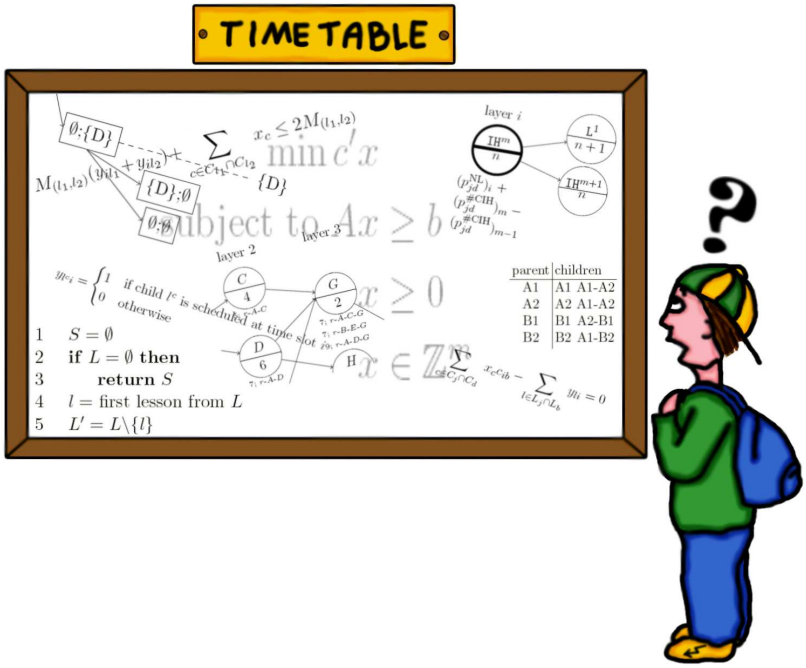
Master's Thesis

Timetabling and Clustering
in Dutch Secondary Schools
using Linear Programming and Column Generation

June 8, 2009



Universiteit Utrecht

Author:
Casper Zelissen

Supervisor:
Dr. Han Hoogeveen

Preface

When I was in secondary school, one of the vice-principals was responsible for timetabling. Most of the time, his office was crowded with little cards and huge tables representing lessons and timetables, respectively. His job was to combine the two, producing timetables that were acceptable to both students and teachers. Although I was never completely satisfied with my own timetable, I then realized his job must have been very difficult. At that time, I was already intrigued by this timetabling problem and it came as a pleasant surprise when my supervisor Han Hoogeveen proposed to conduct my graduation research into this subject.

When I told my tutor Erik Balder that I wanted to do my research in the area of scheduling, he advised me to contact Han. Following this advise certainly was a good decision: Han proved to be a supervisor that always patiently awaited my progress. At times, he gave me excellent tips that helped me to move on, when some problems arose. Most of all, I appreciated that Han always granted me the freedom of taking my own initiative and to work in my own pace: this is freedom I need!

Another person that I owe big thanks is Victoire Camps, timetabler of a secondary school in Amersfoort. She was the only one (among the many timetablers I contacted) that made time for letting me interview her. These interviews helped me a lot: not only did she provide me with all information I needed to understand the things that are important for timetabling, she also gave me data that I could use as input for my own algorithms.

Apart from Han and Victoire, there are three other persons I have to thank. First of all, I would like to thank my girlfriend Mirjam. Not only because she designed the front cover, but also because she endured my moodiness when I was working on this thesis. I also want to thank Guido Diepen who helped me setting up and using the optimization software CPLEX. Finally, my tutor Erik Balder deserves my acknowledgment for reviewing this thesis.

Contents

Chapter 1

Introduction

Like I already mentioned in the preface, this thesis is about secondary school timetabling. To be more precise, I emphasize that with 'timetabling', I do *not* mean handling the daily changes in the schedules due to ill teachers etcetera, but I investigated how to make the schedules for all teachers and classes that serve as a starting point for the rest of the year (or couple of months).

To find out what kind of things we have to take into account and how timetabling is currently done, I interviewed timetabler Victoire Camps from *Meridiaan College 't Hooghe Landt* (HLC) in Amersfoort a couple of times. I learned that nowadays, virtually every school uses extensive (and expensive) software packages for this task. It is not uncommon for a modern school to employ full-time timetablers to operate these programs. Because this software allows them to input more and more information and constraints, timetables (hopefully) get better and expectations get higher. Victoire also provided me with data files from her timetabling program, which I could use to test my own algorithms.

1.1 Topics and Goals

Today's software uses local search algorithms, which are fast and easy to implement. A known disadvantage of local search is that it finds local optima. (Integer) Linear Programming, or (I)LP, is a method that does not have this disadvantage. In this thesis we investigate whether it is possible to solve this problem using (I)LP. We will see that this is indeed possible utilizing *Column Generation* (CG) techniques. It is my intention approximate real-life timetabling as good as possible, and to be able to meet as much real-life constraints as possible. Moreover, my algorithm has to produce a solution within a reasonable amount of time.

From my interviews at the HLC, I learned that before timetabling (which is basically assigning lessons to time slots) can start, another optimization

problem has to be solved: *clustering*. This problem is about assigning students to groups of their optional subjects (this applies to students in upper school) and combining these groups that can be given simultaneously. While not initially my plan, I have decided to extend my research to this problem as well. This is because the solution to the clustering problem highly affects the quality of solutions to the timetabling problem. We will see that again linear programming can be used to solve this problem.

Summarizing, the main goal of my research can be formulated as follows:

> Design a (integer) linear programming model for both the timetabling- and the clustering problem that is able to meet as much real-life constraints as possible and produces a solution within a reasonable amount of time.

1.2 Outline of the Thesis

This thesis consists of four parts. In part I, the timetabling problem is treated. Although clustering must actually be done *before* timetabling, I still decided to treat the clustering problem after the timetabling problem, in part II. This is because I think one can only understand the clustering problem completely, if he knows what the timetabling problem holds. The outline of parts I and II is exactly the same: they begin with a problem description, followed by a chapter describing how I modeled the problem. Then I treat the column generation part and I conclude with a chapter describing some implementation matters.

In part III, the two problems come together. In its first Chapter 10, I propose some adaptations to both problems in order to make them 'cooperate' better. This part also contains the computational results and the conclusion. There are three appendices that together make up part IV. In Appendix A, I show what my JAVA programs look like, and Appendix B treats the several optimization techniques that I used in this chapter to serve those who are unfamiliar with this theory. Finally, Appendix C shortly describes all important concepts, variables and constraints that were used in this thesis, and can serve as a reference while reading this thesis.

Part I

Timetabling

Chapter 2

Problem Description

Introduction As I mentioned in the introduction, the timetabling problem is basically assigning lessons to time slots. When I started working on this project, my idea of this problem was much simpler than it turned out to be. After my first interview at the HLC I returned home quite disillusioned by the number of things that have to be taken into account, and that I had never been thinking of. My idea of timetabling was probably biased by the fact that at my own school all was done by hand. Moreover, the HLC was not exactly the simplest of schools I could have chosen. In this chapter we will see which concepts play a crucial role in this problem and how these concepts relate to each other. I also mention which of the real-life constraints I have disregarded and explain why. In Section 2.2, I treat the objective. Finally, I will define the problem with its constraints and objective.

2.1 Modeling Concepts

2.1.1 Classes

At the HLC, there are 59 classes in total. It is a multilateral school, which means that it teaches at different levels: VWO (6 years), HAVO (5 years) and VMBO-T (4 years). The upper school for HAVO and VWO students starts in year 4, for VMBO-T students in year 3. This subdivision between lower- and upper school is relevant, because in lower school each student has exactly the same timetable as his classmates, whereas in upper school students choose optional subjects resulting in individual timetables. Each class can have its own preferences regarding the layout of timetables (see Section 2.2.1), such as: first-year classes are not allowed to have intermediate hours, VMBO upper school classes are preferred to have the first hour off each day etc.

2.1.2 Teachers

Of the nearly 150 teachers at the HLC, only about 20% has a full-time appointment, and are therefore available to teach at all time slots. The other teachers may block parts of the week that they wish to be off. A part-time teacher may decide to block fewer hours than he (or she, I use the male pronouns when I refer to teachers and students from now on) is entitled to, but has to keep in mind that all non-blocked time slots are available to the timetabler. Teachers who live far away from school usually prefer to have the first hour off. Most teachers do not like intermediate hours, but some do (but not too many, this typically applies to teachers with other tasks, besides teaching). Normally, the situation that a teacher has to go to school for just one hour teaching is being avoided. All these cases can be regarded as timetable layout preferences (see Section 2.2.1). In that sense, they are similar to classes.

In the current timetabling system, all teachers also have a classroom in which they prefer to teach. I will treat this issue in Section 2.1.5 covering classrooms.

2.1.3 Lessons

This timetabling problem is about scheduling lessons. The key properties of a lesson are the class that has the lesson, the teacher who gives the lesson and the number of hours the lesson must be given in a week (let us call this the *multiplicity* of the lesson). Almost all lessons in lower school are of this form. To be able to deal with all the different kinds of objects that must be scheduled, we need a more general description of the concept 'lesson'. For instance, at the HLC also the staff meetings are being scheduled. A meeting can be seen as a lesson with several teachers, but no classes. Therefore I allow a lesson to be defined by a *set* of teachers and a set of classes, together with its multiplicity. Note that I allow one of the sets to be empty (as is the case with meetings). The teachers sets, classes sets and multiplicity of lessons are all predetermined, that is, not part of the problem. This means that I do not need to decide which teacher teaches which class.

Cluster Patterns

A *cluster pattern* is a special case of a lesson, which I will treat in more detail in Chapter 6. This kind of lesson appears in upper school where students have optional subjects. Apart from the whole-class lessons that everyone has (like English), there are times when students go their own way to have class in their optional subjects. This results in lessons in which students from different classes are mixed. Ideally, several of those lessons can be given simultaneously. These lessons together are called a cluster pattern and can be seen as one big lesson that involves several classes and teachers.

Double Hours

Sometimes it is desirable to have a lesson for two hours in a row. This applies for instance to Physical Education (PE), in order to minimize the loss of time due to changing clothes. Such a lesson is called a double hour. It may even happen that a lesson must be scheduled for more than two consecutive hours, e.g. long meetings. I will call this an n-double hour (where n is the number of consecutive hours). The number of required double hours (and of which type) can thus be an additional property for a lesson. A meeting of multiplicity 5, for instance, can have the property that it must be scheduled as a 5-double hour, but also as two double hours and a single hour.

Fixed Lessons

Sometimes the timetabler (or maybe some other high-placed staff member) predetermines at what time a lesson must be scheduled. In other words, this lesson is being fixed to a time slot. This sometimes occurs to staff meetings. Although I do not have to schedule such a lesson, it is still important to include it in the model, since a fixed lesson can cause intermediate hours.

KWT Lessons

At the HLC, there is a system that allows upper school students to fill their intermediate hours by taking coaching lessons in subjects that they choose themselves. These so-called *keuze-werktijd-lessen* (KWT lessons) also give students the opportunity to reach the legal minimum number of lesson hours he must have in a year. Before it can be determined which KWT lessons can be chosen at which time slots, the regular lessons have to be scheduled before. This is because information about intermediate hours of students and free hours of teachers is required. Therefore, the scheduling of KWT lessons happens after the 'regular' timetabling, and is not a part of the timetabling problem.

2.1.4 Time

The timetabler schedules lessons at *time slots*. Usually, a secondary school teaches five days a week, and on each day a number of slots are available for teaching. The number of time slots per day varies from school to school and depends, among other things, on the duration of a slot. At the HLC, there are 9 time slots per day. There are breaks after the third and fifth slot. Breaks are relevant for two reasons. Firstly, double hours sometimes may not 'cross' a break. Secondly, breaks can be used to travel from one location to the other (see Section 2.1.6 about locations).

Periods

Besides the subdivision of a week into time slots, the year can also be divided. The HLC partitions a year into 5 periods. Each period, a different set of lessons may be taught. Consequently, the timetable differs each period. This format allows a bit more flexibility regarding multiplicities of lessons. For example, a school may want to give a class 2.6 math hours per week, on average (over the whole year). To achieve this, they can decide to give 3 hours a week in periods 1,2 and 5, and 2 hours a week in periods 3 and 4. It can (and will) also occur that a class has certain subjects not all year, but only in a subset of the periods. Which lesson must be taught in which period is not part of the problem, but predetermined. Therefore, timetabling can be done for each period separately, and I do not have to deal with this concept. However, for the clustering problem it *is* relevant (see Chapter 6).

Spreading Lessons

It is undesirable to have the same lesson (subject) multiple times on one day (except for double hours of course). This is because teachers prefer giving many little chunks of homework to one big task for the entire week. Generally, I want to spread the different hours of one lesson as *evenly* as possible over the days in the week. What evenly means depends on the multiplicity of the lesson. Normally this multiplicity is less then 5, so I can say that I want every time slot that a lesson is scheduled to be on a different day. There can be additional preferences like: if the multiplicity is 2, I do not want the lesson to be scheduled on consecutive days. However, these kind of additional preferences are not regarded as important at the HLC.

2.1.5 Classrooms

Almost all classrooms are meant for a certain type of lessons. The most obvious example is the gymnasium, but also there are classrooms furnished for Mathematics, languages, Physics/Chemistry etc. The HLC thinks it is important that each lesson is being given in an appropriate classroom. For this reason, the software package they use assigns the lessons not only to time slots, but also to classrooms at the same time. As I mentioned before, all teachers have a preferred classroom. However, there are many more teachers than classrooms, so usually a classroom has more than one teacher preferring it. To handle the scenario that two (or more) teachers give a lesson at the same time and preferring the same room, each classroom has a *diversion room* of the same type. The diversion scheme of rooms of the same type are in such a way that they form a cycle. Figure 2.1 shows the upper school math teachers and rooms and their relations. The three ovals represent rooms, and the three-letter-abbreviations are the teachers. There is an arrow from each teacher to his preferred classroom, and from each room to its

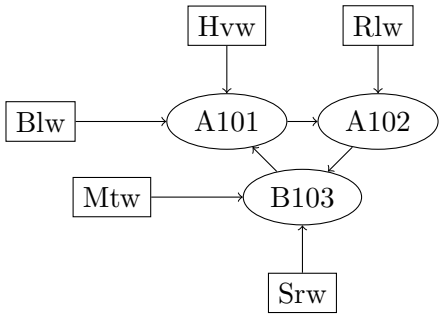


Figure 2.1: Math classrooms

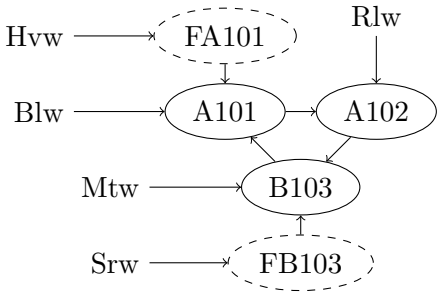


Figure 2.2: Fictitious rooms

diversion room. Using the scheme from Figure 2.1, the timetabling software behaves undesirably: if more than one teacher has the same preferred room, the teacher who comes first alphabetically takes precedence. To avoid this strange behaviour, they use fictitious rooms. These rooms do not represent real space, and a lesson that is scheduled in such a room, is automatically diverted to a real room. This trick seems to solve the problem. Figure 2.2 shows the actual scheme that is being used.

As we have seen in the figures, there are three (upper school) Math rooms. This obviously means that at most three Math lessons may be scheduled at the same time. This is a constraint that I respect in my model. I do not assign lessons to classrooms while scheduling, but only guarantee feasibility with these kind of constraints. This approach simplifies the problem. Moreover, the actual assignment of lessons to classrooms can be done as post-processing (trying to maximize the number of lessons that teachers give in their preferred room), and therefore this does not really need to be in my model.

2.1.6 Locations

In Section 2.1.5, we have seen that only upper school Math rooms are in a classroom cycle (there is a cycle for lower school Math rooms as well). Why can't they be in one big cycle? This is because the HLC has two locations: one for lower school, and one for upper school. Students always have their lessons in their own building. There are, however, teachers that teach in both lower and upper school. Because the two locations are some distance apart, a teacher can not have two consecutive lessons in two different locations: This teacher needs time to travel (about 10 minutes). Ideally, all of his lessons scheduled on one day are in the same building, but this is not always possible. If such a trip is necessary, it has to be during a break or an intermediate hour. I also want to avoid the situation that a teacher travels back and forth multiple times a day. Generally, it is desirable to schedule in such a way that every teacher has at most one trip a day. I will consider

Figure 2.3: Sliders

this as a timetable layout preference (see Section 2.2.1).

