(\mathfrak{s}', p)$ to one. For each period $p \in \mathcal{P}$ and course $c' \in \mathcal{C}$, set the capacity of the arc (p, c') to one if $p \in \mathcal{P}_{c'} \setminus (\mathcal{P}_{d_1} \cup \mathcal{P}_{d_2})$ or if $c' = c_1 \wedge p \in \mathcal{P}_{c'} \cap \mathcal{P}_{d_1}^{k_1}$ or if $c' = c_2 \wedge p \in \mathcal{P}_{c'} \cap \mathcal{P}_{d_2}^{k_2}$ or if $c' \notin \{c_1, c_2\} \wedge p \in \mathcal{P}_{c'} \setminus \left(\mathcal{P}_{d_1}^{k_1} \cup \mathcal{P}_{d_2}^{k_2}\right)$; otherwise, set it to zero. Solve the MFP on the graph, and if the value of the flow is less than L_γ, then there is a conflict and we add an edge to $\mathcal{E}$ between $v_{c_1,d_1}^{k_1}$ and $v_{c_2,d_2}^{k_1}$.

We applied all the pre-processing techniques that we have described in this section to the model first, and then we detected the conflicts afterwards, as conflict detection is affected by the pre-processed model. In this paper, we have applied an additional pre-processing technique that has not been described by Bagger et al. (2016). We considered the model after the pre-processing and conflict detection, including all the valid inequalities presented in Section 2.1, with the exception of the w and s variables and their associated constraints, i.e., we only consider the feasibility part of the model. We then iterated through each variable $x_{c,d}^k$ and set the lower bound to one. Then we solved the LP relaxation; if the model was infeasible, we removed the variable. Otherwise, we reset the lower bound of $x_{c,d}^k$ to zero.

3. Dantzig-Wolfe Decomposition

Martin (1999, chapter 11) states that the Dantzig-Wolfe decomposition should be chosen, so that the pricing problem contains a *vast majority* of the constraints, and so that the pricing problem has a *special* structure. The model we have reformulated is the model from Section 2 with the additional pre-processing, based on solving a series of LP-relaxations, as we have mentioned in the end of Section 2.2. However, we have not included all the valid inequalities described by Bagger et al. (2016). We have only included the ones where all the variables that are within the support can be associated with a single day. We have reformulated the model, so that we have a pricing problem for each day $d \in \mathcal{D}$. We want to keep the master problem simple. Hence, we only keep the constraints that ensure integer feasibility, or where the variables that are within the support correspond to the same day.

For each day $d \in \mathcal{D}$, let $\mathcal{H}_d$ be the set of columns associated with d. In our reformulation, a column $h \in \mathcal{H}_d$ represents a full pattern assignment for day d. Let λ_b^h be a binary variable that takes value one if column $h \in \mathcal{H}_d$ is selected for day $d \in \mathcal{D}$ and let α_d^h be the associated cost. For each day $d \in \mathcal{D}$ and column $h \in \mathcal{H}_d$ the parameter $\overline{x}_{c,d}^{k,h}$ is one if the column assigns course $c \in \mathcal{C}$

to pattern $k \in \mathcal{K}_{c,d}$. Furthermore, we have defined the set $\mathcal{C}_{\min} \subseteq \mathcal{C}$ as the set of courses where substitutions (9) and (10) do not apply. Lastly, we have defined the set $\Theta_{\geq 2} \subseteq \Theta$, which is the set of the pattern cliques in Θ that contains variables from at least two different days. We have formulated the LP-relaxation of our master problem (MP) as follows:

$$\min \sum_{c \in \mathcal{C}_{\min}} W^{\mathbf{MWD}} w_c + \sum_{d \in \mathcal{D}, h \in \mathcal{H}_d} \alpha_d^h \lambda_d^h \tag{29}$$

$$\text{s.t.} \sum_{\substack{d \in \mathcal{D}, h \in \mathcal{H}_d, \\ k \in \mathcal{K}_{c,d}}} L_k \overline{x}_{c,d}^{k,h} \lambda_d^h = L_c, \quad \forall c \in \mathcal{C} \tag{30}$$

$$\sum_{\substack{d \in \mathcal{D}, h \in \mathcal{H}_d, \\ k \in \mathcal{K}_{c,d} : L_k \geq 1}} \overline{x}_{c,d}^{k,h} \lambda_d^h + w_c \geq D_c^{\min}, \quad \forall c \in \mathcal{C}_{\min} \tag{31}$$

$$\sum_{h \in \mathcal{H}_d, c \in \mathcal{C}, k \in \mathcal{K}_{c,d}} a_t^k \overline{x}_{c,d}^{k,h} \lambda_d^h \leq R, \quad \forall d \in \mathcal{D}, t \in \mathcal{T} \tag{32}$$

$$\sum_{v_{c,d}^k \in \mathcal{V}_\theta, h \in \mathcal{H}_d} \overline{x}_{c,d}^{k,h} \lambda_d^h \leq 1, \quad \forall \theta \in \Theta_{\geq 2} \tag{33}$$

$$\sum_{h \in \mathcal{H}_d} \lambda_d^h \leq 1, \quad \forall d \in \mathcal{D} \tag{34}$$

$$w_c \geq 0, \quad \forall c \in \mathcal{C}_{\min} \tag{35}$$

$$\lambda_d^h \geq 0, \quad \forall d \in \mathcal{D}, h \in \mathcal{H}_d \tag{36}$$

Constraints (30) ensure that all lectures are assigned for each course. Constraints (31) calculate the violations of the **MWD** constraints, for the courses where the substitutions (9) and (10) do not apply. Constraints (32) ensure that no more lectures are scheduled in each period than the number of rooms available. Constraints (33) ensure that all the conflicts that spread over multiple pricing problems are not violated. Constraints (34) ensure that at most one column is chosen for each day. We could make the constraints as equalities. The reason that we have kept the constraints as inequalities, is that if we consider a column for a day where no lecture is scheduled, then this column has a cost of zero and it will not be within the support of any other constraints. We have solved the model with a Column Generation algorithm where we have only considered a restricted set of columns in the master problem, i.e., the *restricted master problem* (RMP). The first columns we added to RMP were found by solving the model described in Section 2 excluding the variables w and s and all the associated constraints to these variables, i.e., we have considered only the feasibility part. The solution of the model was then the first set of columns.

Let the dual variables of the constraints (30), (31), (32), (33) and (34) be denoted β_c, ϕ_c, $\mu_{d,t}$, ζ_θ and π_d^0 respectively. Consider an optimal dual solution $\left(\overline{\beta}, \overline{\phi}, \overline{\mu}, \overline{\zeta}, \overline{\pi}^0\right)$ of the restricted master problem in some iteration of the column generation algorithm. We define the parameter $\overline{\phi}_c^k$ to be equal to $\overline{\phi}_c$ if $c \in \mathcal{C}_{\min}$ and $L_k \geq 1$; otherwise, we set it to zero. For each course $c \in \mathcal{C}$, day $d \in \mathcal{D}$ and pattern $k \in \mathcal{K}_{c,d}$ we define $\overline{\pi}^k_{c,d}$:

$$\overline{\pi}^k_{c,d} := L_k \overline{\beta}_c + \overline{\phi}^k_c + \sum_{t \in \mathcal{T}} a^k_t \overline{\mu}_{d,t} + \sum_{\theta \in \Theta_{\geq 2} : v^k_{c,d} \in \mathcal{V}_\theta} \overline{\zeta}_\theta \tag{37}$$

For a day $d \in \mathcal{D}$ we describe the associated pricing problem in the following manner. Let x^k_c be a binary variable taking value one if course $c \in \mathcal{C}$ is assigned to pattern $k \in \mathcal{K}_{c,d}$ and zero otherwise. Let $s_{q,t}$ be a binary variable taking value one if curriculum $q \in \mathcal{Q}$ has an isolated lecture scheduled in time slot $t \in \mathcal{T}$. For each day $d \in \mathcal{D}$, let Θ_d be a set of cliques found in the same way as the cliques Θ in Section 2. However, here we restricted the cliques to be in the sub-graph of the pattern clique graph induced by considering only the nodes belonging to day d. Note that, as we applied the substitutions mentioned in Section 2, the variable $s_{q,t}$ is only defined for $|\mathcal{C}_{q,d,t}| > 1$. Lastly, let $\alpha^k_{c,d}$ be the cost of pattern $k \in \mathcal{K}_{c,d}$ for course $c \in \mathcal{C}$ and day $d \in \mathcal{D}$ after the substitutions (9), (10) and (12). We can then formulate the pricing problem for day $d \in \mathcal{D}$:

$$\min \sum_{\substack{q \in \mathcal{Q}, t \in \mathcal{T}: \\ |\mathcal{C}_{q,d,t}| > 1}} W^{\mathbf{IL}} s_{q,t} + \sum_{c \in \mathcal{C}, k \in \mathcal{K}_{c,d}} \left(\alpha^k_{c,d} - \overline{\pi}^k_{c,d} \right) x^k_c - \overline{\pi}^0_d \tag{38}$$

$$\text{s.t.} \sum_{k \in \mathcal{K}_{c,d}} x^k_c = 1, \quad \forall c \in \mathcal{C} \tag{39}$$

$$\sum_{v^k_{c,d} \in \mathcal{V}_\theta} x^k_c \leq 1, \quad \forall \theta \in \Theta_d \tag{40}$$

$$\sum_{c \in \mathcal{C}_q, k \in \mathcal{K}} \overline{a}^k_t x^k_c \leq s_{q,t}, \quad \forall q \in \mathcal{Q}, t \in \mathcal{T} : |\mathcal{C}_{q,d,t}| > 1 \tag{41}$$

$$x^k_c \in \mathbb{B}, \quad \forall c \in \mathcal{C}, k \in \mathcal{K}_{c,d} \tag{42}$$

$$s_{q,t} \in \mathbb{B}, \quad \forall q \in \mathcal{Q}, t \in \mathcal{T} \tag{43}$$

Consider a solution $(\overline{x}, \overline{s})$ of the pricing problem (38) – (43) for day $d \in \mathcal{D}$. If the solution has a negative objective value then we can add it to the master problem as a new column $h \in \mathcal{H}_d$ by setting $\overline{x}^{k,h}_{c,d} = \overline{x}^k_c$ for every course $c \in \mathcal{C}$ and pattern $k \in \mathcal{K}_{c,d}$.