2.1.7 Other Issues

I think the concepts described above give a reasonably accurate description of the important issues of this problem. However, given the extensiveness of the software currently used and the fact that every interview I discovered new concepts, I am certain that there are still some things that I haven't addressed, but that do play a role. Nevertheless, I expect these issues to be minor, and of little influence on the scheduling process (apart from some fine-tuning). My goal is to approximate real-life timetabling as good as possible, and I am confident that I am pretty close.

2.2 Objective

Of course the timetabler wants to meet as many constraints as possible. In the software package currently used, the importance of certain constraints can be controlled by numerous sliders (see Figure 2.3). Sliding all the way to the right makes it a hard constraint (that is, the constraint has to be met), and sliding all the way to the left discards the constraint. Usually the sliders are somewhere in the middle. I expect that these sliders control the number of penalty points for a non-met constraint, but the exact relation between the sliders and the outcome is quite vague.

As we will see in the problem definition (Section 2.3), almost all constraints have to be met. There are only two types of constraints that may be violated (and thus induce penalty points): Firstly, spreading lessons evenly over a week is not always possible, and therefore considered as a soft constraint. Secondly, there are the so-called timetable layout preferences of teachers and classes.

2.2.1 Timetable Layout Preferences

Ideally, the timetabler produces a schedule that makes students and teachers as happy as possible. What exactly 'as happy as possible' means of course differs from person to person, and is mainly expressed in the timetable layout preferences of the classes and teachers, which is part of the input. These preferences usually consist of how they value (not) having a lesson at a particular time slot and to which extent they accept having intermediate hours. I call it timetable *layout* preferences because I do not allow preferences regarding specific lessons. A teacher, for instance, can not demand having class h5a on Friday, but can prefer to have at least 4 lessons on Friday. In my model, a teacher or class (the preferences of a class are of course not determined by the students themselves, but by the staff) is allowed to have the following types of preferences:

- I prefer to be off on time slot x
- I prefer to have a lesson on time slot x
- I prefer to have at least/most n lessons in location b on day x
- I prefer to have at least/most n intermediate hours on day d/per week
- I prefer to have at most n consecutive intermediate hours on day d
- I prefer to have at most n trips between locations on day d/per week

I think I am quite flexible if I allow these types, and they allow me to handle almost all real-life preferences. Like I mentioned in Section 2.1.2, part-time teachers are allowed to block parts of the week. This is rather a demand than a preference, and therefore treated as a hard constraint.

2.3 Problem Definition

Having treated all concepts that are relevant to this timetabling problem, I can summarize the input I need, the objective and constraints, and the output I get, in a problem definition:

Input

- A set of classes
- A set of teachers with their blocked time slots (if any)
- A set of lessons with for each lesson its classes set, teachers set, multiplicity
- For each class and teacher, its timetable layout preferences
- For each lesson, properties like: double hours, fixation to time slots, location and the classroom type in which it has to be
- For each classroom type, the number of rooms

Objective

- Assign lessons to time slots minimizing penalty points

Soft constraints (violating induce penalty points)

- The timetable layout preferences of teachers and classes have to be respected as much as possible
- Lessons have to be distributed as evenly as possible over a week

Hard constraints (must be met)

- Blocked time slots of teachers must be respected
- The multiplicity of lessons must be respected
- Double hours must be scheduled as such
- Double hours may not cross a break (optional)
- Fixed lessons must be scheduled at their fixed time slot
- All lessons scheduled at a particular time slot must fit in appropriate classrooms
- Teachers must have time to travel between locations

Output

- Assignment of lessons to time slots (that is, a complete timetable)

Chapter 3

Modeling

Introduction To model the timetabling problem described in Chapter 2, I use Integer Linear Programming (ILP). To be able to deal with the timetable layout preferences (see Section 2.2.1), I additionally need the technique of Column Generation (CG). For those who are unfamiliar with these techniques, I refer to Appendix B. This chapter describes how to model the timetabling problem as an Integer Linear Program: which variables and constraints are needed and how to model the objective. The issue of how to generate columns is treated in Chapter 4.

I start this chapter with a basic model, which has only the absolutely essential constraints. After that, I describe how to extend this basic model to cover all constraints that are in the problem description from Section 2.3. Finally, I state the complete model. I use consistent numbering and naming throughout this chapter. Numbered constraints correspond to those in the final model (Section 3.3). Indices are named as follows: i denotes a time slot, l a lesson, j is a teacher or class (abusing notation a bit) and c stands for a column (the concept 'column' is explained in Section 3.1.1). I will denote sets by capital letters. Combining them with indices (hopefully) gives an intuitive idea of their meaning: For example, C_j denotes the set of columns for teacher/class j.

3.1 Basic Model

In this section, I construct a basic model for the timetabling problem, in order to get the basic idea. I think this model should allow me to schedule lessons, respecting their multiplicity and minimize the deviation from the timetable layout preferences. All additional constraints like double hours, spreading, classrooms etc. are treated in Section 3.2.

3.1.1 Variables

Timetabling is about assigning lessons to time slots. Please remember from Section 2.1.3 that for each lesson, its teachers and classes set are known. It is natural to introduce a 0-1 variable for each pair of a lesson l and a time slot i:

$$y_{li} = \begin{cases} 1 & \text{if } l \text{ is scheduled at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

Since the y_{li} are variables, the = sign must be read as 'takes value'. Using these variables, I can measure how well the lessons are spread over a week. This spreading issue is not part of the basic model, and is treated in Section 3.2.

However, the quality of the solution also depends on to what extent the timetable layout preferences are respected. Of course, the values of the variables y_{li} exactly determine the timetable for each class and each teacher, but the quality of such a timetable depends on preferences regarding intermediate hours, trips between locations etc. (see Section 2.2.1) of this teacher or class. It is impossible to include all these preferences in the ILP-model, and be able to measure deviations from these preferences using only variables y_{li}. Therefore, I additionally introduce 'columns' or vectors representing timetable layouts. A timetable layout for a teacher or class can be seen as a 0-1 vector c with an element for each time slot i:

$$c_i = \begin{cases} 1 & \text{if this teacher/class has a lesson at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

Such a vector is thus not lesson specific, it only determines when there is *a* lesson and when this teacher/class is free. From Section 2.2.1 we know that I should be able to distinguish lessons in different locations in a timetable layout. With this description, I am not, but this issue will be addressed in Section 3.2.5. Note that these vectors are constants in the model.

There are many feasible columns for each teacher and class. In fact, there are so many that I can not fit all possible columns in the model. This is why I need column generation. Because the preferences for every class and teacher are known, I can introduce a cost function f_j for teacher/class j that takes a column as input, and gives the corresponding cost as output. This cost will be high if the column deviates from its preferences, and low (or zero) if this class/teacher likes this timetable layout. The cost of a column for teacher/class j can thus be written as $f_j(c)$ and because this cost is computed by the column generation algorithm, it is a constant in the ILP model. Which columns to include in the model, and how to calculate costs is treated in Chapter 4. Given some set of columns in the model, I need

0-1-variables that control which columns are chosen:

$$x_c = \begin{cases} 1 & \text{if column } c \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

3.1.2 Objective

Since spreading lessons is not included in this basic model, the only objective is to respect the timetable layout preferences as much as possible. With the cost functions f_j and variables x_c it is easy to calculate the total deviation from these preferences in the resulting timetable. Of course, I try to minimize this deviation, resulting in the following objective:

$$\min \sum_j \sum_{c \in C_j} f_j(c) x_c \quad \text{(obj)}$$

Here, C_j denotes the set of columns for teacher/class j.

3.1.3 Constraints

First of all, I have to make sure that for each teacher and class, exactly one layout pattern (column) is chosen. This means that for each j, exactly one x_c with $c \in C_j$ must take value 1 (and all other variables must take value 0). I simply add the following constraint to achieve this:

$$\sum_{c \in C_j} x_c = 1 \quad \text{for all } j \quad (2)$$

Choosing a column c for teacher or class j (that is: $x_c = 1$ for some $c \in C_j$) affects the possibilities for scheduling lessons (and vice versa). For instance, if $c_i = 0$ for this column, there may not be a lesson of teacher/class j scheduled at time slot i. On the other hand, if $c_i = 1$, there *must* be exactly one lesson of teacher/class j scheduled at time slot i. It is easy to see that these constraints connect variables x_c and y_{li} as follows:

$$c \in C_j, x_c = 1, c_i = 0 \Rightarrow \sum_{l \in L_j} y_{li} = 0$$

$$c \in C_j, x_c = 1, c_i = 1 \Rightarrow \sum_{l \in L_j} y_{li} = 1$$

Here L_j denotes the set of lessons for teacher or class j (that is, the set of lessons that have j in its teachers/classes set).

If we combine these two propositions with the fact that exactly one column is chosen for each teacher/class j, we can conclude that the value of $\sum_{l \in L_j} y_{li}$ must equal $\sum_{c \in C_j} x_c c_i$! This means that I can easily add the

following constraint to the ILP model, fixing the correspondence between the variables x_c and y_{li}:

$$\sum_{c \in C_j} x_c c_i - \sum_{l \in L_j} y_{li} = 0 \quad \text{for all } j, i \quad (1)$$

Besides the bounds on the variables x_c (constraints (2)), there is also a bound on the sum of variables y_{li} of lesson l: its multiplicity (which is part of the input). The following constraint takes care of this:

$$\sum_i y_{li} = m_l \quad \text{for all } l \quad (3)$$

Finally, I have to make sure all variables can only take values 0 or 1:

$$x_c \in \{0, 1\} \qquad \text{for all } c \qquad (9a)$$
$$y_{li} \in \{0, 1\} \qquad \text{for all } l, i \qquad (9b)$$

3.1.4 ILP-formulation

$\min \sum_j \sum_{c \in C_j} f_j(c) x_c$ (obj) Objective			
Subject to:			
$\sum_{c \in C_j} x_c c_i - \sum_{l \in L_j} y_{li} = 0$	for all j, i	(1)	Variable correspondence
$\sum_{c \in C_j} x_c = 1$	for all j	(2)	Choose one column
$\sum_i y_{li} = m_l$	for all l	(3)	Multiplicity of lessons
$x_c \in \{0, 1\}$	for all c	(9a)	Variable bounds and integrality
$y_{li} \in \{0, 1\}$	for all l, i	(9b)	

Surprisingly, with only 3 types of constraints, I have a basic ILP-model for the timetabling problem covering the most important issues. All other constraints from the problem definition are described in Section 3.2.

3.2 Adding More Constraints

3.2.1 Fixed Lessons and Blocked Time Slots

Fixing lessons to time slots can be implemented easily into the model. For each lesson l that is fixed to time slot i, I simply add the constraint:

$$y_{li} = 1 \quad (4)$$

For a teacher j who has blocked a time slot i, I could add similar constraints $y_{li} = 0$ for all $l \in L_j$. My approach, however, is to only allow to add columns c for this teacher that have $c_i = 0$. This has the same effect, and at the same time reduces the number of columns that can be added to the model. The column generation algorithm (see Chapter 4) thus takes care of this issue.

3.2.2 Double Hours

For some lessons, it is required to have a double hour, or even an n-double hour. How many double hours of what type is required for lesson l, can be expressed in an hour group vector h^l (one for each lesson). Each component of this vector represents a group of lessons (an *hour group*) that must be scheduled consecutively, and the value of such a component is the number of required consecutive lessons. A lesson l of multiplicity 3, for instance, can have $h^l = (1, 1, 1)$ (no double hours), $h^l = (2, 1)$ (one double hour, equivalent to $h^l = (1, 2)$) or $h^l = (3)$ (one long 3-double hour). The number of hour groups required for lesson l (that is, the number of components of h^l) is denoted by n_{h^l}. Moreover, we have the obvious relation $\sum_k h^l_k = m_l$.

Now that we have a formal notation, I can construct constraints that make sure these hour groups are scheduled as such. If a lesson l does not need any double hours (that is, h_l is an all-one vector), I do not need the extra constraints and variables. Denote the set of lessons that have at least one double hour by L^{DH}. The basic idea is to introduce for each hour group k of a lesson $l \in L^{\mathrm{DH}}$ a *sublesson* l^k, which must have all its hours consecutively. For each such a sublesson, I introduce 0-1-variables $y^n_{l^k i}$, defined as follows:

$$y^n_{l^k i} = \begin{cases} 1 & \text{if the } n\text{th consecutive hour of sublesson } l^k \\ & \text{is scheduled at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

The fact that there are variables for each consecutive hour n, enables me to add the following constraints that make sure the hours are indeed scheduled consecutively:

$$y^n_{l^k i} - y^{n+1}_{l^k (i+1)} = 0 \quad \text{for all } l \in L^{\mathrm{DH}}, i,\ 1 \le k \le n_{h^l},\ 1 \le n < h^l_k \quad \text{(5a)}$$

Note that for a group of size $h^l_k = 1$, there are no such constraints. Of course, an hour group may not cross a day boundary, while time slot $i + 1$ may be the first time slot of the next day. Because the length of this hour group is known, I can determine exactly at what time slots this hour group may and may not start. I can also do this, if this hour group may not cross a break. Fixing $y^1_{l^k i} = 0$ for all time slots i, where this hour group may not start, solves the problem:

$$y^1_{l^k i} = 0 \quad \text{for all } l \in L^{\mathrm{DH}}, \text{ infeasible } i,\ 1 \le k \le n_{h^l} \quad \text{(5b)}$$

Finally, the original variables y_{li} must correspond to the new variables:

$$y_{li} = \sum_{k=1}^{n_{h^l}} \sum_{n=1}^{h_k^l} y_{l^k i}^n \quad \text{for all } l \in L^{\mathrm{DH}}, i \quad \text{(5c)}$$

Note that this last constraint, also implies that the different hour groups may not overlap, because $y_{li} \leq 1$. It may seem that I have to add a lot of extra variables and constraints to the model to include double hours. However, most lessons are not in L^{DH}: In that case, I do not need all these extra variables and constraints. These constraints do not force every hour groups to be on a different day; the spreading constraints (see Section 3.2.3) are responsible for this desirable behaviour.

3.2.3 Spreading Lessons

In general, I want all the hours that a lesson is given to be on a different day. Of course, double hours are an exception to this rule. More precisely, I prefer to have all *hour groups* of a lesson on a different day. This is not always possible: the most obvious case is that there are more hour groups than days in a week. Because I model the spreading issue as a soft constraint, *preferring* to have a good spreading can do no harm. To be able to distinguish between time slots on different days, I need another piece of notation: by I_d, I denote the set of time slots that are on day d. I also need new (nonnegative) variables that measure deviation from the ideal one hour group per day:

$$z_{ld} = \begin{cases} n & \text{if } n+1 \text{ hour groups of lesson } l \text{ are scheduled on day } d \\ 0 & \text{otherwise} \end{cases}$$

I could add a hard upper bound on z_{ld} if I want to limit this deviation. Adding the following constraints for each $l \in L^{\mathrm{DH}}$ actually measures the deviation:

$$\sum_{i \in I_d} \sum_{k=1}^{n_{h^l}} y_{l^k i}^1 - z_{ld} \leq 1 \quad \text{for all } l \in L^{\mathrm{DH}}, d \quad \text{(6a)}$$

By only considering the first lesson of each hour group (by using $y_{l^k i}^1$), each group is counted only once. If l does not have any double hours, I do not have variables $y_{l^k i}^1$. In that case, I simply use the following constraints instead:

$$\sum_{i \in I_d} y_{li} - z_{ld} \leq 1 \quad \text{for all } l \notin L^{\mathrm{DH}}, d \quad \text{(6b)}$$

The last thing I have to do is to actually penalize positive values of z_{ld}, by including them in the objective. The importance of spreading lessons, compared to the importance of respecting timetable layouts, is expressed by

a weighing constant c_z. This means that every positive z_{ld} is penalized by $c_z z_{ld}$. Now I am ready to state the new objective:

$$\min \sum_j \sum_{c \in C_j} f_j(c) x_c + c_z \sum_l \sum_d z_{ld} \quad \text{(obj)}$$

3.2.4 Classrooms

At the HLC, every lesson has to be scheduled in an appropriate classroom. The classrooms can be partitioned in certain classroom types, for instance upper school Physics/Chemistry rooms, lower school languages rooms, PE rooms etc. Which lesson must be scheduled in which room type is known, just like the number of rooms per type. The set of lessons that must be scheduled in classroom group r is denoted by L^r, and the number of rooms of type r by n^r. To make sure all lessons fit in appropriate rooms at each time slot, I add the following constraints:

$$\sum_{l \in L^r} y_{li} \leq n^r \quad \text{for each } r, i \quad (7)$$

3.2.5 Locations

The only constraint from the problem definition in Section 2.3 that I have not addressed yet, is that teachers must have time to travel between locations. This, however, is not a constraint that I will add to the ILP model, but is more easily implemented in the column generation phase. For each lesson l, it is known in which location it must be. To be able to distinguish between lessons in different locations (I will use the letter b (building) for locations), I have to redefine the vectors representing columns:

$$c_{ib} = \begin{cases} 1 & \text{if this teacher/class has a lesson in location } b \text{ at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

Note that the column has become two-dimensional, making it a matrix rather than a vector. There are two locations at the HLC, so b can take values 1 and 2. Since classes have all their lessons in the same building, c_{ib} equals 0 for at least one of the bs. Since teachers (and classes) can only have one lesson at a time $\sum_b c_{ib} \leq 1$ must hold for all i. It is the responsibility of the column generation algorithm to only generate columns that have either a break or an intermediate hour between lessons that are in different locations. The timetable layout preferences of teacher j regarding the maximum number of trips is included in the cost function f_j.

To make things more clear, consider the representation of a column for one of the teachers in Table 3.1. For each of the 45 time slots (5 days, 9 time slots per day) i, the elements of the column are displayed as $c_{i1}; c_{i2}$.

	1	2	3	4	5	6	7	8	9
Mon	0;1	0;1	0;1	0;0	0;1	1;0	0;0	1;0	0;0
Tue	0;0	0;0	0;0	0;0	0;0	0;0	0;0	0;0	0;0
Wed	1;0	0;0	1;0	0;1	0;1	0;1	0;0	0;1	0;0
Thu	0;0	0;0	0;0	0;0	1;0	1;0	0;0	0;1	0;1
Fri	0;0	0;1	0;1	0;1	0;0	0;1	0;0	0;0	0;0

Table 3.1: Example column

The vertical lines after time slots 3 and 5 represent breaks. We can count that per week, this teacher has 6 lessons on location 1 (lower school), and 14 on location 2 (upper school). The fact that this column has no lessons on Tuesday, is not a coincidence: this teacher works 4 days a week, and has blocked Tuesday. On Friday, this teacher can stay in location 2, having 1 intermediate hour. On Thursday, this teacher also has an intermediate hour, but this one is necessary to travel from location 1 to location 2. On Monday and Wednesday this teacher has trips too, but he travels during a break in this case.

We can see that this column (a slight adaptation from an actually used column for teacher Ase) meets the hard trip constraints. However, I can imagine that this teacher would not be very happy with this column, since he has trips on 3 of his working days. Also, this teacher has 5 intermediate hours, which seems quite a lot (assuming this teacher prefers as few intermediate hours as possible), but is regarded as acceptable.

Since I now have a new definition for columns, I also need to adapt the constraints that guarantee correspondence between the variables x_c and y_{li}. For instance, if a lesson l that must be given in location b is scheduled at time slot i (that is, $y_{li} = 1$), then all teachers/classes that give/have this lesson must have $x_c c_{ib} = 1$. Defining L_{jb} as the set of lessons of teacher/class j that must be given in location b, this results in the following constraints:

$$\sum_{c \in C_j} x_c c_{ib} - \sum_{l \in L_{jb}} y_{li} = 0 \quad \text{for all } j, i, b \quad (1)$$

3.2.6 Day Columns

Until now, I assumed that a column represents the timetable layout for a complete week. In Section 5.2.4, we will see that this approach has some downsides compared to using *day columns*, representing timetable layouts for a single day. Of course, the column generation algorithm still has to be able to respect the timetable layout preferences, and calculate the cost of such a day column. Under the assumption that the cost of a timetable layout of a week is simply the sum of the costs of the day layouts, this is no problem. However, if we look at the types of preferences that we allow

in Section 2.2.1, we see that there are two types of preferences that are concerned with the layout of the entire week:

- I prefer to have at least/most n intermediate hours *per week*
- I prefer to have at most n trips between locations *per week*

If I use day columns, the costs associated with deviating from these two types of preferences can no longer be measured by the column generation algorithm: I have to include constraints in the ILP model to measure deviation from these preferences.

However, I first have to adapt some of the other constraints to be able to handle day columns instead of week columns. If I write C_{jd} for the set of columns of teacher/class j for day d, the constraints guaranteeing correspondence between the x_c and y_{li} variables now becomes:

$$\sum_{c \in C_{jd}} x_c c_{ib} - \sum_{l \in L_{jb}} y_{li} = 0 \quad \text{for all } j, d, i \in I_d, b \quad (1)$$

Also, I now have to choose one column for all teachers and classes *per day* (instead of one in total):

$$\sum_{c \in C_{jd}} x_c = 1 \quad \text{for all } j, d \quad (2)$$

Now I am ready to add the constraints measuring deviation from the *week* timetable layout preferences. Let us write the number of intermediate hours in day column c as $\text{int}(c)$, and the number of trips as $\text{trips}(c)$ (these are constants in the model). Other constants I need are the preferred upper bound $\hat{n}_j^{\text{int}}$ and lower bound $\check{n}_j^{\text{int}}$ on the number of intermediate hours per week plus the preferred upper bound on the number of trips per week $\hat{n}_j^{\text{trips}}$ for teacher/class j. Just like with spreading lessons, the deviations from these preferences are measured with nonnegative variables $\hat{z}_j^{\text{int}}$, $\check{z}_j^{\text{int}}$ and $\hat{z}_j^{\text{trips}}$, respectively. This leads to the following constraints:

$$\sum_{c \in C_j} x_c \text{int}(c) - \hat{z}_j^{\text{int}} \leq \hat{n}_j^{\text{int}} \qquad \text{for all } j \qquad (8a)$$

$$\sum_{c \in C_j} x_c \text{int}(c) + \check{z}_j^{\text{int}} \geq \check{n}_j^{\text{int}} \qquad \text{for all } j \qquad (8b)$$

$$\sum_{c \in C_j} x_c \text{trips}(c) - \hat{z}_j^{\text{trips}} \leq \hat{n}_j^{\text{trips}} \qquad \text{for all } j \qquad (8c)$$

I could also add hard upper bounds on the variables, to limit eventual deviation. Finally, I need to add the new variables to the objective with

corresponding weighing constants $\hat{c}^{\text{int}}$, $\check{c}^{\text{int}}$ and $\hat{c}^{\text{trips}}$. Also, the cost function now depends on the day d, and is denoted by f_{jd}. The new objective becomes:

$$\min \sum_j \sum_d \sum_{c \in C_{jd}} f_{jd}(c) x_c + c_z \sum_l \sum_d z_{ld} + \sum_j (\hat{c}^{\text{int}} \hat{z}_j^{\text{int}} + \check{c}^{\text{int}} \check{z}_j^{\text{int}} + \hat{c}^{\text{trips}} \hat{z}_j^{\text{trips}}) \quad \text{(obj)}$$

3.3 Full Model

Having treated all necessary constraints, I am ready to formulate the full ILP model:

$\min \sum_j \sum_d \sum_{c \in C_{jd}} f_{jd}(c) x_c + c_z \sum_l \sum_d z_{ld} + \sum_j (\hat{c}^{\text{int}} \hat{z}_j^{\text{int}} + \check{c}^{\text{int}} \check{z}_j^{\text{int}} + \hat{c}^{\text{trips}} \hat{z}_j^{\text{trips}})$			(obj)	Objective
Subject to:				
$\sum_{c \in C_{jd}} x_c c_{ib} - \sum_{l \in L_{jb}} y_{li}$	$= 0$	for all $j, d, i \in I_d, b$	(1)	Variable correspondence
$\sum_{c \in C_{jd}} x_c$	$= 1$	for all j, d	(2)	Choose one column per day
$\sum_i y_{li}$	$= m_l$	for all l	(3)	Multiplicity of lessons
y_{li}	$= 1$	for all l fixed to i	(4)	Fixed lessons
$y^n_{l^k i} - y^{n+1}_{l^k (i+1)}$	$= 0$	for all $l \in L^{\text{DH}}, i, 1 \le k \le n_{h^l}, 1 \le n < h^l_k$	(5a)	Double hours
$y^1_{l^k i}$	$= 0$	for all $l \in L^{\text{DH}}$, infeasible i, $1 \le k \le n_{h^l}$	(5b)	
$\sum_{k=1}^{n_{h^l}} \sum_{n=1}^{h^l_k} y^n_{l^k i}$	$= y_{li}$	for all $l \in L^{\text{DH}}, i$	(5c)	
$\sum_{i \in I_d} \sum_{k=1}^{n_{h^l}} y^1_{l^k i} - z_{ld}$	≤ 1	for all $l \in L^{\text{DH}}, d$	(6a)	Spreading
$\sum_{i \in I_d} y_{li} - z_{ld}$	≤ 1	for all $l \notin L^{\text{DH}}, d$	(6b)	
$\sum_{l \in L^r} y_{li}$	$\le n^r$	for each r, i	(7)	Classrooms
$\sum_{c \in C_j} x_c \text{int}(c) - \hat{z}_j^{\text{int}}$	$\le \hat{n}_j^{\text{int}}$	for all j	(8a)	Week layout preferences
$\sum_{c \in C_j} x_c \text{int}(c) + \check{z}_j^{\text{int}}$	$\ge \check{n}_j^{\text{int}}$	for all j	(8b)	
$\sum_{c \in C_j} x_c \text{trips}(c) - \hat{z}_j^{\text{trips}}$	$\le \hat{n}_j^{\text{trips}}$	for all j	(8c)	
x_c	$\in \{0, 1\}$	for all c	(9a)	Variable bounds and integrality
y_{li}	$\in \{0, 1\}$	for all l, i	(9b)	
$y^n_{l^k i}$	$\in \{0, 1\}$	for all $l \in L^{\text{DH}}, i, 1 \le k \le n_{h^l}, 1 \le n < h^l_k$	(9c)	
z_{ld}	≥ 0	for all l, d	(9d)	
$\check{z}_j^{\text{int}}, \hat{z}_j^{\text{int}}, \hat{z}_j^{\text{trips}}$	≥ 0	for all j	(9e)	

Chapter 4

Generating Columns

Introduction In Chapter 3, I have described the full ILP model for the timetabling problem. This model needs columns representing timetable layouts for teachers and classes as input. Because there are more feasible columns than fit in memory, I use column generation. Which columns I regard as useful, how to generate useful columns and how to calculate the corresponding constants (like its cost $f_{jd}(c)$) that are needed in the ILP model, is the subject of this chapter. Those who are unfamiliar with the column generation (CG) technique are advised to read Appendix B first.

This chapter begins with a section that explains the main ideas I use to generate good columns: We will see that we have to solve a shortest path problem in a directed graph. Section 4.2 describes how this is done for a simplified model. To be able to handle all important concepts, I need to add functionality to this basic model; this is described in Section 4.3. After this, I am ready to give a complete description of the graph that is used, and I conclude with a summary of the CG algorithm.

4.1 General Idea

4.1.1 Penalties

Each teacher and class can have its own preferences regarding the layout of its timetable. The kind of preferences I allow is formulated in Section 2.2.1. As we have seen in Section 3.2.6, the preferences concerning an entire week are handled in the ILP model; I only need to care about preferences for the layout of a single day.

Violating a preference induces penalty points. How many penalty points are induced for teacher/class j on day d are stored in a number of vectors, which are displayed in Table 4.1. A component may be 0 if the corresponding situation is preferred, I have a positive value x for undesirable situations (inducing a penalty of x), which can be even equal to ∞ if I want to exclude a situation (this has the same effect as a hard constraint). The infinity

Vector	Components	Description:
		penalty for having...
p_{jd}^{L}	$(p_{jd}^{\mathrm{L}})_i$	... a lesson at time slot $i \in I_d$
p_{jd}^{NL}	$(p_{jd}^{\mathrm{NL}})_i$	... no lesson at time slot $i \in I_d$
$p_{jd}^{\#\mathrm{L}}$	$(p_{jd}^{\#\mathrm{L}})_n$	... n lessons
$p_{jd}^{\#\mathrm{IH}}$	$(p_{jd}^{\#\mathrm{IH}})_n$	... n intermediate hours
$p_{jd}^{\#\mathrm{CIH}}$	$(p_{jd}^{\#\mathrm{CIH}})_n$	... n consecutive intermediate hours
$p_{jd}^{\#\mathrm{TR}}$	$(p_{jd}^{\#\mathrm{TR}})_n$	... n trips

Table 4.1: Penalty vectors for teacher/class j and day d

sign stands for a number that is so large that such a situation will never be generated. Using these vectors, all timetable layout preferences from Section 2.2.1 can be stored. For example, if a teacher j prefers to have 3 to 5 lessons on day d, but definitely no more than 6, he could have penalty vector $p_{jd}^{\#\mathrm{L}} = (4, 2, 1, 0, 0, 0, 2, \infty, \infty, \ldots)$. Also, the concept of blocked time slots for part-time teachers can be included: suppose teacher j has blocked the first 3 time slots of day d, and really prefers to start teaching at time slot 5, his penalty vector p_{jd}^{L} would look like: $p_{jd}^{\mathrm{L}} = (\infty, \infty, \infty, 4, 0, 0, \ldots)$.

4.1.2 Costs

These penalty vectors together determine the cost functions f_{jd} for every teacher and class j and day d: As we will see in this chapter, the column generation algorithm uses these vectors to compute the cost $f_{jd}(c)$ of a generated column c, and passes it as a constant to the ILP model. The objective of the CG algorithm could be to find a column minimizing this cost. This is useful, for instance, if I want to generate for each teacher and class j a number of columns with low cost. Note that, besides the cost $f_{jd}(c)$, it is also easy to determine the other constants needed by the ILP model: $\mathrm{int}(c)$ and $\mathrm{trips}(c)$.

From the theory of CG (see for instance [1], Chapter 6), the CG algorithm must also be able to generate new columns that could improve the quality of the solution of the LP relaxation of the ILP model (for the current set of columns), *given an optimal solution of this LP relaxation.* To this end, I need the concept of *reduced cost*: adding a column c can improve the quality of the solution (of the LP relaxation) only if the reduced cost of the corresponding variable x_c has negative reduced cost. In this case, the reduced cost of a variable x_c for column c for teacher/class j and day d is defined by:

$$f_{jd}^{\mathrm{red}}(c) := f_{jd}(c) - \sum_b \sum_{i \in I_d} \pi_{ib}^{1jd} c_{ib} - \lambda_{jd}^2 - \mathrm{int}(c)(\lambda_j^{8\mathrm{a}} + \lambda_j^{8\mathrm{b}}) - \mathrm{trips}(c)\lambda_j^{8\mathrm{c}}$$

In this definition, π^{1jd} is the matrix of dual multipliers of constraints (1) for this j and d, λ^2_{jd} is the corresponding dual multiplier of constraint (2), and $\lambda^{8a}_j, \lambda^{8b}_j$ and λ^{8c}_j are the dual multipliers for constraints (8) of this teacher/class j. These dual multipliers are part of the input of the CG algorithm and follow from the solution of the LP relaxation (with the current set of columns).

4.1.3 Objective and Directed Graph

We know that a column corresponds to a set of values c_{ib} for a teacher/class j and a day d. Note that this pair (j, d) is fixed, and that I run the column generation algorithm for every such pair. The reduced cost of such a column c, given the penalty vectors of the fixed pair (j, d) and the dual multipliers, depends on how lesson hours and free hours succeed each other (for lesson hours it is also important in which building they take place). This means that I have to decide, for each time slot i, whether one of the values c_{ib} becomes equal to 1 (a lesson in location b), or if they are all zero (a free time slot i).

A natural way of doing this is by constructing a graph, with for every time slot i a number of nodes that I can choose from (lesson nodes, intermediate hour nodes etc.). These nodes together form *layer* i in the graph. I add arcs from a node u in layer i to a node v in the next layer (time slot) $i + 1$ if this node v can be the successor of u. Finally if I add a source r before the first layer (with arcs from r to all nodes in layer 1) and a sink s after the last layer (and arcs to s from all nodes in the last layer), I have that each path from r to s automatically corresponds to an assignment of values to the c_{ib}.

The objective is to choose a path (column) with minimal reduced cost. The advantage of above graph construction is that if I correctly assign costs to the arcs and/or the nodes, the *length* of each path from r to s equals the reduced cost of the corresponding column. This means that I have transformed the problem of finding a column with minimal reduced cost to a simple shortest path problem (see Section B.3). I have chosen to only assign costs to the nodes, although I realize that assigning them to arcs could be useful if the costs explicitly depend on the order of the chosen nodes. However, we will see that I do not really need to do this.