We also included the valid inequalities (18) from Bagger et al. (2016) that can be associated with the specific day. The constraints (32) could be included in the pricing problem instead of the master problem as each of them can be associated with a specific day. However, keeping these constraints in the MP ensures that the pricing problem has a *special* structure. We have described how we can exploit this *special* structure in Section 4.

4. Pre-processing, Inequalities and Solution Method for the Pricing Problem

The pricing problems described in Section 3 can be difficult to solve for a generic MIP solver. In this section, we describe the techniques we used to speed up the solution process. We describe how we removed some of the variables of the pricing problem in an iteration of the column generation algorithm in Section 4.1. Next, we describe how we used the pre-solving technique to derive inequalities in Section 4.2. Lastly, we describe how we used Local Branching as described by Fischetti and Lodi (2003) to solve the pricing problem in Section 4.3.

4.1. Pre-processing

In this section, we describe a pre-processing technique to eliminate some of the variables from the pricing problems. The technique is based on the objective function coefficients of the x variables. Since the coefficients change in each iteration of the column generation algorithm, the variables we removed in one iteration must be reinserted for the next iteration.

Consider some course $c \in \mathcal{C}$ in the pricing problem for the day $d \in \mathcal{D}$ in any iteration of the column generation algorithm. Consider the patterns $k_1, k_2 \in \mathcal{K}_{c,d}$ where $k_1 \neq k_2$. In the pre-processing technique, we consider a feasible solution where c is assigned to k_2. Then we check if the solution is still feasible if we reassign c to pattern k_1. If the solution is still feasible, then we check whether the objective value increased after the reassignment. Here, we exploit that the room occupancy constraints are part of the master problem, so we only have to consider the constraints (39) and (40) in the pricing problem for feasibility. When c is assigned to k_2, then the value of the variable $x_c^{k_2}$ is one. Assigning c to k_1 instead of k_2 corresponds to setting the value of $x_c^{k_2}$ to zero, and the value of $x_c^{k_1}$ to one. Both $x_c^{k_1}$ and $x_c^{k_2}$ are in the constraint (39) associated with c, which implies that since the solution was feasible before, then this constraint cannot be violated in the new solution. Let $\mathcal{G}_d = (\mathcal{V}_d, \mathcal{E}_d) \subseteq \mathcal{G}$ be the sub-graph, where $\mathcal{V}_d \subseteq \mathcal{V}$ is the set of nodes associated with day d and $\mathcal{E}_d \subseteq \mathcal{E}$ is the set of edges where both end points are in $\mathcal{V}_d$. Every edge in $\mathcal{E}_d$ is contained in at least one of the constraints (40). To check if the constraints (40) are fulfilled, we need to consider the neighbourhoods in $\mathcal{G}_d$ of the nodes $v_{c,d}^{k_1}, v_{c,d}^{k_2} \in \mathcal{V}_d$. Let the neighbourhood of $v_{c,d}^{k}$ in $\mathcal{V}_d$, excluding every node that corresponds to c, be denoted by $\mathcal{N}_{c,d}^{k} \subseteq \mathcal{V}_d$ for every $k \in \mathcal{K}_{c,d}$. Note that $v_{c,d}^{k_1}$ and $v_{c,d}^{k_2}$ must be connected by an edge since c cannot be assigned to more than one pattern. Assume that every node in $\mathcal{N}_{c,d}^{k_1}$ is also a neighbour of $v_{c,d}^{k_2}$, so, $\mathcal{N}_{c,d}^{k_1} \subseteq \mathcal{N}_{c,d}^{k_2}$. This case is illustrated in Figure 2.

For any solution where c is assigned to k_2, the values of all the variables that correspond to the neighbours of $v_{c,d}^{k_2}$ must be zero. Since $\mathcal{N}_{c,d}^{k_1}$ is contained in the neighbourhood of $v_{c,d}^{k_2}$, then all the variables in $\mathcal{N}_{c,d}^{k_1}$ must also be zero. When we reassign c from k_2 to k_1 it means that we are changing the value for $x_c^{k_2}$ from one to zero, and the value for $x_c^{k_1}$ from zero to one. As all of the variables in $\mathcal{N}_{c,d}^{k_1}$ are zero, and we have now changed $x_c^{k_2}$ to zero, then in all the constraints (40) where $x_c^{k_1}$ is within the support, all other variables must be zero, which means that the solution

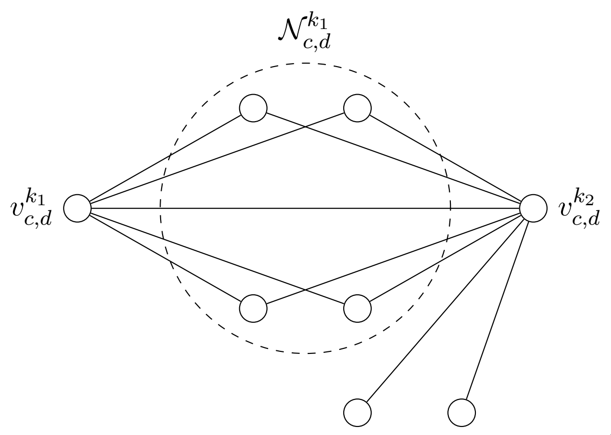

Figure 2: Illustration of the nodes in the pattern conflict graph, $\mathcal{G}_d$, corresponding to the patterns, k_1 and k_2 for course c, and the neighbourhood $\mathcal{N}^{k_1}_{c,d} \subseteq \mathcal{V}_d$.

must be feasible. We did not account for the nodes $v^k_{c,d}$ for $k \in \mathcal{K}_{c,d} \setminus \{k_1, k_2\}$, but as c is assigned to exactly one pattern, then all these variables must be zero as well. Now, we need to check if we can guarantee that the objective value does not increase.

Consider the objective function (38), which consists of a sum of the s variables followed by a sum of the x variables and a constant. The change of the sum of the x variables is the difference between the coefficients of $x^{k_1}_c$ and $x^{k_2}_c$. The change in the sum of the s variables is unknown, as it depends on the assigned patterns of the other courses. However, if we assume that we know an upper bound $\delta^{k_1,k_2}_{c,d}$ on how much the value can increase, then the total objective value cannot increase under the following condition:

$$\left(\alpha^{k_2}_{c,d} - \pi^{k_2}_{c,d}\right) - \left(\alpha^{k_1}_{c,d} - \pi^{k_1}_{c,d}\right) \geq \delta^{k_1,k_2}_{c,d} \tag{44}$$

The upper bound $\delta^{k_1,k_2}_{c,d}$ is determined by the maximum number of isolated lectures that are introduced when we make the reassignment. Consider the difference in the time slots that are contained in the two patterns. As an example, let $|\mathcal{T}| = 6$ and let pattern k_2 contain lectures in time slots t_2 and t_3, and let pattern k_1 contain lectures in time slots t_2, t_3, and t_5. This example is illustrated in a matrix in Figure 3. Each row in the matrix corresponds to a pattern and the columns correspond to the time slots. The symbol "$\times$" denotes that the pattern contains a lecture in the corresponding time slot.

	t_1	t_2	t_3	t_4	t_5	t_6
k_2		×	×			
k_1		×	×		×	

Figure 3: Illustration of the example of changing out pattern k_2 with k_1. Here a lecture is added.

As we are reassigning c from k_2 to k_1, then this means that c is assigned an extra lecture in time slot t_5. Since the pattern k_1 does not contain a lecture in either time slot t_4 nor in time slot t_6, then the lecture in t_5 is potentially a new isolated lecture for every curriculum that c belongs to. So for every lecture that is added in the reassignment, if k_1 does not have a lecture in an adjacent time slot, then there is a potentially isolated lecture.

The next step is to consider the case where lectures are removed. Consider an example where k_2 contains lectures in time slots t_3 and t_5, and k_1 contains a lecture in t_3. This example is illustrated in Figure 4.

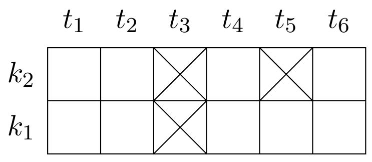

Figure 4: Illustration of the example of changing out pattern k_2 with k_1. Here a lecture is removed.

When we reassign c from k_2 to k_1, time slot t_5 is removed and the time slots that are adjacent, t_4 and t_6 can become potentially isolated lectures. Since k_1 contains a lecture in time slot t_3, which is adjacent to t_4, then only t_6 is counted as a potential isolated lecture. So, when a lecture gets removed from the reassignment, we consider the adjacent time slots as potentially isolated lectures, unless there are lectures adjacent to those time slots in the new pattern.

After we have found all the potentially isolated lectures, we iterate through every curriculum that c belongs to, i.e., $\mathcal{Q}_c$. For each $q \in \mathcal{Q}_c$ we consider all the potentially isolated lectures previously found. We then remove every time slot t where $a_t^k = 0$ for every $k \in \mathcal{K}_{c',d}$ and every $c' \in \mathcal{C}_q$, i.e., if no course can be scheduled in t, this means that there cannot be an isolated lecture. As an example let the potential isolated lectures for q be in time slots t_2, t_3, and t_5 after the reassignment. This example is illustrated in Figure 5.

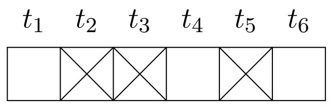

Figure 5: Illustration of the potential isolated lectures.

Let a sequence of these potentially isolated lectures be a set of consecutive time slots where there is a lecture in each of them but no lecture after the last and before the first lecture. In the example in Figure 5 there are two sequences; the first sequence consists of time slots t_2 and t_3, and the second sequence consists of time slot t_5. It is not possible to have two isolated lectures in adjacent time slots, so the maximum number of isolated lectures in each sequence is the number of time slots in the sequence divided by two and rounded up to the nearest integer.