When I have found a shortest path, I have to decide whether the corresponding column is good enough. If I am minimizing reduced cost, it is required to be negative, as only adding a column with negative reduced cost to the current LP relaxation could improve the solution. If the reduced cost is nonnegative, I do not add it to the model, and if this is the case for all pairs (j, d), I know that I have an optimal solution of the LP relaxation. If I simply want columns with low cost $f_{jd}(c)$ (for instance, to construct a set of columns for each teacher and class to *start* solving of the LP relaxation

with), I can fix all dual multipliers to zero, and still minimize *reduced* cost (so I do not need to adapt my algorithm). In this case the cost does not need to be negative, of course.

State Vector

The cost of a node not only depends on its type (lesson, intermediate hour etc.), but also on what happened before reaching this node. For example, a lesson node in layer i induces a penalty $(p_{jd}^{\mathrm{L}})_i$, but may also induce a penalty $(p_{jd}^{\#\mathrm{L}})_n$ if it is the nth lesson. This means that somehow, I have to count the number of lessons so far. To this end, I introduce a *state vector* describing the current state (one component v^l of this vector thus counts the number of lessons so far). In this case, this means that I must have several copies of this lesson node in layer i, each with a different state vector: one with $v^l = 1$, one with $v^l = 2$ etc.

The arcs in the graph have to make sure that this state vector always correctly represents the current state. For instance a lesson node in layer i with state $v^l = 2$ may have an arc to a lesson node in layer $i+1$ with $v^l = 3$, but there is of course no arc to a lesson node with $v^l = 4$. To be able to deal with all the different kinds of penalties from Table 4.1, I must not only count the number of lessons v^l, but also the number of intermediate hours v^{int} and the number of trips v^{trip}. This means that there can be a lot of different state vectors and therefore a lot of nodes in the graph: assuming 9 layers, this number of nodes could easily exceed 1000.

4.2 Basic Model

Now that we have a general idea about how to solve this problem, I can describe the directed graph that I need in more detail. In this section, I will describe a simplified version that illustrates how such a graph is constructed, but that is not too comprehensive to get distracted from the main idea. In this basic model, I assume that penalty vectors $p_{jd}^{\#\mathrm{IH}}$ and $p_{jd}^{\#\mathrm{TR}}$ are all zero. Moreover, there will be no trips; all lessons are in location 1. Since I do not need to count the total number of intermediate hours (since $p_{jd}^{\#\mathrm{IH}}$ is all zero) and trips (because $p_{jd}^{\#\mathrm{TR}}$ is all zero), under these assumptions, the state vector of each node has to have only one component: the number of lessons (in location 1). I also assume that the dual multipliers $\lambda_j^{8\mathrm{a}}$, $\lambda_j^{8\mathrm{b}}$ and $\lambda_j^{8\mathrm{c}}$ equal 0, so now the the reduced that I wish to minimize becomes:

$$f_{jd}^{\mathrm{red}}(c) := f_{jd}(c) - \sum_{i \in I_d} \pi_{i1}^{1jd} c_{i1} - \lambda_{jd}^{2}$$

4.2.1 Types of Nodes

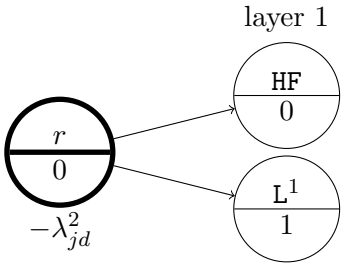
There are 6 types of nodes that I have to consider in the basic model:

1. A source r
2. Lesson nodes $\mathtt{L}^b$ for a lesson in location b
3. Intermediate hour nodes $\mathtt{IH}^m$ for the mth consecutive intermediate hour
4. Heading (before the first lesson of the day) free hour nodes `HF`
5. Tailing (after the last lesson of the day) free hour nodes `TF`
6. A sink s

For each node, I will explain its cost and which successors (nodes in the next layer this node has an arc pointing to) it has. To this end, I use little diagrams (see for example 4.2.2). The thick circle represents the node that is being described: the upper half contains its type, the lower half its state vector. Since only component v^l (the number of lessons) is of interest in the basic case, I only display the value of this component in the lower half of each node. The cost is displayed underneath the described node. There are arcs to its successors, which are displayed without cost. I will assume in the description of the different node types that layer i is not the last layer.

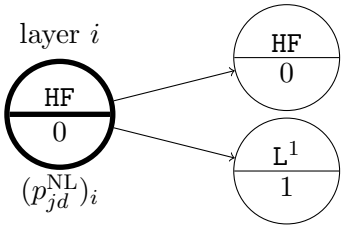
4.2.2 Source

There are only two nodes in layer 1: the successors of the source. There can be either a lesson, or a heading free hour. The cost of the source is the term $-\lambda^2_{jd}$ that must always be part of the cost of any path. Since the source is in any path, it seems like a good place to put this cost.



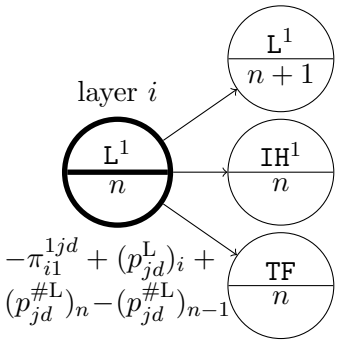
4.2.3 Heading Free Hour

A heading free hour can be followed by either another heading free hour, or the first lesson of the day. The only costs associated with a heading free hour are those for having no lesson at time slot i: $(p^{\mathrm{NL}}_{jd})_i$.



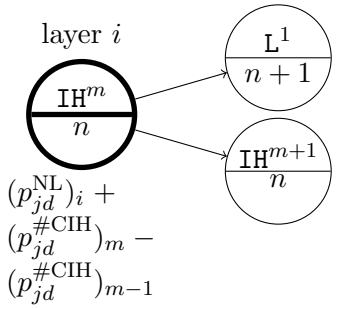
4.2.4 Lesson

The cost of this lesson node looks quite complicated. Because c_{i1} will be equal to one if this node is in the final column, I have to add the term $-\pi_{i1}^{1jd}$ to the cost. Moreover I have the penalty $(p_{jd}^{\mathrm{L}})_i$ for having a lesson at this time slot i. I also have to include the penalty for having n lessons: $(p_{jd}^{\#\mathrm{L}})_n$. However, a path to this node must have visited $n-1$ lesson nodes before. To avoid cumulating these costs, I have to subtract the penalty $(p_{jd}^{\#\mathrm{L}})_{n-1}$ from the previous (the $(n-1)$th) lesson. Note that I only have to do this for $n > 1$. A lesson can be followed by either another lesson (updating the value of v^l to $n+1$), an intermediate hour (the first one in row) or a tailing free hour.



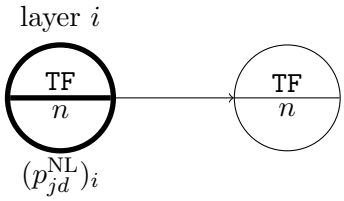
4.2.5 Intermediate Hour

An intermediate hour has only two successors: it is either followed by another intermediate hour, or by a lesson. There are two types of costs: a penalty for having no lesson at this time slot, and a penalty for having m consecutive intermediate hours. Just like with lessons, I avoid cumulating costs by subtracting $(p_{jd}^{\#\mathrm{CIH}})_{m-1}$ (if $m > 1$).



4.2.6 Tailing Free Hour

A tailing free hour can only be followed by another tailing free hour. Just like a heading free hour the only penalty induced is $(p_{jd}^{\mathrm{NL}})_i$.

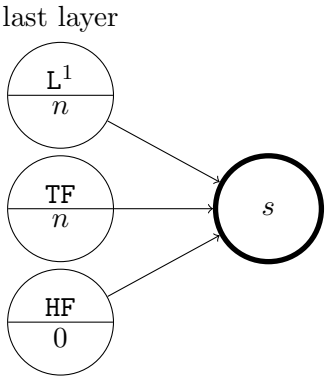


layer i	node	n lessons	cost	c_{i1}
	r	0	$-\lambda^2_{jd}$	
1	HF	0	$(p^{\mathrm{NL}}_{jd})_1$	0
2	$\mathtt{L}^1$	1	$-\pi^{1jd}_{21} + (p^{\mathrm{L}}_{jd})_2 + (p^{\#\mathrm{L}}_{jd})_1$	1
3	$\mathtt{IH}^1$	1	$(p^{\mathrm{NL}}_{jd})_3 + (p^{\#\mathrm{CIH}}_{jd})_1$	0
4	$\mathtt{L}^1$	2	$-\pi^{1jd}_{41} + (p^{\mathrm{L}}_{jd})_4 + (p^{\#\mathrm{L}}_{jd})_2 - (p^{\#\mathrm{L}}_{jd})_1$	1
5	$\mathtt{L}^1$	3	$-\pi^{1jd}_{51} + (p^{\mathrm{L}}_{jd})_5 + (p^{\#\mathrm{L}}_{jd})_3 - (p^{\#\mathrm{L}}_{jd})_2$	1
6	$\mathtt{IH}^1$	3	$(p^{\mathrm{NL}}_{jd})_6 + (p^{\#\mathrm{CIH}}_{jd})_1$	0
7	$\mathtt{IH}^2$	3	$(p^{\mathrm{NL}}_{jd})_7 + (p^{\#\mathrm{CIH}}_{jd})_2 - (p^{\#\mathrm{CIH}}_{jd})_1$	0
8	$\mathtt{L}^1$	4	$-\pi^{1jd}_{81} + (p^{\mathrm{L}}_{jd})_8 + (p^{\#\mathrm{L}}_{jd})_4 - (p^{\#\mathrm{L}}_{jd})_3$	1
9	TF	4	$(p^{\mathrm{NL}}_{jd})_9$	0
	s			

Table 4.2: Example path

4.2.7 Sink

Of course, a sink does not have successors, but only predecessors. The sink can be preceded by a tailing free hour or a lesson (both with several possible state vectors, only one possibility (n) is drawn in the diagram). It can even be preceded by a heading free hour to allow a path without any lessons. Since there is only one sink, independent of the state, the sink has no state vector. Moreover, there is no cost associated with the sink.



4.2.8 Example Graph and Path

In Figure 4.1, we see a graph with all nine layers. I included this figure to illustrate how such a graph looks like. For typographical reasons, I limited the scope of the graph to cover at most 5 lessons (all nodes with a state vector of $n > 5$ are not displayed) and at most 2 consecutive intermediate hours (nodes $\mathtt{IH}^m$ with $m > 2$ are excluded). Every path from r to s in this graph corresponds to a column, and the length of such a path equals the reduced cost of the corresponding column. To illustrate this concept, consider the path we get by following the thickened arcs (let us assume this is the minimum length path). The corresponding column and cost are in Table 4.2.

We see that this path corresponds to a column with 4 lessons and 3 intermediate hours (one single intermediate hour at time slot 3 plus two

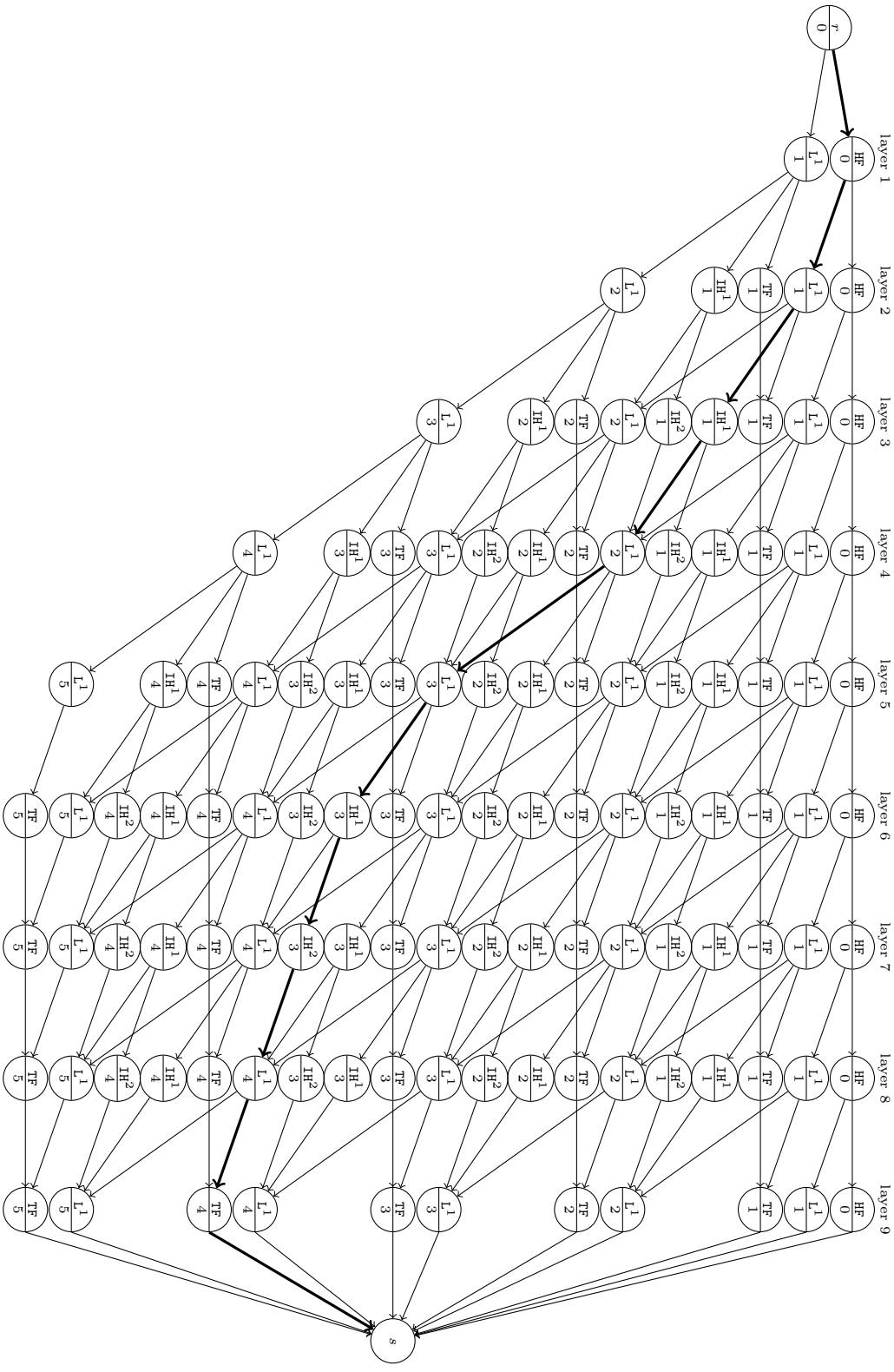


Figure 4.1: Example graph with at most 5 lessons and at most 2 consecutive intermediate hours

consecutive ones at slots 6,7). If we add all individual costs, we get:

$$-\lambda^2_{jd}+\sum_{i\in I_d}\Big[c_{i1}((p^{\mathrm{L}}_{jd})_i-\pi^{1jd}_{i1})+(1-c_{i1})(p^{\mathrm{NL}}_{jd})_i\Big]+(p^{\#\mathrm{L}}_{jd})_4+(p^{\#\mathrm{CIH}}_{jd})_1+(p^{\#\mathrm{CIH}}_{jd})_2$$

A quick check tells us that this is indeed the correct reduced cost $f^{\mathrm{red}}_{jd}(c)$ of this column. Without the terms involving dual multipliers, the above expression gives the cost $f_{jd}(c)$ of this column. This cost is needed if I add the columns to the LP relaxations (and of course for the ultimate ILP model): it serves as coefficient of the variable x_c in the objective. Moreover, the model needs constants $\mathrm{int}(c) = 3$ and $\mathrm{trips}(c) = 0$ and the values c_{ib}, which can be determined easily.

4.3 Adding Functionality

4.3.1 Fixed Lessons

If this teacher or class has a lesson l in location b that is fixed to time slot i, I must only generate columns that have $c_{ib} = 1$ (all other columns are useless). To force this, I simply remove all except the $\mathtt{L}^b$ nodes from layer i. This way, all paths from source to sink are forced to go through a node $\mathtt{L}^b$ in layer i, and all resulting columns have $c_{ib} = 1$.

4.3.2 Blocked Time Slots

If a teacher has blocked some time slots, the corresponding $(p^{\mathrm{L}}_{jd})_i$ are set to ∞, to avoid generating columns with lessons at these time slots. Most of the time, the blocked hours are the first- or the last few hours of a day. In that case the shortest path algorithm gives a path that visits nodes `HF` or `TF`, respectively. There are some teachers that have blocked time slots in the middle of a day. It then would be possible to have lessons before- and after these blocked time slots, and the shortest path algorithm would be forced to visit intermediate hour nodes `IH` at these blocked slots, inducing a lot of cost.