We calculate the value of $\delta_{c,d}^{k_1,k_2}$ iteratively, and we initially set it at zero. Then we iterate over all the curricula $\mathcal{Q}_c$. For each curriculum we iterate over every sequence $\{t_i, t_{i+1}, \ldots, t_j\}$, after we removed some of the time slots as mentioned before, and then we add the cost of the maximum number of isolated lectures in this sequence to $\delta_{c,d}^{k_1,k_2}$:

$$\delta_{c,d}^{k_1,k_2} \leftarrow \delta_{c,d}^{k_1,k_2} + W^{\mathbf{IL}} \left\lceil \frac{j-i+1}{2} \right\rceil \tag{45}$$

Now we have calculated an upper bound on the increase in the cost of the isolated lectures. For a course $c \in \mathcal{C}$ and pattern $k_1, k_2 \in \mathcal{K}_{c,d}$, we say that k_1 *dominates* k_2, if $\mathcal{N}_{c,d}^{k_1} \subseteq \mathcal{N}_{c,d}^{k_2}$ and (44) is fulfilled. If one pattern dominates another, then it implies that we can remove the dominated pattern from the solution and the node from the graph. Note that every time we remove a node from the graph, the graph is changed. The change of the graph can affect the dominance of some pairs of nodes. As an example, consider course $c \in \mathcal{C}$, day $d \in \mathcal{D}$, and patterns $k_1, k_2 \in \mathcal{K}_{c,d}$ where (44) applies and $\mathcal{N}_{c,d}^{k_1} \backslash \mathcal{N}_{c,d}^{k_2} = \left\{ v_{c',d}^{k} \right\}$, i.e., there exists a node $v_{c',d}^{k} \in \mathcal{N}_{c,d}^{k_1}$ where $v_{c',d}^{k} \notin \mathcal{N}_{c,d}^{k_2}$, so k_1 does not dominate k_2. Assume that during the pre-process, we removed the node $v_{c',d}^{k}$ from the graph. Then $\mathcal{N}_{c,d}^{k_1} \subseteq \mathcal{N}_{c,d}^{k_2}$ is valid after the removal, and k_1 now dominates k_2. Therefore, whenever a node is removed, we can potentially create more dominated nodes that can be removed. Instead of checking the entire graph over, whenever we remove a node, we iterate through the nodes one at a time and check if it can be removed, while we take the previously removed nodes into consideration. The order in which we processed the nodes was based on a lexicographical decreasing order. For the lexicographical ordering we defined an indicator variable $I_{c,d}^{k}$ for each node $v_{c,d}^{k}$. The indicator variable is set to one, if there exists any pattern $k' \in \mathcal{K}_{c,d}$, where the neighbourhood of $v_{c,d}^{k}$ is contained in the neighbourhood of $v_{c,d}^{k'}$:

$$I_{c,d}^{k} = \begin{cases} 1, & \exists k' \in \mathcal{K}_{c,d} : \mathcal{N}_{c,d}^{k'} \subseteq \mathcal{N}_{c,d}^{k} \\ 0, & \text{otherwise} \end{cases}$$

We then say that for two nodes $v_{c,d}^{k}, v_{c',d}^{k'} \in \mathcal{V}_d$, $v_{c,d}^{k}$ is lexicographically larger than or equal to $v_{c',d}^{k'}$, denoted by $v_{c,d}^{k} \geq_{lex} v_{c',d}^{k'}$, if the following holds:

$$I_{c,d}^{k} > I_{c',d}^{k'} \vee \left(I_{c,d}^{k} = I_{c',d}^{k'} \wedge \left| \mathcal{N}_{c,d}^{k} \right| \geq \left| \mathcal{N}_{c',d}^{k'} \right| \right) \tag{46}$$

Instead of actually removing the nodes from the graph, we kept track of which nodes that were in the graph by the parameter UB_c^k for each course $c \in \mathcal{C}$ and pattern $k \in \mathcal{K}_{c,d}$. First, we initialized UB_c^k to be one for every $c \in \mathcal{C}$ and every $k \in \mathcal{K}_{c,d}$. Then, when a node $v_{c,d}^{k}$ was *removed* from the graph, we changed the corresponding value to zero: $UB_c^k \leftarrow 0$. The values UB_c^k were then used as upper bounds for the corresponding variables in the pricing-problem. The process is summarized in Algorithm 1.

We applied the pre-processing technique in Algorithm 1 before we solved the pricing problem in every iteration of the column generation algorithm. When we solved the pricing problem, we then *unfixed* the variables again, i.e., we changed the upper bounds of the variables back to one,

Algorithm 1: PreProcessPricingProblem

```
input: The nodes $\mathcal{V}_d^{sort}$ in lexicographical decreasing order according to (46), the value
       $\delta_{c,d}^{k_1,k_2}$ for each course $c \in \mathcal{C}$ and patterns $k_1, k_2 \in \mathcal{K}_{c,d}$ where $k_1 \neq k_2$

Initialise the upper bounds $UB_c^k \leftarrow 1$ for all the variables
// Iterate over all the nodes in the sorted order
for $v_{c,d}^{k_2} \in \mathcal{V}_d^{sort}$ do
    // Iterate over all the other feasible patterns for the course
    for $k_1 \in \mathcal{K}_{c,d} \setminus \{k_2\}$ do
        if $UB_c^{k_1} = 1$ and condition (44) is met then
            // If $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ is empty or all the upper bounds for the corresponding
               variables are set to zero, then the upper bound of the variable
               associated with $v_{c,d}^{k_2}$ can be set to zero
            if $\mathcal{N}_{c,d}^{k_1 \setminus k_2} = \emptyset \vee \left(UB_{c'}^{k'} = 1, \forall v_{c',d}^{k'} \in \mathcal{N}_{c,d}^{k_1 \setminus k_2}\right)$ then
                $UB_c^{k_2} \leftarrow 0$
                break
```

so they were ready for the next iteration.

4.2. Optimality Inequalities

In this section, we describe an extension to the pre-processing technique from Section 4.1. Like in the pre-processing phase, the inequalities we derive here are only applicable in a single iteration of the column generation algorithm. Consider some course $c \in \mathcal{C}$ in the pricing-problem for day $d \in \mathcal{D}$ in any iteration of the column generation algorithm. Consider the patterns $k_1, k_2 \in \mathcal{K}_{c,d}$, where $k_1 \neq k_2$. We only considered the patterns k_1 and k_2, in Section 4.1, where we could guarantee that if we had a feasible solution when c was assigned to k_2, then we could create a new feasible solution by reassigning c to k_1. In this section we do not keep this restriction, but consider every pair of patterns k_1 and k_2 where the condition (44) is fulfilled.

Let the neighbourhood of the node $v_{c,d}^{k_1}$ in $\mathcal{G}_d$, except for the nodes corresponding to c, be denoted $\mathcal{N}_{c,d}^{k_1} \subseteq \mathcal{V}_d$. Every node in $\mathcal{N}_{c,d}^{k_1}$ was a neighbour of $v_{c,d}^{k_2}$ in Section 4.1. As we have removed this restriction in this section, then there might be some nodes in $\mathcal{N}_{c,d}^{k_1}$ which are not neighbours of $v_{c,d}^{k_2}$. We denote these nodes by $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$, i.e., every node in $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ is a neighbour of $v_{c,d}^{k_1}$, but not a neighbour of $v_{c,d}^{k_2}$.

Given any feasible solution where c is assigned to k_2 then reassigning c to k_1 is a feasible solution only if the values of the variables corresponding to the nodes in $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ are all zero. This means that if none of the variables in $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ are selected (have a value of one), then we can reassign c to k_1 without increasing the objective value. So it cannot be beneficial to assign c to k_2, if none of the variables in $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ are selected, which leads us to the following inequality:

$$x_c^{k_2} \leq \sum_{v_{c',d}^k \in \mathcal{N}_{c,d}^{k_1 \setminus k_2}} x_{c'}^k \tag{47}$$

Note that if $\mathcal{N}_{c,d}^{k_1 \setminus k_2} = \emptyset$, then the right-hand side of the inequality is zero, and we have the case from the pre-processing in Section 4.1. Hence, we only consider k_1 and k_2 where $\mathcal{N}_{c,d}^{k_1 \setminus k_2} \neq \emptyset$. We do not add the inequalities (47) to the pricing-problem as the number of these constraints is $O\left(|\mathcal{K}_{c,d}|^2\right)$. Instead, we do an aggregation. Consider again the course c and pattern $k_1 \in \mathcal{K}_{c,d}$. Let $\mathcal{V}' \subseteq \mathcal{K}_{c,d}$ be the set of patterns where $\mathcal{N}_{c,d}^{k_1 \setminus k_2} \neq \emptyset$ and where (44) is fulfilled for every $k_2 \in \mathcal{V}'$. We let $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'} \subseteq \mathcal{N}_{c,d}^{k_1}$ denote the union of all the sets $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ for $k_2 \in \mathcal{V}'$, i.e., $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'} = \bigcup_{k_2 \in \mathcal{V}'} \mathcal{N}_{c,d}^{k_1 \setminus k_2}$. We illustrate this in Figure 6.

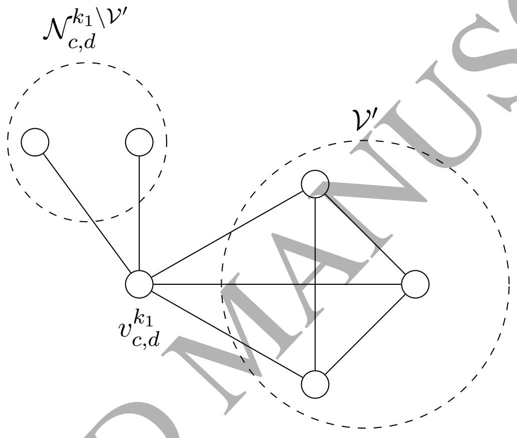

Figure 6: Illustration of the nodes in the pattern conflict graph $\mathcal{G}_d$ corresponding to the course c, the pattern k_1, the set of patterns $\mathcal{V}'$, and the set of nodes $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'} = \bigcup_{k_2 \in \mathcal{V}'} \mathcal{N}_{c,d}^{k_1 \setminus k_2}$.