To avoid counting these blocked time slots as costly intermediate hours, I simply remove all nodes in the corresponding layers from the graph. Of course, to repair connectedness in the graph, we have to add arcs from the last layer before the arisen gap to the first layer after the gap in the obvious way (just pretend the corresponding time slots do not exist). Since the new graph has less layers than there are time slots, we have to be careful in translating the shortest path back to values c_{ib}: of course, we set $c_{ib} = 0$ for every blocked time slot i.

4.3.3 Trips

It is impossible to have two consecutive lessons in different buildings, except when there is a break between them. In the basic model from Section 4.2, this was not an issue, since all lessons were in the same location. When there are lessons in both locations, I have to be careful in the construction of the graph, to avoid a node $\texttt{L}^1$ to be immediately followed by $\texttt{L}^2$ (or vice versa), if there is no break between them. To this end, I introduce an extra component v^{fl} ('fl' for feasible location) in the state vector controlling which type of lesson nodes may follow the current node:

$$v^{\text{fl}} = \begin{cases} 0 & \text{if this node may not be followed by any lesson node} \\ 1 & \text{if this node may only be followed by a lesson node } \texttt{L}^1 \\ 2 & \text{if this node may only be followed by a lesson node } \texttt{L}^2 \\ 3 & \text{if this node may be followed by lesson nodes } \texttt{L}^1 \text{ and } \texttt{L}^2 \end{cases}$$

The value $v^{\text{fl}} = 0$ applies only to nodes TF, and $v^{\text{fl}} = 3$ is only the case for the source r and nodes HF. Lesson nodes $\texttt{L}^b$ have $v^{\text{fl}} = b$ and intermediate hour nodes have the same v_b as their predecessor(s). There is only an arc to lesson node $\texttt{L}^b$ if its predecessor has $v^{\text{fl}} = b$ or $v^{\text{fl}} = 3$.

To be able to count trips, and calculate the cost induced by them, I have to introduce another kind of node: trip nodes $\texttt{TR}^b$. Here b denotes the location where the trip goes to. I also need these nodes to make trips possible: in the above setup, the first lesson node fixes v^{fl}, and all paths from this node to the sink have this same v^{fl} (or 0 in case of TF nodes). A trip node $\texttt{TR}^b$ changes the v^{fl} of its predecessor into b, forcing a trip to this location b. The trip nodes must always directly follow a lesson node. However, they are not in the next layer, but *between* layers. This is because a trip may be during a break between time slots. Moreover, if a trip must be made during an intermediate hour, I still want to be able to count the intermediate hour in the next layer.

Figure 4.2 shows how the trip nodes fit in the graph. Only the value of v^{fl} is displayed in the bottom half of each node and only trips from location 1 to location 2 are displayed. Of course, the $\texttt{L}^1$ nodes have other successors besides $\texttt{TR}^2$ (these are in layer $i + 1$, see Section 4.2.4). If there is no break between time slots i and $i + 1$, an intermediate hour node must follow the trip node. However, this node $\texttt{IH}^1$ has $v^{\text{fl}} = 2$, so the first to come lesson will be in location 2. If there is a break, the trip node may be followed immediately by a lesson in location 2. The cost of the trip nodes will be discussed in Section 4.4.

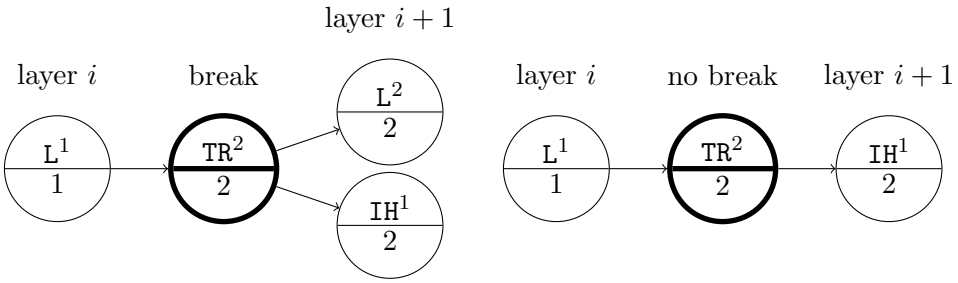


Figure 4.2: TR nodes

4.4 Complete Graph Description

Now that I have covered all different kinds of nodes, I can give a complete description of the state vector and the cost for each node type. All components of the state vector that I need are listed in Table 4.3. Note that I also keep track of the number of lessons *per location* v^{l^b}, although the cost only depends on the total number of lessons v^{l}. Why I do this is explained in Section 5.2.2. The different types of nodes with their costs are in Table 4.4.

Finally, all nodes with their successors are listed in Table 4.5. There are some remarks to be made. Between braces are the components of the state vector that must change in the successor. The source has an all-zero state vector, except for component $v^{\mathrm{fl}} = 3$. Successor $\mathtt{TR}^{b'}$ of $\mathtt{L}^b$ has $b' \neq b$. Finally, nodes HF, $\mathtt{L}^b$ and TF in the last layer have the sink s as only successor.

Component	Description
v^{l^b}	number of lessons in location b so far
v^{l}	number of lessons so far ($\sum_b v^{\mathrm{l}^b}$)
v^{int}	number of intermediate hours so far
v^{trip}	number of trips so far
v^{fl}	see Section 4.3.3

Table 4.3: Components of state vector v

Node	Cost
r	$-\lambda^2_{jd}$
$\mathtt{HF}$	$(p^{\mathrm{NL}}_{jd})_i$
$\mathtt{L}^b$	$-\pi^{1jd}_{ib} + (p^{\mathrm{L}}_{jd})_i + (p^{\#\mathrm{L}}_{jd})_{v^{\mathrm{l}}} - (p^{\#\mathrm{L}}_{jd})_{v^{\mathrm{l}}-1}$
$\mathtt{IH}^m$	$-(\lambda^{8\mathrm{a}}_j + \lambda^{8\mathrm{b}}_j) + (p^{\mathrm{NL}}_{jd})_i + (p^{\#\mathrm{CIH}}_{jd})_m - (p^{\#\mathrm{CIH}}_{jd})_{m-1} \dots$ $\dots + (p^{\#\mathrm{IH}}_{jd})_{v^{\mathrm{int}}} - (p^{\#\mathrm{IH}}_{jd})_{v^{\mathrm{int}}-1}$
$\mathtt{TR}^b$	$-\lambda^{8\mathrm{c}}_j + (p^{\#\mathrm{TR}}_{jd})_{v^{\mathrm{trip}}} - (p^{\#\mathrm{TR}}_{jd})_{v^{\mathrm{trip}}-1}$
$\mathtt{TF}$	$(p^{\mathrm{NL}}_{jd})_i$
s	0

Table 4.4: Node types with costs

Node	Successors
r	$\mathtt{HF}$, $\mathtt{L}^b$ ($v^{\mathrm{l}^b} = 1$, $v^{\mathrm{l}} = 1$, $v^{\mathrm{fl}} = b$)
$\mathtt{HF}$	$\mathtt{HF}$, $\mathtt{L}^b$ ($v^{\mathrm{l}^b} = 1$, $v^{\mathrm{l}} = 1$, $v^{\mathrm{fl}} = b$)
$\mathtt{L}^b$	$\mathtt{L}^b$ ($v^{\mathrm{l}^b} + 1$, $v^{\mathrm{l}} + 1$), $\mathtt{IH}^1$ ($v^{\mathrm{int}} + 1$), $\mathtt{TR}^{b'}$ ($v^{\mathrm{trip}} + 1$, $v^{\mathrm{fl}} = b'$), $\mathtt{TF}$ ($v^{\mathrm{fl}} = 0$)
$\mathtt{IH}^m$	$\mathtt{L}^b$ ($v^{\mathrm{l}^b} + 1$, $v^{\mathrm{l}} + 1$), $\mathtt{IH}^{m+1}$ ($v^{\mathrm{int}} + 1$)
$\mathtt{TR}^b$	$\mathtt{IH}^1$ ($v^{\mathrm{int}} + 1$), if $\mathtt{TR}^b$ in break: $\mathtt{L}^b$ ($v^{\mathrm{l}^b} + 1$, $v^{\mathrm{l}} + 1$)
$\mathtt{TF}$	$\mathtt{TF}$

Table 4.5: Node types with successors

4.5 Algorithm Summary

As a summary, I list what the column generation algorithm does, together with its input and output.

Input

- The penalty vectors of teacher j for day d
- The dual multipliers from the optimal solution of the LP relaxation of the ILP model

---

Objective

- Find a column with minimal reduced cost $f_{jd}^{\text{red}}(c)$

Method

- Construct a directed graph
- Assign costs to nodes of this graph using the penalty vectors and dual multipliers
- Find the shortest path in this graph
- Translate this shortest path to a column, and determine all needed constants

---

Output

- The column: values c_{ib}
- The cost $f_{jd}(c)$
- Constants $\text{int}(c)$ and $\text{trips}(c)$

Chapter 5

Implementation Matters

Chapter 3 and 4 describe the theoretical model and algorithms for the timetabling problem. When I tried to implement these models in my own timetabling program, I observed that there were several things that could be improved. These improvements apply to both the running time and the quality of the solution. This chapter treats these matters: Section 5.1 is about improving the solving of the ILP timetabling model and Section 5.2 treats several ways to improve the column generation. How well these improvements work is treated in Chapter 11, presenting the computational results (running times, solution qualities etc.).

5.1 ILP Timetabling

5.1.1 Solving in Stages

To test my timetabling program, I tried to schedule lessons from a small subset of the classes of the HLC, for instance only the first-year classes. Using this approach, the program produced solutions very quickly, so I did not have to wait very long to check whether my program worked properly. When I confirmed that everything went well, I tried to schedule the complete school, with all its lessons from all classes. Unfortunately, this was too much to ask: after running it for a week, my program did not even find a *feasible* solution, let alone a *good* solution. I made sure that there was at least one feasible solution by adding all columns corresponding to the schedule that is currently being used at the HLC, however, my program did not find this solution within a reasonable amount of time.

Maybe the model became too big with all classes, teachers, lessons and columns, causing the solver to become too slow. Adding fewer columns is generally not a good idea, as it reduces the chance of finding a combination of columns that allow a feasible solution. Therefore, I decided to reduce the problem size by decomposing the problem by partitioning the set of lesson. The current timetablers also use this approach: I choose a subset of the

classes for which I schedule all lessons, and while fixing the lessons from this partial schedule, try to schedule the next subset of classes. I repeat this process until all classes have been scheduled.

Of course, such a decomposition generally reduces the quality of the final solution, because fixing lessons from partial solutions is basically adding constraints to the model. Deciding how to decompose the classes is important, as well as the order in which I solve the partial problems. First of all, I can not have too many stages since more stages means more quality loss. Also, having too few stages may result in too little improvement of the running time. This is a trade-off I have to make. The order in which I solve the stages is also important for the quality of the final solution, and sometimes even crucial: it may happen that some partial problem becomes infeasible, because of the fixed lessons of a previously solved stage.

To illustrate this infeasibility issue, consider a one hour teacher meeting with all Math teachers. Because of the blocked time slots of some of the teachers, this meeting can only take place on Friday, hour 5,6,7 or 8 (these are the only time slots that all Math teachers are available). If this meeting is not in the first stage, it may happen that teacher A, for example is scheduled to teach a class on Friday, hour 6 and 7, and teacher B on Friday, hour 5 and 8. Now if I fix these lessons, it becomes impossible to schedule the meeting in subsequent stage. Therefore it is wise to start with 'difficult' lessons in early stages. Difficult lessons are lessons with large teachers or classes sets, such as big cluster patterns or meetings. If I still discover a lesson that is impossible to schedule in some stage (that is, its multiplicity is larger than the number of time slots it can be scheduled to), I can unfix the lessons that cause this blockade. Checking for infeasible lessons and unfixing eventual blocking lessons can be done before the start of each stage.

Finally, it may be beneficial, for solving an early stage, to adapt the cost functions of teachers who have a lot of lessons that will be scheduled in some subsequent stage: for example, an intermediate hour in this early stage is probably not so bad because it can be filled in a subsequent stage.

5.1.2 Empty Cluster Patterns

In upper school, the majority of lessons are cluster patterns. Some cluster patterns are well filled (most students are assigned to one of the groups in this cluster pattern, let us call these lessons 'full'), and some are not (the so-called empty lessons). In the timetabling algorithm, no distinction between full and empty lessons is made, while this could be important: it is desirable to have empty lessons at the beginning or end of the day, as they would cause an intermediate hour for a lot of students if they would be scheduled in the middle of a day. Such an intermediate hour is not recognized nor penalized in the algorithm: a lesson is a lesson, whether empty or full.

To make the algorithm preferring to schedule empty lessons at the be-

ginning or end of a day, I can use a simple trick. For each upper school class j, I add a fictitious extra class $\tilde{j}$ to the model. I add this class to the classes set of all whole-class lessons of class j, and to the classes set of all full lessons. In a solution of the timetabling problem, such a class $\tilde{j}$ thus has the exact same schedule as class j, except for the empty lessons: they do not appear in the schedule of $\tilde{j}$. This means that scheduling such an empty lesson in the middle of a day, produces an intermediate hour for class $\tilde{j}$. If I penalize intermediate hours for this class, such a schedule becomes undesirable, and the algorithm automatically tries to avoid this situation.

I could extend this approach by making a finer partition of the lessons (e.g. dividing into three classes: empty-, half-empty- and full lessons). This of course would require more fictitious classes, but potentially increases the quality of the solution. I suggest scaling the penalties of the fictitious classes according to the number of students involved. The timetabling program at the HLC uses the finest partition one can think of: it makes schedules for each individual student. Although this is theoretically possible in the model (add a fictitious class $\tilde{j}$ for each student having only the lessons the corresponding student is assigned to), this approach would make the model far too big and way too slow.

Another option is to include this distinction between full and empty lessons in the columns. I then would have entries $c_{ibf} = 1$ if a full lesson is to be scheduled at time slot i and $c_{ibe} = 1$ for an empty lesson. Just like with the different locations, I then would have to add extra constraints (1) to the ILP model in order to make the variables y_{li} correspond to the columns. The column generation algorithm would have to be adapted to penalize situations where an empty lesson is neither the first, nor the last lesson of the day. Since the above approach works well, I have not implemented this idea; let us regard it as further research (see Section 12.3).

5.2 Column Generation

5.2.1 Using Additional Information

As we have seen in Section 4.1.1, I can force the column generation algorithm to only generate columns that do or do not have a lesson time slot i by removing infeasible nodes from layer i. This is useful in case of blocked time slots or fixed lessons. Sometimes, I can use some other information that also limits the columns that may be generated. For example, if a teacher only teaches VMBO classes, which must always have the 9th hour off, it is useless to generate columns for this teacher, that have a lesson at time slot 9.

To make use of this kind of information, some data that could be useful for the column generation phase must be stored, before the actual solving starts:

- First of all, I determine for each lesson l, at which time slot it could be scheduled. Lesson l could be scheduled at time slot i only if all teachers and classes of this lesson are available at this time slot: They may not have blocked this time slot or have an other lesson that is fixed at time slot i.

- If the number of time slots that lesson l can be scheduled to equals the multiplicity of l, I can fix l to these time slots. This means that I remove all except $\mathtt{L}^b$ nodes (assuming lesson l is given in location b) from the layers corresponding to these fixed time slots. Of course, if the number of feasible time slots is less than the multiplicity, the model is infeasible.

- Now for each class and teacher j, for each time slot i and for each location b, I determine whether there is at least one of its lessons in location b that can be scheduled at time slot i, using the stored information of the lessons. If there is no such lesson, I can remove the useless nodes $\mathtt{L}^b$ from layer i.

Using the extra information, prevents me from generating useless columns.

5.2.2 Constrained State Vector

If I want to generate a column for day d for teacher or class j, I do not know in advance how many lessons there must be in this column (only the total number of lessons per week is known). This number of lessons of the resulting column with minimal reduced cost depends on the penalty vectors (in particular $p_{jd}^{\#\mathrm{L}}$) and the dual multipliers. To introduce some more flexibility for the ILP solver to choose columns, making it easier to form combinations of day columns that have the required number of lessons per week, I decided to generate not only the best day column, but also good columns with a different number of lessons.