Consider any feasible solution where none of the nodes in $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'}$ are selected, i.e., the corresponding variables are all set to zero. In this solution it is not beneficial to assign c to any of the patterns in $\mathcal{V}'$, as we can create a new solution by reassigning c to k_1 without increasing the objective value. So we can add the following inequality:

$$\sum_{v_{c,d}^{k'} \in \mathcal{V}'} x_c^{k'} \leq \sum_{v_{c',d}^{k'} \in \mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'}} x_{c'}^{k'} \tag{48}$$

We have to be careful when we add these constraints as they can cut away the optimal solution. As an example, consider the pricing problem for day $d \in \mathcal{D}$, and a course $c_1 \in \mathcal{C}$ where $\mathcal{K}_{c_1,d} = \{k_1, k_2\}$. Assume that both $\left(\alpha_{c_1,d}^{k_2} - \overline{\pi}_{c_1,d}^{k_2}\right) - \left(\alpha_{c_1,d}^{k_1} - \overline{\pi}_{c_1,d}^{k_1}\right) \geq \delta_{c_1,d}^{k_1,k_2}$ and $\left(\alpha_{c_1,d}^{k_1} - \overline{\pi}_{c_1,d}^{k_1}\right) - \left(\alpha_{c_1,d}^{k_2} - \overline{\pi}_{c_1,d}^{k_2}\right) \geq \delta_{c_1,d}^{k_2,k_1}$ holds. Let $v_{c_2,d}^{k_3}$ be the only neighbour of $v_{c_1,d}^{k_1}$, which is not a neighbour of $v_{c_1,d}^{k_2}$, and likewise, let $v_{c_3,d}^{k_4}$ be the only neighbour of $v_{c_1,d}^{k_2}$, which is not a neighbour of $v^{k_1}_{c_1,d}$. The example is illustrated in Figure 7.

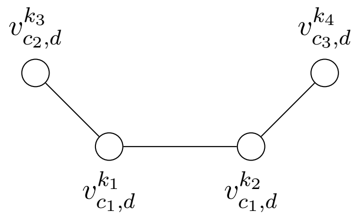

Figure 7: Illustration of the example for the pricing problem for $d \in \mathcal{D}$, and course $c_1 \in \mathcal{C}$, where $\mathcal{K}_{c_1,d} = \{k_1, k_2\}$. $v^{k_3}_{c_2,d}$ is the only neighbour of $v^{k_1}_{c_1,d}$, which is not a neighbour of $v^{k_2}_{c_1,d}$, and $v^{k_4}_{c_3,d}$ is the only neighbour of $v^{k_2}_{c_1,d}$, which is not a neighbour of $v^{k_1}_{c_1,d}$.

In the example in Figure 7 the constraints (48) are: $v^{k_1}_{c_1,d} \leq v^{k_4}_{c_3,d}$ and $v^{k_2}_{c_1,d} \leq v^{k_3}_{c_2,d}$. Assume that $x^{k_3}_{c_2} = x^{k_4}_{c_3} = 0$ in any optimal solution. Due to constraints (39), we must assign c_1 to either k_1 or k_2 as these are the only feasible patterns, however, the constraints (48) implies that $x^{k_1}_{c_1} = x^{k_2}_{c_1} = 0$ in the optimal solution, which is infeasible.

The way we avoid this issue is to create a list L of all the patterns that we allow to be included in the left-hand side of the inequality (48). Initially this list contains all the patterns of c, i.e., $L \leftarrow \mathcal{K}_{c,d}$. We then iterate through every pattern $k_1 \in \mathcal{K}_{c,d}$, and construct the set $\mathcal{V}'$. We then remove the patterns from $\mathcal{V}'$ that are not in the set L. If $\mathcal{V}'$ is non-empty, then we add the inequality (48), and remove k_1 from L. We continue this procedure until all courses and patterns have been processed. If we use the example from Figure 7, then we initialize the list $L \leftarrow \{k_1, k_2\}$. We then process $k_1 \in L$, and construct the set $\mathcal{V}' \leftarrow \{k_2\} \cap L = \{k_2\}$. As $\mathcal{V}' \neq \emptyset$, we add the constraint $x^{k_2}_{c_1} \leq x^{k_3}_{c_2,d}$ and remove k_1 from L, i.e., $L \leftarrow L \backslash \{k_1\} = \{k_2\}$. Next, we process $k_2 \in L$, and construct the set $\mathcal{V}' \leftarrow \{k_1\} \cap L = \emptyset$. As $\mathcal{V}' = \emptyset$, then the constraint $x^{k_1}_{d_1} \leq x^{k_4}_{c_1}$ is not added, and the optimal solution is not cut away. The process is summarised in Algorithm 2.

4.3. Local Branching

In this section, we provide a brief introduction to Local Branching introduced by Fischetti and Lodi (2003), and how we applied it to the pricing problem. Local Branching is a framework for general MIPs, that explores solution neighbourhoods by adding invalid inequalities. The neighbourhoods are searched by a general MIP solver. In our description of the local branching framework we focus on our implementation and do not cover every aspect of the techniques described by Fischetti and Lodi (2003). We refer to Fischetti and Lodi (2003) for details, and a more general description.

Given a feasible solution $\overline{x}$ let $\Delta(x, \overline{x})$ be the distance between the solution $\overline{x}$ and any other solution x. The idea of the local branching framework is then to explore the neighbourhood of the solution $\overline{x}$ by applying the constraint $\Delta(x, \overline{x}) \leq \kappa$ to the MIP model for some value κ and then solve the model with a generic MIP solver. Hopefully this model is much easier to solve than the

Algorithm 2: AddOptimalityInequalities

```
input : The pricing-problem $d \in \mathcal{D}$, an upper bound $UB_c^k$ for each course $c \in \mathcal{C}$ and
        pattern $k \in \mathcal{K}_{c,d}$, and the value $\delta_{c,d}^{k_1,k_2}$ and the set of nodes $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ for each course
        $c \in \mathcal{C}$, pattern $k_1 \in \mathcal{K}_{c,d}$, and pattern $k_2 \in \mathcal{K}_{c,d} \setminus \{k_1\}$

// Iterate over all the courses
for $c \in \mathcal{C}$ do
    // Initialise the list $L$ to contain all the patterns of $c$
    $L \leftarrow \mathcal{K}_{c,d}$
    // Iterate over all the patterns that are feasible for the course $c$
    for $k_1 \in \mathcal{K}_{c,d}$ do
        // Initialise the sets $\mathcal{V}'$ and $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'}$ to be empty
        $\mathcal{V}' \leftarrow \emptyset$
        $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'} \leftarrow \emptyset$
        // Iterate over all the patterns that have not been removed from $L$
        for $k_2 \in L \setminus \{k_1\}$ do
            if $UB_c^{k_2} = 1$ and condition (44) is met then
                // Add $v_{c,d}^{k_2}$ to $\mathcal{V}'$
                $\mathcal{V}' \leftarrow \mathcal{V}' \cup \left\{v_{c,d}^{k_2}\right\}$
                // Add every node in $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$, where the upper bound is one, to
                    $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'}$
                $\mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'} \leftarrow \mathcal{N}_{c,d}^{k_1 \setminus \mathcal{V}'} \cup \left\{v_{c',d}^{k'} \in \mathcal{N}_{c,d}^{k_1 \setminus k_2} \,|\, UB_{c'}^{k'} = 1\right\}$
        if $\mathcal{V}' \neq \emptyset$ then
            Add the constraint (48) to the pricing-problem
            // Remove $k_1$ from the set $L$ to avoid cutting away optimal solutions
            $L \leftarrow L \setminus \{k_1\}$
```

original model, and hopefully, there is a better solution inside the neighbourhood of $\overline{x}$. Since the variables x are binary, then a distance measurement can be the *Hamming Distance*:

$$\Delta(x, \overline{x}) \coloneqq \sum_{c \in \mathcal{C}, k \in \mathcal{K} : \overline{x}_c^k = 0} x_c^k + \sum_{c \in \mathcal{C}, k \in \mathcal{K} : \overline{x}_c^k = 1} \left(1 - x_c^k\right)$$

and κ can be any non-negative integer value. This distance measurement counts the number of variables in x that change value compared to the solution $\overline{x}$. Due to the constraints (39), we know that if one variable changes value from one to zero, then another variable must change value from zero to one. Therefore, we redefine the distance measurement to only count the number of binary variables that change value from zero to one (Fischetti and Lodi, 2003):

$$\Delta(x,\overline{x}) \coloneqq \sum_{c\in\mathcal{C},k\in\mathcal{K}:\overline{x}_c^k=0} x_c^k$$

This distance measurement is referred to as the *Asymmetric Hamming Distance*. When a neighbourhood of a solution has been explored we can then replace the constraint by $\Delta(x,\overline{x}) \geq \kappa+1$ which leads to a branching framework, i.e., given a solution $\overline{x}$ we can branch the problem into two sub-problems:

$$\begin{matrix} \Delta(x,\overline{x}) \leq \kappa \\ \text{(left branch)} \end{matrix} \quad \bigvee \quad \begin{matrix} \Delta(x,\overline{x}) \geq \kappa+1 \\ \text{(right branch)} \end{matrix}$$

In the basic local branching framework the left branch is solved by a generic MIP solver to optimality. If the optimal solution is an improvement of $\overline{x}$, then it is used as the branching solution in the right branch to create two new sub-problems.

When the basic local branching framework cannot find improving solutions, then Fischetti and Lodi (2003) described methods to diversify. In its essence, the goal of the diversification is to find a new solution outside of the explored neighbourhoods. If such a solution can be found, then the local branching framework can be reapplied on that solution.

The question that remains is how to select the solutions on which to apply the local branching framework. The goal of the pricing problem is to find solutions with a negative objective value. So if we consider some solutions which almost have a negative objective value, then we can use them in the local branching framework.

Assume that we are in some iteration of the column generation algorithm, where we have added columns to the RMP in previous iterations. As we are considering an optimal solution of the RMP, none of these columns has a negative reduced cost, but some of them might have a reduced cost of zero, e.g., the columns that are basic. The reduced costs of these columns are equal to the objective values of the corresponding solutions in the pricing-problem. Hence, the previously generated columns with a reduced cost of zero, can be used as the branching solutions for the local branching framework.

In our implementation, we took all the previously generated columns that are basic in the RMP, as we know that the reduced costs of these columns are zero. For each of these columns we let $\overline{z}$ be the number of variables that are set to one in the column and fixed to zero in the pre-processing from Section 4.1. When we add the local branching constraint on that column, then if $\overline{z} \geq \kappa+1$, the model becomes infeasible. Therefore, we only consider columns where $\overline{z} \leq \kappa$. We then put the columns where $\overline{z} \leq \kappa$ in a ordered list, so that the first column in the list is the last column that was generated in a previous iteration, and the last column in the list is the first column to be generated in a previous iteration. We use the first column in the list as the initial solution for the basic local branching framework, and the remaining columns are used for diversification. To illustrate this, consider an example of four previously generated feasible columns with a reduced cost of zero; $\overline{x}^1$, $\overline{x}^2$, $\overline{x}^3$ and $\overline{x}^4$. We denote these columns as 0-columns. In Figure 8a a two-dimensional representation of the solution space of the pricing problem is represented with the four 0-columns.

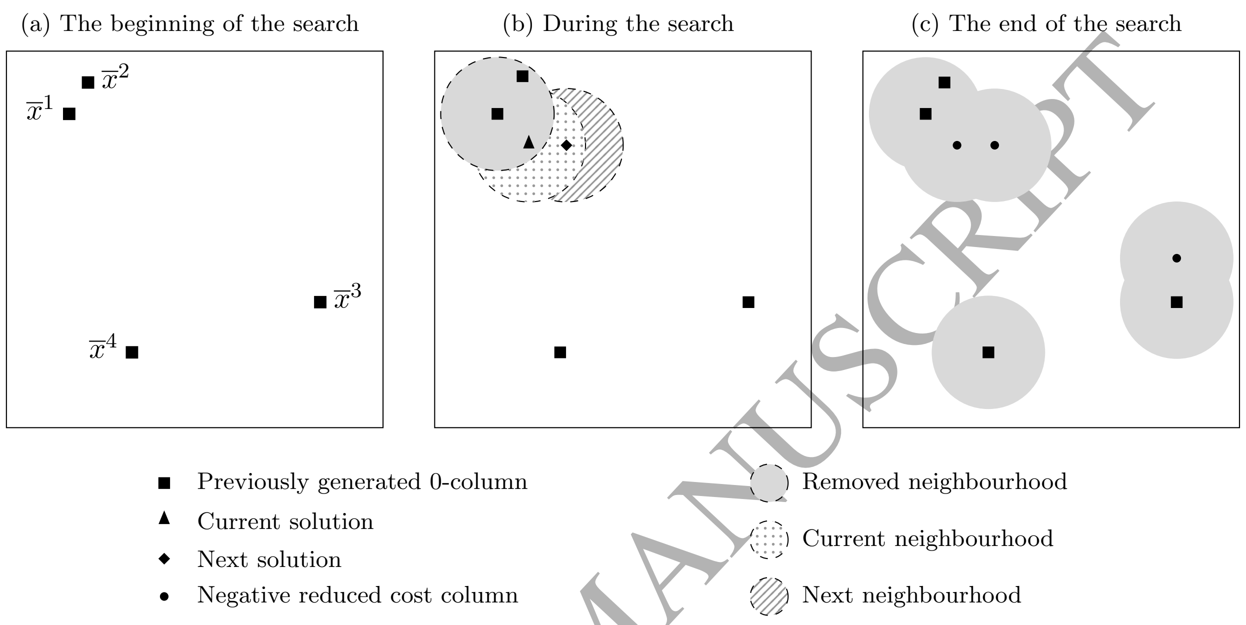

Figure 8: Illustration of the solution space of the example.

We first add $\Delta\left(x,\overline{x}^1\right) \leq k$ to the model and solve it to optimality. Assume that the optimal solution $\overline{x}^1_1$ has a lower objective value than $\overline{x}^1$. We then replace $\Delta\left(x,\overline{x}^1\right) \leq k$ by $\Delta\left(x,\overline{x}^1\right) \geq k+1$ and add $\Delta\left(x,\overline{x}^1_1\right) \leq k$ to the model. Again, we solve the model to optimality and find the solution $\overline{x}^1_2$. In Figure 8b the current state of the search is illustrated. The solution $\overline{x}^1_1$ is illustrated as a triangle and denoted as the *current* solution. The solution $\overline{x}^1_2$ is illustrated as a small diamond and denoted as the *next* solution.

After the neighbourhood of $\overline{x}^1_2$ is explored, then no further improving solutions are found. The solution $\overline{x}^2$ is skipped as it is inside the searched neighbourhoods. Next, the solution $\overline{x}^3$, which is outside the neighbourhoods, is provided. The local branching framework is applied once again, leading to the improved solution $\overline{x}^2_1$. No improved solutions are found in the neighbourhood of $\overline{x}^2_1$, and so the neighbourhood of the last solution $\overline{x}^4$ is searched. Here no improving solutions are found, and the local branching is stopped. In Figure 8c the solution space after running the local branching framework on all the 0-columns is illustrated. The squares mark the 0-columns, and the dots mark the columns found with a negative reduced cost.

If any columns with a negative reduced cost were found during the local branching search, then we added these to the RMP and stopped. We did not search for solutions in the remaining solution space in that iteration of the column generation algorithm. If no columns with a negative reduced cost were generated, then we solved the model to optimality, with all the added right local branching constraints $\Delta(x, \overline{x}) \geq \kappa + 1$, and any columns found here with a negative reduced cost was added to the RMP. Our implementation of the Local Branching heuristic is summarized in Algorithm 3.

Algorithm 3: LocalBranching

```
input : The pricing problem $d \in \mathcal{D}$, the basic columns $basicColumns_d$ from RMP for $d$,
        and the upper bound $UB_c^k$ for each course $c \in \mathcal{C}$ and for each pattern $k \in \mathcal{K}_{c,d}$

output: The columns $columns_d$ with a negative reduced cost
$columns_d \leftarrow \emptyset$
// We initialise the set $considered$ to be empty, as we have not processed any
   basic columns yet
$considered \leftarrow \emptyset$
// $best = 0$
// We iterate over all the basic columns
for $col \in basicColumns_d$ do
    // Let $\overline{x}$ by the solution corresponding to the column $col$
    // Calculate the $\overline{z}$ value to check if the neighbourhood is infeasible
    $\overline{z} \leftarrow \sum_{c\in\mathcal{C},k\in\mathcal{K}_{c,d}} \left(1 - UB_c^k\right) \overline{x}_c^k$
    // Iterate over the previously considered basic columns
    for $col' \in considered$ do
        // Let $\widehat{x}$ by the solution corresponding to the column $col'$
        // Update the $\overline{z}$ value to check if $col'$ is inside a previously explored
           neighbourhood
        $\overline{z} \leftarrow \max\left\{\overline{z}, \sum_{c\in\mathcal{C},k\in\mathcal{K}_{c,d}} \left(1 - UB_c^k\right) \widehat{x}_c^k\right\}$
    if $\overline{z} \leq \kappa$ then
        $considered \leftarrow considered \cup \{col\}$
        Solve the pricing problem with the added local branching constraint $\Delta(\overline{x}, x) \leq \kappa$
        // Let $localColumns$ be all the columns found by the MIP solver, in
           decreasing order of the reduced costs
        for $col' \in localColumns$ do
            // Let $R_{col'}$ be the reduced cost of the generated column $col'$
            if $R_{col'} < best$ then
                $columns_d \leftarrow columns_d \cup \{col'\}$
                $best = R_{col'}$
```

5. Computational Results

In this section, we describe our computational experiments and compare our results with other approaches from literature. We have conducted all tests on an Intel® Core™ i7-6700K CPU

@ 4.00GHz processor with four cores, each with two threads, and 32GB memory, and running Windows 10. We used Gurobi version 7.5.1 provided by Gurobi Optimization, Inc. (2016) both for the RMP and for the pricing problems. We set the pre-solver to the most aggressive setting (Presolve=2), the number of threads that can be run in parallel to be equal to the number of threads of the computer (Threads = 8), and the MIP gap to zero (MIPGap = 0.0). The cut-off value was set to -10^{-5} (CUTOFF = -1e-5) in the pricing problems, so the solver will not return solutions with objective values, which are greater than -10^{-5}. We set the limit on the solutions to be returned to be the maximum possible (PoolSolutions = int.MaxValue), so we can extract all the columns that Gurobi generates with a negative reduced cost. The remaining parameters were set to their default values. We implemented our code in C# and used the Parallel.ForEach method from the System.Threading.Tasks library to solve the pricing problems in parallel as they are independent.

We followed Cacchiani et al. (2013) and tested our algorithm on 20 of the 21 data instances from ITC2007 named comp01 through comp21. The instance that we did not include in the tests was comp11, as the best-known upper bound is zero, thus we cannot improve the trivial lower bound of zero.