For example, if class j must have 5 to 7 lessons on day d, I generate the best column with 5 lessons, the best column with 6 lessons and the best column with 7 lessons. This results in adding 3 columns instead of one, if all three have negative reduced cost. For teachers, I can also introduce diversity in lessons in different locations: I could, for example, generate the best column that has 3 lessons in location 1 and 4 in location 2, plus the best column that has 4 lessons in location 1 and 3 in location 2 etcetera: For all possible partitions of the lessons over the two locations, I generate a column.

How can I constrain the number of lessons (of a certain type)? This can be done by constraining the arcs from nodes in the last layer to the sink: I only allow arcs from nodes in the last layer that have a state vector that corresponds to the constraint (all other arcs to sink s are removed from the

graph). For example, if I want a column with 3 lessons in location 1 and 4 in location 2, I remove all arcs from nodes in the last layer that have $v^{l1} \neq 3$ or $v^{l2} \neq 4$, to the sink.

This trick is especially useful if I generate for each teacher and class a set of columns to start with (before actually start solving). At that point I do not have dual multipliers yet, so I try to minimize $f_{jd}(c)$ (that does not have to be negative, of course). This produces column sets with a lot of diversity, and provides a good starting point for the solver.

5.2.3 Neighbor Columns

The fact that columns have low reduced cost, means that they are useful in the LP relaxation, but does not necessarily mean that they are useful in the final ILP model. For instance, in the LP relaxation it could be the case that a lot of columns were generated for some teacher j that have a lesson on some time slot i: apparently this lesson at time slot i was useful in the LP relaxation. In the ILP model, however, the constraints could force (because they require integer variables) that there can not be a lesson for this teacher at time slot i. This means that a lot of generated columns become useless, and may even cause an infeasible ILP model.

Therefore, using an idea I got from Guido Diepen [2], every time a column c is generated, I also generate a set of neighbors of this column c. These neighbors are also columns that are generated based on column c, but that differ from c in at least one time slot. These columns are not added to the LP relaxation (they also do not need to have negative reduced cost), but stored in a column pool. The unique columns from this pool are ultimately added to the ILP model.

These neighbor columns of column c are generated as follows: for every component c_{ib}, if it equals 1, I temporarily take out the lesson nodes $\mathtt{L}^b$ from layer i, generating a neighbor column with $c_{ib} = 0$. If $c_{ib} = 0$, I do the opposite: I remove all nodes from layer i except the lesson nodes $\mathtt{L}^b$, generating a neighbor column with $c_{ib} = 1$. This way, I again introduce more diversity, and thus more flexibility in the ILP model.

5.2.4 Day Columns versus Week Columns

As I mentioned in Section 3.2.6, I use day columns instead of week columns. Initially, I used week columns, but then encountered some problems connected with this choice: I observed that the optimal solutions of the LP relaxation were highly degenerate. Consequently, there were a lot of columns that had negative cost (and thus were generated), but did not improve the quality of the solution. Therefore, in the LP phase, the solver kept adding columns, and it took many iterations (too many, for large problem instances) before there were no columns with negative reduced cost to add.

I considered using the technique of stabilized column generation (see [3]) to overcome this degeneracy. However, there is an even better way to avoid needing many iterations: use day columns. This approach reduces the total number of potentially useful columns drastically. For instance, if a class has 35 lessons in a week, there can be (at most) $45!/(35! \cdot 10!)$ = over 3 billion different week columns. However, there can be only 2^9 day columns for day d, so the total number of day columns is at most $5 \cdot 2^9 = 2560$. Having much fewer columns that could be generated obviously reduces degeneracy.

There are some more advantages to using day columns. First of all, because I can combine the day columns from different days, only a few columns per day could combine into a large number of week columns (if, for example I have 3 day columns with 4 lessons for each day (so only 15 columns in total), I could combine them in $3^5 = 243$ ways producing week columns with 20 lessons). This means, that using day columns introduces more flexibility. Moreover, the graph in the column generation phase gets a lot smaller, making the shortest path algorithm faster.

On the other hand, there are also some downsides: I had to include constraints (8a)-(8c) in the ILP model, increasing the size of this model. Moreover, these constraints only allow me to penalize the deviation proportionally to the amount of deviation, in contrast to using penalty vectors allowing me to say: having 2 extra intermediate hours is four times as bad as having 1 extra intermediate hour. Moreover, if using week columns, I would not have to generate columns with different numbers of lessons (see Section 5.2.2), because the total number of lesson hours per week is known (and thus fixed). However, the advantages of using day columns outweigh the downsides, in my opinion.

Part II

Clustering

Chapter 6

Problem Description

Introduction When students make the transition from lower to upper school, they have to make some choices. Hopefully, they have some idea about which direction they wish to go after secondary school at that time, so they can choose subjects that suit their future plans. On the other hand, they are allowed to drop certain subjects that are less interesting (or that they are bad at). Of course, the students are not entirely free in choosing subjects (especially in the first year of upper school), but their choices are bound by a number of rules, imposed by the government and the school itself. In particular, students choose one from predetermined *sets* of subjects (also called *profiles* or *sectors*). For instance, a nature-and-science profile includes subjects like Physics, Chemistry, Mathematics etc. Generally, there are three types of subjects in upper school:

1. *whole-class subjects* that every student has with his 'own' class; let us call this *stem class*. Which student is in which stem class is known beforehand (usually the same as in lower school). English and PE (Physical Education) are examples of whole-class subjects.

2. *subject-set subjects* that students follow in groups of students who have chosen the same subject-set (together with students who have this subject as a free-choice subject, see the next item). This means that students of different stem classes (that are in the same year and level) are mixed.

3. *free-choice subjects* that are not in a students chosen subject-set, but that he has chosen to reach the obligatory minimum number of lessons or just because it appeals to him. Subjects like Latin and Music fall into this category. Often, a student is also allowed to choose subjects that are in a different subject-set than his own. Of course, these subjects are also followed by groups of mixed students as well.

Although the whole-class subjects could be included (see the section on page 64), we will see that the clustering problem is mainly about subject-set

subjects and free-choice subjects (that is, the subjects in which students from different stem classes are mixed).

Just like in Chapter 2, we will see in this chapter which concepts play a crucial role in this problem and how these concepts relate to each other. I also mention which of the real-life constraints I have disregarded, which constraints I have added and explain why. In Section 6.3, I treat the objective and to what extent it differs from the currently used objective. Finally, I will summarize in Section 6.4: the problem definition. However, I begin with explaining the basic idea of the clustering problem.

6.1 Basic Idea

There are two kinds of decisions that have to be made solving the clustering problem. First of all, for subjects with much interest, the subscribers must be partitioned into *groups* (for subjects with only few participants, there is only one group). Secondly, I have to make sets, the so-called cluster patterns, of groups that can be scheduled simultaneously at a time slot (deciding at which time slot to schedule such a cluster pattern is part of the timetabling problem). Because forming cluster patterns limits the choice of assigning students to groups (since students can not have more than one lesson simultaneously) and vice versa, these two decisions are jointly made while clustering. Note that different groups of the same subject do not need to be in the same cluster pattern (in fact, this is sometimes impossible if one teacher teaches more than one of these groups). The formation of cluster patterns in turn affects the timetabling problem. That is why I decided to include this problem in my research project.

For example, suppose there are two subjects A and B, both with 50 participants that have to be partitioned into 2 groups per subject, denoted by A1, A2, B1 and B2. It is part of the clustering problem to decide which student is assigned to which group. Secondly, it has to combine groups into cluster patterns, for example A1-B2 and A2-B1 (apparently, there is no student assigned to both A1 and B2 or to both A2 and B1). If subject A has to be taught 2 times a week, and subject B 3 times (these multiplicities are known), so have to be all their groups. In this case, we would could have the following solution (set of cluster patterns): A1-B2, A1-B2, A2-B1, A2-B1 and B1-B2. Of course, there can be no students assigned to both B1 and B2 since they are groups of the same subject, so the pattern B1-B2 is only feasible if they are not taught by the same teacher (which is known beforehand).

6.2 Modeling Concepts

6.2.1 Subjects

As said in the introduction, clustering is about upper school students choosing optional subjects. The number of optional subjects a student can choose from is usually much higher than the number of whole-class subjects: e.g. in 4HAVO (an extreme example), there are 7 whole-class subjects and no less than 18 optional subjects.

Choice Rules

On average, the 4HAVO students choose only 6 optional subjects. This number is a consequence of the *choice rules* that each student must obey. He must have a minimum number of lessons in order to reach the legal minimum number of teaching hours that every student must have each year. There is also a maximum number of subjects a student may choose, because an overactive student may make timetabling impossible. Moreover, there are rules concerning the subject-set the student has chosen. Of course, each student must subscribe to all subjects in his subject-set, but there may also be rules like: If you choose subject set X, you have to choose at least 2 of the subjects A, B and C. Finally, mostly applying to free-choice subjects, there can be rules of the form: It is not allowed to choose both Music and Handicraft. Such a rule makes clustering a bit easier (in this example, I know that I can (probably) schedule Music and Handicraft groups simultaneously, since there can be no student who is subscribed to both). However, although some of these rules provide useful information for the clustering problem, I do not have to include them in the model (they are not used in the currently used clustering software either). This is because the consequences of the rules are apparent in the choices of the students (which *are* part of the input).

6.2.2 Groups

After the students have chosen their optional subjects, I know the number of subscribers for each subject. If all students fit in one classroom (a classroom has a capacity of about 30 students), usually all subscribed students are incorporated into one group, and the teacher of this group is being decided. If there are more students, say 85, there has to be more than one group. In this case, 4 groups of around 21 students can be made, or 3 larger groups of about 28 students. Whether to make 3 or 4 groups is being decided by the school and depends on budget and staff resources. Again, a teacher must be assigned to each of the 3 or 4 groups. It may happen that one teacher is assigned to more than one group of the same subject (or even to multiple groups of different subjects). All these decisions are made *before* the actual

clustering begins. Note that deciding which student is being assigned to which group (in case of a multi-group subject) *is* part of the problem.

Group Sizes

When a subject with 85 subscribers is being divided into 4 groups, I normally want the group sizes to be around $85/4 \approx 21$ students each. This will be included as a soft constraint, so I am allowed to deviate from this ideal size (in order to be able to make better cluster patterns), but doing so induces penalty points (see Section 6.3). There can also be hard bounds on the group sizes, guaranteeing a more or less balanced result. In this example, I could add the hard constraints that the groups must have at least 18 and at most 24 students. An obvious (hard) upper bound for group sizes is the maximum number of students that fit in a room (about 30 at the HLC). In rare situations, I do *not* want the group sizes to be evenly balanced. This can happen when an inexperienced teacher is assigned to one of the groups. The timetabler may want this teacher to have fewer students than the experienced teachers who teach the other groups. Therefore ideal group size and hard bounds may differ from group to group.

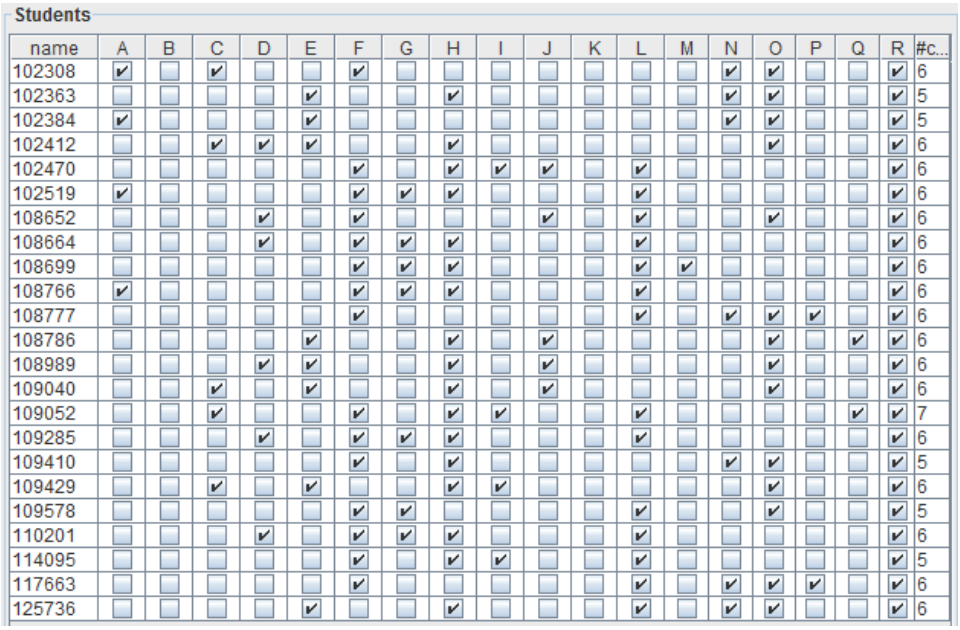
6.2.3 Students

I hope it is clear at this point that students play an important role in the clustering problem: students choose subjects, and when a subject has multiple groups, this student must be assigned to one of them. This assignment is part of the clustering problem. I ultimately want to combine groups that can be scheduled simultaneously into cluster patterns, and how I can combine depends on the group assignments: a set of groups can form a cluster pattern if there is no student who is assigned to more than one group in this set.

Stem Classes

In lower school, each student has all his lessons with his 'own' class. In upper school, this generally only happens in whole-class subjects. Optional subjects are being followed in mixed classes. To avoid confusion, I call the first type stem classes. Stem classes sometimes remain the same from the first to the last year, but it can also occur that classes are being regrouped at some point in time. In particular, regrouping can be beneficial before the first year of upper school based on the subject choices of students. An other reason to regroup may be that classes get too small due to students that repeat a class or drop out: Regrouping could then reduce the number of classes.

So optional subjects are followed in groups of students of different stem classes. Of course all students must be in the same year and of the same



Students

name	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	#c...
102308	✔		✔			✔								✔	✔			✔	6
102363					✔			✔						✔	✔			✔	5
102384	✔				✔									✔	✔			✔	5
102412			✔	✔	✔			✔							✔			✔	6
102470						✔		✔	✔	✔		✔						✔	6
102519	✔					✔	✔	✔				✔						✔	6
108652				✔		✔				✔		✔			✔			✔	6
108664				✔		✔	✔	✔				✔						✔	6
108699						✔	✔	✔				✔	✔					✔	6
108766	✔					✔	✔	✔				✔						✔	6
108777						✔						✔		✔	✔	✔		✔	6
108786					✔			✔		✔					✔		✔	✔	6
108989				✔	✔			✔		✔					✔			✔	6
109040			✔		✔			✔		✔					✔			✔	6
109052			✔			✔		✔	✔			✔					✔	✔	7
109285				✔		✔	✔	✔				✔						✔	6
109410						✔		✔						✔	✔			✔	5
109429			✔		✔			✔	✔						✔			✔	6
109578						✔	✔					✔			✔			✔	5
110201				✔		✔	✔	✔				✔						✔	6
114095						✔		✔	✔			✔						✔	5
117663						✔						✔		✔	✔	✔		✔	6
125736					✔			✔				✔		✔	✔			✔	6

Figure 6.1: Subject choices h4a

level. Let us say that that all students are of the same *type*. Clustering can be done for each type separately, since group assignments and making cluster patterns only concern students and subjects within a type. At the HLC, there are 7 types that have to be clustered: 3,4VMBO; 4,5HAVO and 4,5,6VWO.

Group Assignment Restrictions

Sometimes I am not entirely free to assign students to groups. Figure 6.1 (a screen shot from my own clustering program) shows the subjects chosen by students in 4HAVO stem class h4a.

As we can see, subject R (representing chemistry) is chosen by all 23 students in this class. This is probably not a coincidence: I presume after lower school, new stem classes are formed based on their subject choices. In the other five 4HAVO stem classes (h4b, h4c, h4d, h4e and h4f), there are 43 subscribers to this subject, making a total of 66. This subject is therefore divided into 3 groups (R1, R2 and R3) consisting of 22 students each, on average. However, it is desirable to keep the 23 students of h4a together, and to make them form one of the three groups, say group R1. To achieve this I can add the constraint that each student in h4a may only be assigned to group R1 (let us call these students are *fixed* to R1). The other 43 students that are subscribed to subject R are only allowed to be assigned to one of the groups R2 and R3. In general: I can restrict the assignment of

	Subjects																
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
period 1	3	0	0	3	3	3	0	4	2	0	0	3	3	0	4	3	3	3
2	3	0	3	3	3	3	3	4	0	2	3	4	0	4	3	3	3	0
3	0	3	0	3	3	0	3	4	2	0	0	3	0	4	0	3	3	3
4	3	0	3	0	3	3	3	4	2	2	3	4	3	3	3	3	3	0
5	3	3	3	3	0	3	3	4	2	2	3	3	2	3	3	3	3	4

Table 6.1: 4HAVO lessons table for optional subjects

students to a *subset* of the groups of a subject. I call this type of constraints *group assignment restrictions*.