We built the models for the pricing problems in Gurobi once, and then changed the objective function accordingly in each iteration, because the pricing problems are the same, except for the objective function, in each iteration. We solved the RMP in every iteration, and retrieved the dual information. For each pricing problem, we started by running the pre-processing from Section 4.1. After the pre-processing, the inequalities from Section 4.2 were added and then the local branching was applied to the resulting model. As the values $\delta_{c,d}^{k_1,k_2}$ and the sets $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ from Section 4.1 and 4.2 are static, i.e., they do not change between iterations, we calculated all of these values and sets, before we started the column generation algorithm. We extracted all columns found by Gurobi, that had a negative reduced cost, and added them to the RMP. Afterwards, we removed all the constraints that we may have added in the pricing problems, and reset the upper bounds to one. We stopped the algorithm if no columns were generated, or if the lower bound of the MP was greater than or equal to the upper bound of the LP-relaxation of the MP. In each iteration of the column generation algorithm, the objective value of the RMP is an upper bound to the LP-relaxation of the MP. The lower bound of the MP is equal to the objective value of the RMP, plus the sum of the lower bounds of all the pricing problems (Desrosiers and Lübbecke, 2010). We know that the objective value for any integer solution is an integer, so we rounded up the lower bound to the nearest integer. The process is summarized in Algorithm 4 and 5. In both of the algorithms, we use the following notations:

$$\delta := \left\{\delta_{c,d}^{k_1,k_2}\right\}_{c\in\mathcal{C},k_1\in\mathcal{K}_{c,d},k_2\in\mathcal{K}_{c,d}\setminus\{k_1\}} \tag{49}$$

$$\mathcal{N} := \left\{\mathcal{N}_{c,d}^{k_1\setminus k_2}\right\}_{c\in\mathcal{C},k_1\in\mathcal{K}_{c,d},k_2\in\mathcal{K}_{c,d}\setminus\{k_1\}} \tag{50}$$

In Algorithm 5 we use the notation:

$$UB := \left\{UB_c^k\right\}_{c\in\mathcal{C},k\in\mathcal{K}_{c,d}} \tag{51}$$

Algorithm 4: ColumnGeneration

```
// The following nested loops pre-calculate all the $\delta_{c,d}^{k_1,k_2}$ and the sets $\mathcal{N}_{c,d}^{k_1\setminus k_2}$
for $d \in \mathcal{D}$ do
    for every node $v_{c,d}^{k_1} \in \mathcal{V}_d$ do
        for $k_2 \in \mathcal{K}_{c,d}\setminus\{k_1\}$ do
            Calculate $\delta_{c,d}^{k_1,k_2}$ from (45)
            Compute the set of nodes $\mathcal{N}_{c,d}^{k_1\setminus k_2}$
Let $\mathcal{V}_d^{sort}$ be the nodes $\mathcal{V}_d$, sorted in the lexicographical decreasing order (46) for each $d \in D$
// Initialise the variable $columnsGenerated$ to be one
$columnsGenerated \leftarrow 1$
// Initialise the upper and lower bound
$UB_{MP} \leftarrow \infty$, $LB_{MP} \leftarrow -\infty$
while $columnsGenerated > 0 \wedge LB_{MP} < UB_{MP}$ do
    $columnsGenerated \leftarrow 0$
    Solve RMP and let $UB_{MP}$ be the objective value
    Retrieve the dual solution $(\overline{\beta}, \overline{\phi}, \overline{\mu}, \overline{\zeta}, \overline{\pi}^0)$ and compute the values $\overline{\pi}_{c,d}^k$ according to (37)
    // Iterate over all the pricing problems in parallel
    for $d \in D$ do
        // $LB_d$ and $columns_d$ is the lower bound and the columns returned by the
           pricing problem $d \in D$
        $(LB_d, columns_d) \leftarrow$ SolvePricingProblem $(d, \mathcal{V}_d^{sort}, \overline{\pi}, \overline{\pi}^0, \delta, \mathcal{N})$
    // The following loop may not be possible to run in parallel
    for $d \in \mathcal{D}$ do
        Add all the columns in $columns_d$ to the RMP
    // Recalculate the lower bound
    $LB_{MP} \leftarrow \max\{LB_{MP}, \lceil UB_{MP} + \sum_{d\in D} LB_d \rceil\}$
    // Calculate the total number of columns generated in this iteration
    $columnsGenerated \leftarrow \sum_{d\in D} |columns_d|$
```

In Table 2 we have reported the timings of the algorithm with three different settings: BASIC,

Algorithm 5: SolvePricingProblem

```
input  : The day $d \in D$ associated with the pricing problem, the nodes $\mathcal{V}_d^{sort}$ in sorted
         order, the dual values $(\overline{\pi}, \overline{\pi}^0)$, and the value $\delta_{c,d}^{k_1,k_2}$ and the set of nodes $\mathcal{N}_{c,d}^{k_1 \setminus k_2}$ for
         each course $c \in \mathcal{C}$, pattern $k_1 \in \mathcal{K}_{c,d}$, and pattern $k_2 \in \mathcal{K}_{c,d} \setminus \{k_1\}$
output: A lower bound $LB_d$ of the pricing problem, and the generated columns $columns_d$

for $v_{c,d}^{k_1} \in \mathcal{V}$ do
    // Initialise the upper bound to one
    $UB_c^k \leftarrow 1$
if pre-processing is activated then
    // Run Algorithm 1
    $UB \leftarrow$ PreProcessPricingProblem $(d, UB, \overline{\pi}, \delta, \mathcal{N})$
if optimality inequalities are activated then
    // Run Algorithm 2
    AddOptimalityInequalities$(d, UB, \overline{\pi}, \delta, \mathcal{N})$
$columns_d \leftarrow \emptyset$
if local branching is activated then
    Extract the basic columns $basicColumns_d$ from RMP for $d$
    // Run Algorithm 3
    $columns_d = LocalBranching\,(d, basicColumns_d, UB)$
if $columns_d = \emptyset$ then
    Solve the pricing problem to optimality
    Extract all solutions where the objective value is negative and add them to $columns_d$
    if $columns_d \neq \emptyset$ then
        Set $LB_d$ to the most negative objective value of the solutions in $columns_d$
    else
        // Set the lower bound to zero, as no columns were generated
        $LB_d \leftarrow 0$
else
    // Calculate a lower bound, by using the information that every course
       must select exactly one pattern, that has an upper bound of one
    $LB_d \leftarrow \overline{\pi}_d^0 + \sum_{c \in \mathcal{C}} \min_{k \in \mathcal{K}_{c,d}} \left\{ \alpha_{c,d}^k - \overline{\pi}_{c,d}^k \,|\, UB_c^k = 1 \right\}$
Remove all optimality inequalities and local branching constraints, if any were added
Reset all the upper bounds $UB_c^k$ to one
```

PREPRO, and PREINEQ. In BASIC, we solved the pricing problems without the pre-processing, inequalities or local branching. In PREPRO, we solved the pricing problems with the pre-processing activated, and without the inequalities and the local branching. In PREINEQ, we solved the pricing problems with the pre-processing and the optimality inequalities, and without the local branching. The timings have been reported in the format `hh:mm:ss` where `hh` is the amount of hours, `mm` is the amount of minutes, and `ss` is the seconds.

Table 2: The total time spent in the column generation algorithm for the basic implementation (BASIC), with pre-processing in the pricing problem (PREPRO), and with pre-processing and inequalities in the pricing problem (PREINEQ).

Instance	BASIC	PREPRO	PREINEQ
comp01	**1:43**	2:41	3:37
comp02	21:42	**13:37**	20:17
comp03	21:14	**14:57**	22:45
comp04	**2:23**	2:31	3:51
comp05	3:08:19	**1:53:53**	2:14:56
comp06	**19:29**	20:05	27:43
comp07	11:53	**11:31**	14:55
comp08	**2:33**	2:51	3:46
comp09	**3:33**	3:50	5:16
comp10	9:23	**8:26**	11:50
comp12	49:33:18	**31:31:44**	49:59:54
comp13	**3:36**	3:58	5:47
comp14	5:19	**4:57**	6:41
comp15	20:10	**15:06**	21:06
comp16	24:56	**21:08**	27:06
comp17	36:58	**28:44**	49:34
comp18	38:53	**34:11**	44:12
comp19	3:37	**3:36**	5:33
comp20	**13:26**	14:38	19:51
comp21	50:59	**36:07**	1:05:16
Total	57:33:23	37:28:32	58:13:57
Best	**7**	**13**	**0**

In Table 2, we see that the pre-processing improves the running time for 13 of the 20 instances. The additional time for the 7 instances where the running time is not improved is within a few minutes. This is due to the time spent on calculating the δ values and the sets $\mathcal{N}$. Overall, we observe that the pre-processing reduces the total running time of all the instances, by nearly 20 hours. We also see that adding the optimality inequalities does not help the algorithm, in fact it makes the running time longer. The reason for this is that these constraints are generated and added in every iteration. In any given iteration, we create thousands of these constraints, so most of the time is spent on altering the pricing problems. Another thing we can see in Table 2 is that instance comp12 has a significantly longer running time. The reason for this is related to the curricula and the **IL** constraints.

In Table 3 we have reported the results of the column generation algorithm when the **IL** constraints have been removed. For each instance, we have reported the total number of variables in the pricing problems (s) used to model the **IL** constraints. In the next two columns (CG w/o **IL**), we have reported the lower bound obtained (LB), and the total running time of the column generation algorithm (Time), when all the **IL** constraints have been removed from the pricing problems. In the last three columns (Pricing problem components), we have reported the statistics of the connected components of the graphs $\mathcal{G}_d = (\mathcal{V}_d, \mathcal{E}_d)$ for the pricing problems. We have reported the statistics of the connected components, as the nodes are related to variables in the pricing problems, and the edges are related to the **C** constraints. So, if the graph $\mathcal{G}_d$ has many connected components, then we can expect the associated pricing problem to be highly decomposable. Gurobi is able to automatically detect, and exploit when the models can be decomposed. For each instance, we report the total number of connected components for all the pricing problems (Total), the average number of nodes in each connected component (Avg. nodes), and the average number of edges in each connected component (Avg. edges).

Table 3: Statistics of the pricing problems (Pricing problems), and the column generation algorithm when the **IL** constraints are removed (CG w/o **IL**). For each instance the number of variables to model the **IL** constraints is reported (s).