If I add whole-class subjects to the problem (why I would do that is explained in the section on page 64), I must also use these type of constraints: each student must be fixed to the group corresponding to his stem class. Finally, if a subject has only one group, each student subscribed to this course is automatically fixed to this one group, of course.

6.2.4 Periods

In the timetabling problem, I do not need to care about periods, because I can solve the timetabling problem for each period separately. I can not do this with clustering, since the group assignments must remain the same throughout the whole year. However, the number of hours each subject is taught differs from period to period. Most subjects are only taught in a subset of the periods. In 4HAVO, for example, only 12 of the 18 optional subjects are taught in period 1. How many hours a subject is given each period is predetermined, and thus part of the input. This data is usually provided in a lessons table (see Table 6.1).

I could choose to cluster the periods one by one. When the clustering of period 1 is finished, I then would have to fix the resulting group assignments in the clustering of the remaining periods (since these group assignments have to maintain the same throughout the year). This is not a good idea since the optimal group assignment of period 1 could be a bad assignment for period 2. This is because in period 2, a different set of subjects is taught, possibly requiring other combinations of groups (cluster patterns). At this point, however, I am not flexible in combining groups anymore, since group assignments are fixed. Therefore I cluster for the entire year at once.

6.2.5 Teachers

As we have seen in Section 6.2.2, it is predetermined which teacher teaches which group. Sometimes, a teacher teaches more than one group of a subject. It may even happen that a teacher teaches groups of different subjects. As

said before, when combining groups into cluster patterns, I have to make sure that no student is assigned to more than one group in this pattern, because a student can not split itself. The same holds for a teacher: I can not allow two or more groups given by the same teacher to be in a pattern. This is why this information about teachers is also important for the clustering problem.

Availability

There is another reason why it is important to know which teacher teaches which group. This has to do with part-time teachers, who have blocked part of the week. Suppose teacher Abc blocked Monday, Tuesday and Wednesday and teacher Def blocked Thursday and Friday. It is useless to combine two groups taught by those teachers in a cluster pattern, since it will be impossible to schedule at a time slot in the timetabling phase. To avoid this, I can add the constraint that all teachers that are being combined in a pattern must have at least one overlapping available time slot. Usually, I want to be a bit more flexible during the timetabling phase, so demanding more than one available time slot is desirable. The availability of teachers must be part of the input, to be able to include these constraints.

6.2.6 Other Issues

There are no issues that are included in the current clustering approach that I have not mentioned in the preceding sections. We have seen in the section about teacher availability (page 61), that the problem description includes even more constraints. This is necessary to avoid problems during the timetabling phase. The clustering software that the HLC uses does not need these constraints because the formation of cluster patterns is done *in the the timetabling phase* at the HLC (see Section 6.3.2). Therefore, their cluster patterns automatically meet teacher availability constraints. Classroom constraints, which are explained in the next section, are also in this category.

Classroom Constraints

We have seen in the timetabling problem (see Section 2.1.5) that, although I do not assign lessons to classrooms explicitly, classrooms do play a role. At first glance, I do not have to care about classrooms in the clustering problem. However, it can happen that in the timetabling phase, a generated cluster pattern can not be scheduled due to the classroom constraints. Consider three biology groups B1, B2 and B3, and suppose these groups are being taught by three different teachers, who have many overlapping available time slots. The clustering could result in a pattern with B1, B2 and B3: obviously, there can not be a student who is assigned to more than one

of these groups (since they are groups of the same subject), and from the assumption that each group is taught by a different teacher, a teachers conflict is also impossible. However, there are only two biology classrooms, so this cluster pattern can never be scheduled. Therefore, I have to avoid these kind of situations in the clustering phase already. This can be achieved by introducing constraints limiting the number of groups from some set (in this case, the set of biology groups) that can be in a pattern. This means classroom constraints are of the following form:

From groups set G, there may be only n in a cluster pattern

In this case, I suggest allowing only $n = 1$ biology group (G is the set of biology groups) in a cluster pattern, in order to be more flexible in the timetabling phase. Allowing patterns with two biology groups would require (in the timetabling phase) a time slot at which both biology rooms are free, but there may not be such a time slot anymore if part of the schedule is already fixed (see Section 5.1.1).

6.3 Objective

6.3.1 My Approach

My timetabling model needs ready-to-use lessons as input. Therefore, I want the clustering algorithm to produce these lessons in the form of cluster patterns. A cluster pattern can be seen as one big lesson involving multiple classes and teachers (see the section on page 14). It is desirable for such a lesson to consist of many groups, because then a large number of students have class, and only few are free (possibly having an intermediate hour, depending on the time slot at which this pattern will be scheduled in the timetabling phase). The total number of groups is fixed, since the number of groups per subject is predetermined and the number of hours per week is known for each subject. Therefore grouping many groups in parallel means having few cluster patterns in total. This is why my primary objective is to minimize the total number of patterns. Since I cluster for the whole year, and resulting patterns can be different each period, I minimize the *sum* of the number of patterns per period. There is also a secondary objective, which is balancing the number of students among the groups of a subject (see the paragraph on page 58).

To illustrate this objective, and to compare it with the objective that is currently used, let us consider a small example: clustering for 3VMBO (having 101 students). In Table 6.2, I combined the lessons table with the group sizes of the resulting clustering of my algorithm. Note that, although subject E only has 28 subscribers, the school still decided to form two groups. Also note that these groups are perfectly balanced in my result. Table 6.3 shows per period, which patterns were formed. Since the lessons table for

	Subjects						
	A	B	C	D	E	F	G
period 1	0	3	3	3	3	3	0
2	3	3	0	3	0	3	3
3	3	0	3	0	3	0	3
4	0	3	3	3	3	3	0
5	2	3	3	3	3	3	2
#students	35	71	73	96	28	35	66
#groups	2	3	3	4	2	2	3
group 1	16	26	27	22	14	20	22
2	19	24	24	26	14	15	20
3		21	22	25			24
4				23			

Table 6.2: 3VMBO lessons table and resulting group sizes

period 1 and 4		period 2		period 3		period 5	
B2 C3 D3 E1 F2	100	A2 B1 D4 G2	88	A2 C1 E2 G3	84	A1 B3 D1 F1 G1	101
B2 C3 D3 E1 F2	100	A2 B1 D4 G2	88	A2 C1 E2 G3	84	A1 B3 D1 F1 G1	101
B2 C3 D3 E1 F2	100	A2 B1 D4 G2	88	A2 C1 E2 G3	84	B2 C3 D3 E1 F2	100
B1 C2 D2	76	B3 D1 F1 G1	85	A1 C3 E1 G1	74	B2 C3 D3 E1 F2	100
B1 C2 D2	76	B3 D1 F1 G1	85	A1 C3 G1	60	A2 B1 D4 G2	88
B1 C2 D2	76	B3 D1 F1 G1	85	A1 C3 G1	60	B3 D4 E2 G3	82
C1 D1 F1	69	A1 D2 G3	66	C2 E1 G2	58	B2 C1 D2	77
C1 D1 F1	69	A1 D2 G3	66	C2 E1 G2	58	B1 C2 D2	76
C1 D1 F1	69	A1 D2 G3	66	C2 G2	44	B1 C2 D2	76
B3 D4 E2	58	B2 D3 F2	64			C2 E1 F2 G2	73
B3 D4 E2	58	B2 D3 F2	64			A2 C1 G3	70
B3 D4 E2	58	B2 D3 F2	64			C1 D1 F1	69
						C3 D4 E2	59
						D3 E2	39

Table 6.3: 3VMBO patterns with number of students

periods 1 and 4 are identical, it is no surprise that the resulting clustering has the same patterns in these two periods.

Table 6.2 shows that my algorithm balances group sizes reasonably well. The primary objective, however was to minimize the number of patterns needed. In Table 6.3, we can see that in my result periods 1,2 and 4 have 12 patterns, period 3 has only 9 and period 5 (which has the most subjects) has 14, making a total of 59 patterns. It is possible to reduce the number of patterns in period 5 to 12, by shifting some students to other groups (the group assignments are not displayed here). However, with this new group assignment I might need more than 9 patterns in period 3. Since I am minimizing the sum of the number of patterns, this shifting is not necessarily a good thing. I could, however, include upper bounds on the number of patterns per period as extra constraints.

Note that an optimal solution of the clustering problem not necessarily allows an optimal solution of the timetabling problem. However, minimizing the number of patterns is an objective that allows *good* solutions in the timetabling phase, in my opinion.

Including Whole-Class Subjects

It is possible to include whole-class subjects in the clustering model. The HLC timetablers do not do this, but it can be beneficial. First of all, I have to make sure that each student gets fixed to the group corresponding to his own stem class by group assignment restrictions (see the section on page 59). Consequently, the clustering program does not need to care about assigning students to groups, but only to fit the whole-class groups in the cluster patterns. From my 3VMBO clustering (without whole-class subjects) in Table 6.3, consider pattern C2-G2 from period 3. There are only 2 students from stem class t3b in this pattern (they are both in G2). It may be possible to move these two students to G1 or G3, in order to have no t3b students in this pattern at all. In that case it would be possible to add a whole-class group for t3b to this cluster pattern, saving some space (that is, reducing the number of patterns needed). Thus by adding whole-class subjects to the model, the algorithm is forced to make group assignments (for the optional subjects) allowing to fit whole-class groups in the patterns.

It is not always the case that adding whole-class subjects saves space. Sometimes, you end up with cluster patterns that all have either only groups of optional subjects, or only groups of whole-class subjects. Only mixing the two types could save space. Including whole-class subjects increases the size of the model, so it takes longer to solve. Another downside is that by fixing these groups to patterns, you lose some flexibility in the timetabling phase. In Chapter 10, however, the issue of flexibility will be solved.

6.3.2 Current Approach

Until one of the last interviews at the HLC, I thought that their clustering program worked more or less the same way as mine. But when I wanted to compare one of my clusterings with one of theirs I found out that they use a different objective. Table 6.4 shows the complete result of their clustering program (except the exact assignment of students to groups). Surprisingly, the number of students per subject is different from those in Table 6.2. This is because the data I used is older than in their result: Now there are only 99 students in 3VMBO (apparently 2 dropped out). We see that the quality of their group balancing is comparable to mine.

Even more surprising is the bottom half of the table. Their clustering program does not make patterns, but so-called *lines*. A line is similar to a pattern in that it also is a set of groups that could be scheduled in parallel. However it does not seem to use any information of the lessons table: these lines are not period-dependent and each group only appears once. The formation of lines can be seen as clustering like in my approach for one period, in which each subject is given only once a week. The main objective is to have as few lines as possible.

	Subjects						
	A	B	C	D	E	F	G
period 1	0	3	3	3	3	3	0
2	3	3	0	3	0	3	3
3	3	0	3	0	3	0	3
4	0	3	3	3	3	3	0
5	2	3	3	3	3	3	2
#students	33	70	71	95	28	33	66
#groups	2	3	3	4	2	2	3
group 1	16	26	25	23	14	16	19
2	17	24	25	25	14	17	23
3		19	21	22			24
4				25			
line 1	A1		C1	D1		F1	G1
2		B1	C2	D2			G2
3		B2	C3	D3	E1	F2	
4	A2	B3		D4	E2		G3

Table 6.4: 3VMBO HLC clustering

4 groups	A1	C1		E2	G3
	A1	C1		E2	
	A1	C1			G1 (line 1)
	A1	C1			G3
	A1			E2	G3
	A2			E1	G1
	A2			E1	G2
	A2			E2	G3 (line 4)
3 groups		C1	C2	E1	
		C1	C2	E2	
		C1	C3	E1	
		C1	C3	E2	
		C1		E1	G1
		C1		E2	G3
			C2	E1	G2
			C3	E1	G2
<3 groups	A1				G2
	plus all subpatterns				

Table 6.5: Possible patterns

This approach is only possible since, in their approach, it is the responsibility of the *timetabling* software to combine groups into cluster patterns. The main responsibility of their clustering program is to make group assignments in such a way that the timetabling program can *nicely* combine the groups. What 'nicely' means is not exactly clear, but it has something to do with having not too many patterns—my objective of *clustering*—as well as some flexibility in combining groups into patterns. Having as few lines as possible takes care of that. This approach has the advantage of being more flexible in the timetabling phase. As a consequence, teacher availability and classroom constraints do not have to be included. The downsides are that you need a more powerful timetabling program, and that its result is not period dependent.

To illustrate the downside of not being period dependent, let us consider period 3. My approach was able to fit all groups into only 9 patterns (see Table 6.3), using the information from the lessons table explicitly. Table 6.5 shows all patterns that could be formed in the timetabling phase using the group assignments from the HLC clustering program. These are the patterns that their timetabling program could use (it is not committed to the lines in any way, only to the student assignments to the groups). All but one pattern with less than 3 groups are omitted, since they are simply the subpatterns of the larger ones. You can try it yourself, but it is impossible to cover all multiplicities using only 9 of these patterns. In Chapter 10, I will propose a new method that has both flexibility and is period dependent.

	Subjects	
	B	N
period 1	3	0
2	0	3
3	2	0
4	0	3
5	0	2
line 8	B1	N2

Table 6.6: Alternating Subjects

Alternating Subjects

I said before that the current approach does not seem to use the lessons table. This is not exactly true: it is probably used to value the resulting lines. There is a special case where it is obvious that the lessons table *is* used. Consider a part of the lessons table of 5VWO, with one of the resulting lines in Table 6.6. There are some students who have been assigned to both groups B1 and N2. Still, these groups are in one line. This is possible since there is no period in which both subjects are being given (let us call B and N *alternating* subjects), so there can be no harm in placing the two groups in the same line. My approach automatically takes care of this issue, since it has different patterns each period.

6.4 Problem Definition

Having treated all concepts that are relevant to this clustering problem, I can summarize the input I need, the objective and constraints, and the output I get, in a problem definition:

Input

- A set of stem classes of the same type, with their students
- A set of subjects that must be clustered (possibly including whole-class subjects)
- For each period, the number of hours per week each subject must be taught (that is, a lessons table)
- The groups per subject, with their teachers
- Hard bounds on group sizes
- Ideal group sizes
- The subscriptions of the students
- The group assignment restrictions (if any) for each student
- Teacher availability
- Classroom information
- Upper bounds on the number of cluster patterns per period (optional)

Objective

- Form as few cluster patterns as possible

Soft constraints (violating induce penalty points)

- Group sizes have to be as close as possible to their ideal sizes

Hard constraints (must be met)

- The number of hours per week for each subject and for each period must be respected
- Each student must be assigned to exactly one group per subject that he is subscribed to
- Group assignment restrictions must be respected
- The size of each group must be within its hard bounds
- A student may not be assigned to more than one group in each cluster pattern
- A teacher may not teach more than one group in each cluster pattern
- The teachers in each cluster pattern must have sufficient overlapping availability
- The classrooms constraints have to be respected
- The upper bound on the number of cluster patterns per period may not be exceeded

Output

- Assignments of students to groups
- For each period, a set of cluster patterns

Chapter 7

Modeling

Introduction Just like the timetabling problem, the clustering problem can be modeled as an integer linear program. Also, I use the concept of columns in the ILP model. In this case, a column corresponds to a feasible cluster pattern. How these columns are generated is treated in Chapter 8. First, using the problem definition from Section 6.4 as a starting point, I describe which types of variables I need, which constraints I must add and what the objective is. Finally, I state the full model.

Just like in Chapter 3, I use consistent numbering and naming throughout this chapter. Numbered constraints correspond to those in the full model (Section 7.4). Indices are named as follows: i denotes a student, s a subject, g is a group, p a period and c stands for a column.

7.1 Variables

There are two decisions to be made while solving the clustering problem: which cluster patterns to use and how to assign students to groups. Like I said in the introduction, cluster patterns are modeled as columns in the ILP model. A column c can be seen as a 0-1 vector, with a component for each group g:

$$c_g = \begin{cases} 1 & \text{if group } g \text{ is in this cluster pattern} \\ 0 & \text{otherwise} \end{cases}$$

Note that the components of this vector are constants in the ILP model. Since in each period, different subjects may be taught, there is a set of columns C^p for each period p. Just like in the timetabling problem, I need variables x_c to be able to decide which columns to choose. However, in this case, a cluster pattern (column) may be chosen more than once in general, so the x_c are not 0-1 variables in the clustering problem:

$$x_c = \begin{cases} n & \text{if column } c \text{ is chosen } n \text{ times} \\ 0 & \text{otherwise} \end{cases}$$

Of course, there is an upper bound on n, because each group must appear in a fixed number of cluster patterns (the number of hours the corresponding subject must be given in a week). This upper bound is guaranteed by the constraints (1), see Section 7.2.1.