		CG w/o **IL**		Pricing problem components		
Instance	s	LB	Time	Total	Avg. nodes	Avg. edges
comp01	390	0	1:35	10	792.7	116750.1
comp02	1579	0	2:16	15	492.5	38794.9
comp03	1463	0	1:29	34	192.3	14074.2
comp04	1118	0	1:17	50	156.7	9461.3
comp05	3357	15	2:26	14	410.6	49266.2
comp06	1593	0	2:29	44	225.9	16193.5
comp07	1810	0	3:52	19	654.5	46654.7
comp08	1219	0	1:14	41	199.5	9707.2
comp09	1412	0	1:13	22	338.8	21074.2
comp10	1570	0	2:45	15	675.3	47143.9
comp12	3934	0	3:30	7	1163.3	141774.7
comp13	1265	0	1:11	50	157.4	8655.6
comp14	1312	0	1:41	28	266.9	21217.6
comp15	1463	0	1:31	34	192.3	14074.2
comp16	1651	0	2:36	25	419.6	29989.2
comp17	1546	0	2:16	35	267.5	18819.3
comp18	1355	0	1:08	12	489.0	50973.7
comp19	1287	0	1:23	39	168.9	10818.8
comp20	1863	0	3:13	12	915.6	66151.5
comp21	1639	0	3:07	36	261.9	19704.5

In Table 3, we see that when we removed the **IL** constraints, the column generation algorithm was significantly faster for all the instances. We also see that the number of variables to model the **IL** constraints, is largest for instance comp12, which was also the most time consuming instance. We see that the number of variables to model the **IL** constraints, is not much larger in comp12 than in comp05. However, when we consider the connected components in the pricing problems, we see that comp12 has the lowest number of connected components in total, and the average size of the connected components is largest for comp12 both in terms of nodes and edges, which makes the pricing problems intractable.

Next, we activated the local branching heuristic in the pricing problems. We need to decide the value for κ in the local branching framework. Fischetti and Lodi (2003) suggest to set it between 10 and 20, which in our case is between 5 and 10, as we are using the *Asymmetric Hamming Distance*. So we have tested the algorithm for $\kappa = 5$ and $\kappa = 10$. Furthermore, we have also tested for $\kappa = 2$ to mimic a 2-exchange heuristic (Wolsey, 1998). The total running time, including the time for building the model, the pre-processing and the enumeration of cliques, is reported in Table 4 for $\kappa \in \{2, 5, 10\}$, and for the algorithm without local branching (PREPRO). In the line (Total), the total amount of time spent is reported for each value of κ, and (Best) counts the number of times each value of κ has the lowest running time.

Table 4: The total time spent in the column generation algorithm without local branching (PREPRO) compared to local branching for different values of κ.

Instance	PREPRO	$\kappa = 2$	$\kappa = 5$	$\kappa = 10$
comp01	**2:41**	3:37	2:57	2:54
comp02	**13:37**	42:16	43:19	29:44
comp03	**14:57**	20:22	28:08	23:25
comp04	**2:31**	18:06	8:34	6:41
comp05	1:53:53	**1:31:38**	2:31:57	4:29:03
comp06	**20:05**	50:14	51:33	54:43
comp07	**11:31**	1:20:48	41:14	35:51
comp08	**2:51**	18:59	9:06	9:28
comp09	**3:50**	25:51	14:32	14:13
comp10	**8:26**	1:26:17	40:31	30:58
comp12	31:31:44	**18:43:19**	22:56:24	52:43:50
comp13	**3:58**	26:59	17:20	16:04
comp14	**4:57**	22:05	17:11	15:05
comp15	**15:06**	17:50	21:56	18:56
comp16	**21:08**	1:22:25	1:09:42	40:06
comp17	**28:44**	49:57	1:04:27	1:10:50
comp18	**34:11**	57:12	39:53	3:37:03
comp19	**3:36**	15:37	12:23	10:10
comp20	**14:38**	2:26:58	1:41:10	1:12:05
comp21	**36:07**	51:27	1:15:31	1:26:43
Total	37:28:32	**33:51:54**	36:27:48	69:47:49
Best	**18**	**2**	**0**	**0**

In Table 4 we see that for most of the instances, our implementation of the local branching framework did not help to reduce the computational time. The reason is that the time it took for Gurobi to solve the pricing problems for these instances was very close to the time it took to solve a single neighbourhood ($\Delta(x, \overline{x}) \leq \kappa$), and since we solved multiple neighbourhoods in the local branching framework, the algorithm was slowed down. If we look at the total running time for all the instances it can be seen that the algorithm was fastest for $\kappa = 2$. Most of that improvement of the running time was due to instance comp12, where the running time was improved by almost 13 hours.

In Table 5 we report the statistics of the fastest version of the column generation algorithm for each instance. For all of the instances, we report the statistics when the pre-processing was activated, and the optimality inequalities were deactivated. For the instances, comp05 and comp12, we also activated the local branching framework for $\kappa = 2$. For each instance we report the number of iterations (Iter.). Then in the two following columns (Columns), we report the number of columns that were generated in total (Total), and, for comp05 and comp12, how many of them that were generated by local branching (LocBra.). In the next four columns (Time) we report the timings of the algorithm. The total time spent by the algorithm is reported in the column (Total). The time spent on building the models, enumerating the cliques, and pre-processing the pattern formulation is reported in the column (Build). The next two columns, report the total time spent on solving the RMP (Master), and the total time spent on solving the pricing problems (Pricing). In the time spent on the pricing problems, the pre-processing, constraint generation and local branching is included. The timings are given in the format `hh:mm:ss` as in Table 2. The last two columns (Patterns), report the total number of pattern variables in the pricing problems (Total), and the average number of pattern variables that were removed by the pre-processing in the pricing problems. The last line reports the average time spent in each part, compared to the total time, and the average number of pattern variables removed, compared to the total number of pattern variables in the pricing problems.

In Table 5 we see that for comp05 and comp12, the local branching framework was responsible for almost all of the columns generated. We also see that more than 96% of the total running time was spent on solving the pricing problems. Thus, more research on solution methods for the pricing problems are needed before the column generation algorithm can effectively be extended to a Branch & Price algorithm. Furthermore, we see that 0.4% of the time, on average, was spent on solving the RMP, and lastly, 2.2% of the time, on average, was spent on building all the models, and making the pre-calculations. We also see that almost half of the pattern variables were removed, on average, in each iteration of the algorithm. Next, we compared the lower bounds obtained for the four open instances with the best-known bounds found, reported on the website Bonutti et al. (2017). We report the results in Table 6, where we have updated the best-known lower bounds with the bounds by Bagger et al. (2016). The last line in the table reports the average gap from the best-known upper bound.

In Table 6 we see that our approach obtained a lower bound, which is an improvement of the best-known lower bound for all four of the open instances. These improvements reduce the average gap from the best-known upper bounds on these four instances from 24% to 11%.

In Table 7 we compare the lower bounds we obtained with the existing literature on the instances

Table 5: Statistics of the column generation algorithm, where the pre-processing is activated, the optimality inequalities are deactivated, and the local branching is only activated for comp05 and comp12, with $\kappa = 2$.

		Columns		Time				Patterns	
Instance	Iter.	Total	LocBra.	Total	Build	Master	Pricing	Total	Removed
comp01	45	578		2:41	2:23	0	14	7927	3722.3
comp02	176	6303		13:37	1:51	7	11:23	7387	3004.8
comp03	165	5632		14:57	1:17	5	13:21	6539	2652.5
comp04	246	5446		2:31	54	8	1:08	7835	4003.7
comp05	170	5819	5320	1:31:38	2:33	4	1:28:37	5495	1589.9
comp06	254	11062		20:05	1:33	30	17:34	9941	4791.9
comp07	221	9876		11:31	2:10	39	8:11	12436	5893.0
comp08	315	6906		2:51	48	12	1:24	8180	4276.4
comp09	224	4682		3:50	54	5	2:33	7453	3438.5
comp10	225	9150		8:26	1:36	28	5:56	10129	4682.3
comp12	287	14553	13381	18:43:19	3:20	40	18:38:09	8143	2278.0
comp13	280	6660		3:58	51	11	2:33	7870	3776.0
comp14	162	5640		4:57	1:09	6	3:27	7472	3474.4
comp15	164	5345		15:06	1:17	4	13:30	6539	2684.7
comp16	244	10178		21:08	1:34	28	18:36	10491	5189.0
comp17	250	10998		28:44	1:29	27	26:21	9362	4526.9
comp18	185	4411		34:11	1:19	3	32:22	5868	2654.7
comp19	220	5446		3:36	1:06	6	2:07	6588	3003.0
comp20	263	11432		14:38	1:55	44	11:24	10987	5023.6
comp21	276	11175		36:07	2:24	24	32:51	9429	4137.0
Avg.					2.2%	0.4%	96.8%		45.0%

Table 6: Comparison with the best-known bounds for the four open instances. *Updated value from Bagger et al. (2016).

	Best			DW	
Instance	UB	LB	Gap	LB	Gap
comp03	64	54*	16%	58	9%
comp05	284	211	26%	247	13%
comp12	294	175*	40%	248	16%
comp15	62	54*	13%	58	6%
Avg.			24%		11%

comp01–comp10 and comp12–comp14, as these were the only instances, out of the 20 instances we tested, that were available for all the methods in literature. We compare the lower bounds obtained by our approach (DW) with BMPR10 (Burke et al., 2010b), BMPR12 (Burke et al., 2012), LL12 (Lach and Lübbecke, 2012), HB11 (Hao and Benlic, 2011), CCRT13 (Cacchiani et al., 2013), BKSS17 (Bagger et al., 2017) and BDD16 (Bagger et al., 2016). If a paper reported multiple lower bounds, then we took the highest lower bound obtained for each instance. We mark the lower bound obtained in bold font, if the bound is at least as good as the other published approaches,

for each approach and each instance. If an approach obtained a lower bound which is better than all the other approaches in the same table then we have marked it with an underline. In the table, we also report the average gap from the best-known upper bounds (Avg.). In the second last line (Best), we report the number of times each approach obtained a lower bound which is at least as good as the other approaches. In the last line we report the number of times each approach obtained a lower bound which is better than the other approaches. We have used the same notation throughout all the remaining tables.

Table 7: Comparison of the lower bounds for the different approaches.

Instance	UB	BMPR10	BMPR12	LL12	HB11	CCRT13	AN14	BKSS17	BDD16	DW
comp01	5	**5**	**5**	4	4	**5**	0	**5**	0	0
comp02	24	1	6	11	12	16	16	8	**24**	20
comp03	64	33	43	25	38	52	28	38	54	**58**
comp04	35	**35**	2	28	**35**	**35**	**35**	**35**	**35**	**35**
comp05	284	119	183	108	183	166	48	186	210	**247**
comp06	27	16	6	10	22	11	**27**	16	26	23
comp07	6	**6**	0	**6**	**6**	**6**	**6**	**6**	**6**	**6**
comp08	37	**37**	2	**37**	**37**	**37**	**37**	**37**	**37**	**37**
comp09	96	68	0	46	72	92	35	74	**96**	92
comp10	4	**4**	0	**4**	**4**	2	**4**	**4**	**4**	**4**
comp12	294	101	7	53	109	100	99	142	175	**248**
comp13	59	54	0	41	**59**	57	**59**	**59**	**59**	**59**
comp14	51	42	0	46	**51**	48	**51**	44	**51**	49
Avg.		28.0%	77.5%	35.0%	19.4%	21.7%	31.0%	20.8%	14.3%	13.7%
Best		**5**	**1**	**3**	**6**	**4**	**7**	**6**	**8**	**8**
							1		**2**	**3**

In Table 7, we see that our approach obtained a lower bound which is at least as good as the lower bounds of the other approaches on eight of the instances. On three of these instances the lower bound we obtained is better than for the other approaches. Furthermore, we see that our approach has the lowest average gap (13.7%) to the best-known upper bounds.

In Table 8, we compare our results for all 20 instances to HB11, CCRT13, AN14, BKSS17, and BDD16 since they reported results for all these instances.

In Table 8, we see that our approach obtained a lower bound which is at least as good as the lower bound obtained by the other approaches, for 11 of the instances. On four of these instances, our approach obtained a higher lower bound than the other approaches. These four instances are the open instances. Lastly, we see that our approach obtained the second lowest average gap (12.2%) to the best-known upper bounds, where the original pattern formulation has the lowest

Table 8: Comparison of the lower bounds for the different approaches for all 20 instances.

Instance	UB	HB11	CCRT13	AN14	BKSS17	BDD16	DW
comp01	5	4	**5**	0	**5**	0	0
comp02	24	12	16	16	8	**24**	20
comp03	64	38	52	28	38	54	**58**
comp04	35	**35**	**35**	**35**	**35**	**35**	**35**
comp05	284	183	166	48	186	210	**247**
comp06	27	22	11	**27**	16	26	23
comp07	6	**6**	**6**	**6**	**6**	**6**	**6**
comp08	37	**37**	**37**	**37**	**37**	**37**	**37**
comp09	96	72	92	35	74	**96**	92
comp10	4	**4**	2	**4**	**4**	**4**	**4**
comp12	294	109	100	99	142	175	**248**
comp13	59	**59**	57	**59**	**59**	**59**	**59**
comp14	51	**51**	48	**51**	44	**51**	49
comp15	62	38	52	28	38	54	**58**
comp16	18	16	13	**18**	13	**18**	17
comp17	56	48	48	**56**	44	53	**56**
comp18	61	24	**52**	27	36	**52**	**52**
comp19	57	56	48	46	56	**57**	51
comp20	4	2	**4**	**4**	0	**4**	3
comp21	74	61	68	42	57	**74**	71
Avg.		22.3%	19.0%	28.8%	26.2%	10.9%	12.2%
Best		**6**	**6**	**10**	**6**	**13**	**11**
				1		**4**	**4**

average gap (10.9%).

We realize that even better results might be achieved by extending the Column Generation algorithm into a full Branch & Price algorithm. This, combined with cutting plane techniques in the column generation algorithm, could potentially increase the lower bounds. Local branching is a generic method, that can be used in pricing problems for other column generation algorithms. To the best of our knowledge, this work is the first implementation of local branching in the pricing problem for a column generation algorithm. We think that many pricing problems, in general, work well with local branching since every iteration of the column generation algorithm provides new solutions to be used in the framework.

6. Conclusion

In this paper, we applied the Dantzig-Wolfe decomposition to a pattern formulation for the Curriculum-based Course Timetabling (CCT) problem. The pattern formulation is based on enumerating all the time schedules to which the courses can be assigned each day. The pattern formulation only considers the time schedule of the problem, while ensuring that a feasible room assignment can be found, but does not calculate the violations of the room related soft constraints. Thus, the pattern formulation provides feasible time schedules and a lower bound, i.e., solutions where there exists feasible room assignments, and a guarantee that feasible solutions for CCT of costs better than a certain value does not exist. The decomposition resulted in a pricing problem for each day, where each pricing problem generated a schedule for an entire day. We showed that the pricing problem contained a special structure, which we utilized in a pre-processing phase. We then showed how the pre-processing technique could be used to derive inequalities for the pricing problem. Lastly, we described how we applied Local Branching to solve the pricing problem. To the best of our knowledge, this is the first time Local Branching is implemented in a pricing problem, but it is general enough to be applied in other column generation algorithms. We tested our algorithm on 20 data instances used in the Second International Timetabling Competition. On these instances, the pre-processing technique we applied removed approximately 45% of the pattern variables from the pricing problem on average. We compared the lower bounds that we obtained with other approaches from literature. Our algorithm obtained a lower bound, in 11 of the instances, which was at least as good as the other approaches. Four of these instances are still *open*, meaning that the best-known upper bound does not equal the best-known lower bound. In all of these four instances, our algorithm improved the lower bound, so, the average gap was decreased from 24% to 11%. We showed that more than 96% of the total time of the algorithm was spent on solving the pricing problem, and concluded that more research is needed in the pricing problems before the algorithm can be extended into a full Branch & Price algorithm.

Acknowledgments

The authors would like to thank Guy Desaulniers and Jacques Desrosiers for their contribution in the previous work on which this paper is based. The authors would also like to thank Christina Scheel Persson, Sarah MacPhee, and the anonymous reviewers for their valuable feedback to improve the manuscript.

Funding: This work is part of an industrial PhD project funded by Innovation Fund Denmark (IFD). IFD has only supported this work financially, and has not taken part in any research related activities.

References

Asín Aschá, R., Nieuwenhuis, R., 2014. Curriculum-based course timetabling with SAT and MaxSAT. Annals of Operations Research 218, 71–91.

Bagger, N.-C. F., Desaulniers, G., Desrosiers, J., 2016. Daily course pattern formulation and valid inequalities for the curriculum-based course timetabling problem. Submitted to Journal of Scheduling. Preprint available from

Les Cahiers du GERAD.
URL https://www.gerad.ca/en/papers/G-2016-71

Bagger, N.-C. F., Kristiansen, S., Sørensen, M., Stidsen, T. R., 2017. Flow Formulations for Curriculum-based Course Timetabling. Submitted to Annals of Operations Research. Preprint available at Optimization-Online.org.
URL http://www.optimization-online.org/DB_HTML/2016/12/5786.html

Bettinelli, A., Cacchiani, V., Roberti, R., Toth, P., 2015. An overview of curriculum-based course timetabling. TOP 23 (2), 313–349.

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.

Bonutti, A., Di Gaspero, L., Schaerf, A., 2017. Curriculum-based course timetabling. Last visited: January 15, 2017.
URL http://tabu.diegm.uniud.it/ctt/index.php

Bron, C., Kerbosch, J., 1973. Algorithm 457: Finding All Cliques of an Undirected Graph. Communications of the ACM 16 (9), 575–577.
URL http://doi.acm.org/10.1145/362342.362367

Burke, E. K., Mareček, J., Parkes, A. J., Rudová, H., 2008. Penalising Patterns in Timetables: Novel Integer Programming Formulations. In: Operations Research Proceedings 2007. Springer, pp. 409–414.

Burke, E. K., Mareček, J., Parkes, A. J., Rudová, H., 2010a. A supernodal formulation of vertex colouring with applications in course timetabling. Annals of Operations Research 179 (1), 105–130.

Burke, E. K., Mareček, J., Parkes, A. J., Rudová, H., 2010b. Decomposition, reformulation, and diving in university course timetabling. Computers & Operations Research 37 (3), 582–597.

Burke, E. K., Mareček, J., Parkes, A. J., Rudová, H., 2012. A branch-and-cut procedure for the Udine Course Timetabling problem. Annals of Operations Research 194 (1), 71–87.

Cacchiani, V., Caprara, A., Roberti, R., Toth, P., 2013. A new lower bound for curriculum-based course timetabling. Computers and Operation Research 40 (10), 2466–2477.

Desrosiers, J., Lübbecke, M. E., 2010. A Primer in Column Generation. In: Guy Desaulniers, Jacques Desrosiers, Marius M. Solomon (Eds.), Column Generation. Springer Science+Business Media, Inc., Ch. 1, pp. 1–32.

Di Gaspero, L., Schaerf, A., McCollum, B., 2007. The Second International Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3) preliminary presentation . Association for the Advancement of Artificial Intelligence (www.aaai.org).

Fischetti, M., Lodi, A., 2003. Local branching. Mathematical Programming 98 (1-3), 23–47.

Gurobi Optimization, Inc., 2016. Gurobi Optimizer Reference Manual.
URL http://www.gurobi.com

Hao, J. K., Benlic, U., 2011. Lower bounds for the ITC-2007 curriculum-based course timetabling problem. European Journal of Operational Research 212 (3), 464–472.

Kou, L. T., Stockmeyer, L. J., Wong, C. K., 1978. Covering Edges by Cliques with Regard to Keyword Conflicts and Intersection Graphs. Communication of the ACM 21 (2).

Lach, G., Lübbecke, M. E., 2008. Optimal University Course Timetables and the Partial Transversal Polytope. In: International Workshop on Experimental and Efficient Algorithms. Springer, pp. 235–248.

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

Lübbecke, M. E., 2015. Comments on: An Overview of Curriculum-Based Course Timetabling. TOP 23 (2), 359–361.

Martin, R. K., 1999. Large scale linear and integer optimization: a unified approach. Kluwer Academic Publishers.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., Di Gaspero, L., Qu, R., Burke, E. K., 2010. Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing 22 (1), 120–130.

Moon, J., Moser, L., 1965. On cliques in graphs. Israel Journal of Mathematics 3 (1), 23–28.

Wolsey, L. A., 1998. Heuristic Algorithms. In: Graham, R. L., Lenstra, J. K., Tarjan, R. E. (Eds.), Integer Programming. John Wiley & Sons, Inc., Ch. 12, pp. 203–220.