Besides variables x_c, I need variables to be able to assign students to groups. Every student i has a set of subjects S_i he is subscribed to. Each subject has one or more groups. The set of all groups (from all subjects in S_i) this student i may be assigned to (possibly restricted by the group assignment restrictions, see the section on page 59) is denoted by G_i. For each student i and each group $g \in G_i$, I introduce 0-1 variables:

$$y_{ig} = \begin{cases} 1 & \text{if student } i \text{ is assigned to group } g \\ 0 & \text{otherwise} \end{cases}$$

7.2 Constraints

7.2.1 Number of Hours per Week

Of course, the lessons table must be respected. For each subject and period, the lessons table tells us how many hours per week all groups of this subject, must be given: this table determines the values m_g^p, denoting this number for group g and period p. This means that each group g must appear m_g^p times in the chosen columns for period p. I can force this by the following constraint:

$$\sum_{c \in C^p} x_c c_g = m_g^p \quad \text{for all } g, p \quad (1)$$

If a column c has $c_g = 1$, we can see from this constraint that x_c can never exceed value m_g^p. These constraints thus imply upper bounds on the variables x_c.

7.2.2 One Group per Subject

Of course, every student must be assigned to exactly one group of each subject he is subscribed to. Of course, I must respect the group assignment restrictions (see Section 6.2.3). Let us denote the set of groups of subject $s \in S_i$ that student i may be assigned to by $G_{i,s}$ (which is obviously a subset of G_i). This leads to the following constraints:

$$\sum_{g \in G_{i,s}} y_{ig} = 1 \quad \text{for all } i, s \in S_i \quad (2)$$

7.2.3 Number of Cluster Patterns per Period

Optionally, I can include constraints that bound the number of cluster patterns I can use per period (this is usually done to balance the number of

patterns over the periods). Denoting the maximum number of patterns for period p by $\hat{n}^p$, the following constraints take care of this:

$$\sum_{c \in C^p} x_c \leq \hat{n}^p \quad \text{for all } p \quad (3)$$

7.2.4 Group Sizes

I can have hard constraints on the number of students in a group (for instance, to make sure all students fit in a classroom), as well as soft constraints that measure deviation from ideal group sizes. Let us denote the hard upper- and lower bound on the size of group g by $\hat{n}_g$ and $\check{n}_g$ respectively. If we let I_g denote the set of students that can be assigned to group g (this means $g \in G_i \Leftrightarrow i \in I_g$), the following constraints guarantee that the group sizes remain within these bounds:

$$\sum_{i \in I_g} y_{ig} \leq \hat{n}_g \qquad \text{for all } g \qquad (4a)$$

$$\sum_{i \in I_g} y_{ig} \geq \check{n}_g \qquad \text{for all } g \qquad (4b)$$

To be able to measure deviation from the ideal group size n_g^{ideal} of group g, I need additional variables z_g^+ and z_g^- measuring the number of students over and under the ideal size respectively (so at least one of them is 0 in an optimal solution). These variables are penalized in the objective (see Section 7.3). The following constraints measure deviation:

$$\sum_{i \in I_g} y_{ig} - z_g^+ + z_g^- = n_g^{\text{ideal}} \quad \text{for all } g \quad (4c)$$

Since the ideal group size may be non-integer (for instance if I want to evenly balance 55 students over 2 groups: $n_g^{\text{ideal}} = 22.5$), variables z_g^+ and z_g^- may also be non integer. In fact, if I have a non-integer n_g^{ideal}, I will always deviate from this ideal group size (since $\sum_{i \in I_g} y_{ig}$ *is* integer), inducing some penalty points (this is not a problem).

7.2.5 No Overlapping Students

'*A student may not be assigned to more than one group in each cluster pattern*' is one of the hard constraints from the problem definition in Section 6.4. To include this constraint, I need information about the assignment of students to groups (variables y_{ig}) and about which patterns are chosen (variables x_c). This constraint thus combines the two types of variables, and makes it a harder problem to solve.

It is not difficult to see that I can rewrite the constraint as: '*If a student is assigned to both groups g_1 and g_2, there may not be a pattern (in any*

period) containing both these groups'. This shows that I need constraints for each student and all possible pairs of groups he can be assigned to. I need a way to formalize the condition of student i being assigned to both groups g_1 and g_2 to be able to include it in the model. This condition means that $y_{ig_1} = 1 \wedge y_{ig_2} = 1 \Leftrightarrow y_{ig_1} + y_{ig_2} = 2$. Denoting the set of columns that have $c_g = 1$ (regardless of the period the column belongs to) by C_g, I can write the condition that there may not be a pattern which has both groups g_1 and g_2 as $\sum_{c \in C_{g_1} \cap C_{g_2}} x_c = 0$. Therefore, the constraint can be written as:

$$y_{ig_1} + y_{ig_2} = 2 \Rightarrow \sum_{c \in C_{g_1} \cap C_{g_2}} x_c = 0 \quad \text{for all } i,\ g_1,g_2 \in G_i, g_1 \neq g_2$$

If the value of $\sum_{c \in C_{g_1} \cap C_{g_2}} x_c$ would have 1 as upper bound, I could simply add constraint $y_{ig_1} + y_{ig_2} + \sum_{c \in C_{g_1} \cap C_{g_2}} x_c \leq 2$. However, if $y_{ig_1} + y_{ig_2} \leq 1$, there is no additional bound on the x_c and $\sum_{c \in C_{g_1} \cap C_{g_2}} x_c$ can be bigger than one. Of course, I can calculate an upper bound on this quantity (by using the values $m^p_{g_1}$ and $m^p_{g_2}$ for each period, for instance). Denote this upper bound by $M_{(g_1,g_2)}$. Then the following constraints make sure each student has at most one group per chosen cluster pattern:

$$M_{(g_1,g_2)}(y_{ig_1} + y_{ig_2}) + \sum_{c \in C_{g_1} \cap C_{g_2}} x_c \leq 2M_{(g_1,g_2)} \quad \text{for all } i,\ g_1,g_2 \in G_i, g_1 \neq g_2 \quad (5)$$

Note that if g_1 and g_2 are groups of the same subject, this constraint is automatically met, because then at most one of the variables y_{ig_1} and y_{ig_2} equals 1 (by constraint (2)).

7.2.6 Variable Bounds and Integrality

Variables y_{ig} are 0-1-variables. The x_c must be nonnegative integers (an upper bound does not need to be forced, constraints (1) take care of this). Finally, variables z^+_g and z^-_g are also nonnegative, but do not need to be integer. Summarizing, I have the following bounds:

$$x_c \in \mathbb{Z}_{\geq 0} \quad \text{for all } c \quad (6a)$$

$$y_{ig} \in \{0,1\} \quad \text{for all } i,g \in G_i \quad (6b)$$

$$z^+_g, z^-_g \geq 0 \quad \text{for all } g \quad (6c)$$

7.2.7 Other Constraints

There are still three (hard) constraints from the problem definition that I have not treated yet:

- A teacher may not teach more than one group in each cluster pattern
- The teachers in each cluster pattern must have sufficient overlapping availability
- The classrooms constraints have to be respected

These constraints are not included in the ILP model, because they only tell us something about which cluster patterns are allowed and which are not. Therefore, these constraints will be included in the column generation algorithm (see Chapter 8).

7.3 Objective

The main objective is to minimize the total number of cluster patterns used. This can be expressed easily in the following objective:

$$\min \sum_{c} x_c \quad \text{(obj)}$$

There is also a secondary objective: keeping the group sizes as close to their ideal size as possible. The deviation is measured by the variables z_g^+ and z_g^-, so I must include them in the objective as well. However, I can not sacrifice an extra cluster pattern for better group balancing; group sizes have a lower priority. Therefore I include a small weighing constant c_z that forces this priority. The final objective becomes:

$$\min \sum_{c} x_c + c_z \sum_{g} (z_g^+ + z_g^-) \quad \text{(obj)}$$

7.4 Full Model

Now I am ready to formulate the full ILP model with all its constraints:

	$\min \sum_c x_c + c_z \sum_g (z_g^+ + z_g^-)$	(obj)	Objective
Subject to:			
$\sum_{c \in C^p} x_c c_g = m_g^p$	for all g, p	(1)	Lessons table
$\sum_{g \in G_{i,s}} y_{ig} = 1$	for all $i, s \in S_i$	(2)	One group per subject
$\sum_{c \in C^p} x_c \leq \hat{n}^p$	for all p	(3)	Number of cluster patterns per period
$\sum_{i \in I_g} y_{ig} \leq \hat{n}_g$	for all g	(4a)	Group sizes
$\sum_{i \in I_g} y_{ig} \geq \check{n}_g$	for all g	(4b)	
$\sum_{i \in I_g} y_{ig} - z_g^+ + z_g^- = n_g^{\text{ideal}}$	for all g	(4c)	
$M_{(g_1,g_2)}(y_{ig_1} + y_{ig_2}) + \sum_{c \in C_{g_1} \cap C_{g_2}} x_c \leq 2M_{(g_1,g_2)}$	for all i, $g_1, g_2 \in G_i, g_1 \neq g_2$	(5)	No overlapping students
$x_c \in \mathbb{Z}_{\geq 0}$	for all c	(6a)	Variable bounds and integrality
$y_{ig} \in \{0, 1\}$	for all $i, g \in G_i$	(6b)	
$z_g^+, z_g^- \geq 0$	for all g	(6c)	

Chapter 8

Generating Columns

Introduction Although the title of this chapter might suggest that for the clustering problem I also have to use column generation techniques as we know it from LP theory (and of course from the timetabling problem, see Chapter 4), I do not. This is because in the clustering problem all columns fit in memory, so I do not need to only generate columns with low reduced cost, for instance. Instead, I generate *all* feasible columns before I start solving the ILP model. What 'feasible' in this case means, and how to generate such columns, is the subject of this chapter.

I begin with describing which constraints a cluster pattern has to meet to be regarded as feasible. In Section 8.2, I explain how I construct such feasible patterns without wasting time on too many feasibility checks. I conclude with the full column generation algorithm that is used.

8.1 Feasibility Constraints

We know that a column is simply a set of groups that could ultimately be scheduled simultaneously. Of course each group can appear only once in a column. Moreover, I have some other feasibility constraints like:

1. A teacher may not teach more than one group in each cluster pattern
2. The teachers in each cluster pattern must have sufficient overlapping availability
3. The classrooms constraints have to be respected

If I only check the columns for above three constraints, I observed that the column generation algorithm (CG) produced a lot of columns that were still useless in the ILP model. The word *useless* means, that the corresponding variable x_c can never become positive in the ILP model. In the following two sections we will see some examples of useless patterns. Although I could add useless columns to the model, it is desirable to filter them out before

adding the set of patterns: I have to tighten the feasible set, by adding more constraints to the feasibility check. This has several advantages. First of all, it saves memory space. Moreover, the ILP solver runs faster without useless columns. Finally, in some cases it improves the running time of the column generation algorithm, as I can prune more branches of the search tree, because of the more strict feasibility check. In most cases, however, this effect is nullified by the extra time I have to spend on these checks, caused by the extra constraints filtering out useless patterns.

8.1.1 Number of Students

With only the three constraints mentioned so far, a large amount of big patterns (with a lot of groups) are generated. Since the total number of students that have to be assigned is fixed, the number of students per group in such a pattern can get too low. 'Too low' in this case means: lower than the minimum number of students that the ILP model allows to assign to such a group.

I can calculate a lower bound $m(g)$ on the number of students that the ILP model allows to group g. Of course, we have the relation $m(g) \geq \check{n}_g$ (where $\check{n}_g$ is the hard lower bound on the group size of g, see Section 7.2.4). The number of students that are fixed to this group (because of group assignment restrictions, or because g is the only group of the subject, see Section 6.2.3) also is a lower bound on $m(g)$. Finally, if g is a group of subject s, and $\bar{G}^g$ denotes the set of the *other* groups (besides g) of the same subject s, we know that at most $\sum_{\bar{g}\in\bar{G}^g} \hat{n}_{\bar{g}}$ students can be assigned to the other groups of this subject. If we let n_s denote the total number of subscribers to subject s, we know that at least $n_s - \sum_{\bar{g}\in\bar{G}^g} \hat{n}_{\bar{g}}$ students must be assigned to group g. This is a third lower bound on $m(g)$. I can thus define $m(g)$ as the maximum of these three lower bounds:

$$m(g) = \max\{\check{n}_g,\ \#\text{students fixed to } g,\ n_s - \sum_{\bar{g}\in\bar{G}^g} \hat{n}_{\bar{g}}\}$$

These values $m(g)$ can be calculated before I start generating patterns. If I have these values $m(g)$ at my disposal, I can use them to calculate a lower bound on the number of students in a pattern P: $\sum_{g\in P} m(g)$. If the total number of students is less than this lower bound, the pattern is infeasible, of course. This leads to a new constraint:

4. A pattern P is feasible, only if the total number of students is at least $\sum_{g\in P} m(g)$

8.1.2 ILP Feasibility Check

There are still columns that pass the above feasibility checks, but are still useless in the model. Which constraints are responsible for that differs from

case to case. One obvious example is when a student i is fixed to both groups g_1 and g_2. This means that (by constraint (5) of the ILP model) all columns c with both these groups must have $x_c = 0$: these columns are infeasible since a student cannot follow two lessons at the same time.

I could decide to add a constraint to the feasibility check, testing whether there is a student who is fixed to more than one group in the examined pattern. But still, there would be columns generated that are (less obviously) useless. Therefore, I decided to test for each column that meets all above constraints, whether the constraints in the ILP model would force the corresponding variable to 0. For this purpose, I use the following reduced ILP model:

$\sum_{g \in G_{i,s}} y_{ig}$	$=$ 1	for all $i, s \in S_i$	(2)
$\sum_{i \in I_g} y_{ig}$	$\leq \hat{n}_g$	for all g	(4a)
$\sum_{i \in I_g} y_{ig}$	$\geq \check{n}_g$	for all g	(4b)
$y_{ig_1} + y_{ig_2}$	$\leq$ 1	for all i, g_1,$g_2 \in G_i \cap P, g_1 \neq g_2$	(5')
y_{ig}	$\in \{0,1\}$	for all $i, g \in G_i$	(6b)

This model can be seen as a simplified version of the full model from Chapter 7. In this reduced model I have only one column: the pattern P (a set of groups) that has to be checked for feasibility, and I force this column to be used (that is, I fix $x_c = 1$; this is no longer a variable, but becomes a constant). Since we are checking feasibility for one single pattern, constraint (1) (lessons table) and constraint (3) (the number of patterns per period) do not apply here. I am only checking for feasibility, so I do not need an objective or measure group size deviation (constraint (4c)). Since I fixed $x_c = 1$, I can reduce constraint (5) to a simpler one (5'). I can restrict myself in this constraint to groups g_1 and g_2 that are both in pattern P (since otherwise there is no danger of overlapping students).

I am left with a simple model that only has variables y_{ig}, and checks whether there exists a group assignment *for all groups* that meets the hard group size constraints, and has no overlapping students *in this pattern*. If the solver finds out this model is infeasible, then the pattern is also useless in the full ILP model. If there *is* a student assignment that allows this pattern, I regard P as feasible. The solver takes a fraction of a second to solve this reduced ILP model. This is the final step in the feasibility check, and corresponds to constraint:

5. The reduced ILP model for each pattern must be feasible

8.1.3 Pseudocode Feasibility Check

Now that I have formulated all constraints that have to be met before I call a pattern feasible, I can display the pseudocode of IsFeasible, which does

not need further explanation.

IsFeasible(P)

Input: a pattern P
Output: **true** if P is feasible, **false** otherwise

```
1  if P contains two groups taught by the same teacher then
2      return false
3  if the teachers in P have insufficient overlapping availability then
4      return false
5  if the classroom constraints are violated for the groups in P then
6      return false
7  if the total number of students is less than ∑_{g∈P} m(g) then
8      return false
9  if the reduced ILP model for P is infeasible then
10     return false
11 return true
```

8.2 Feasible Column Construction

By examining above five feasibility constraints, we can see that the following property for cluster patterns always holds:

If a cluster pattern is feasible, so are all its subpatterns.

Naturally, P' is a subpattern of P if its set of groups is a subset of the set of groups of P. We will see that if I use this property and construct patterns in a smart way, I can save a lot of feasibility checks (compared to simply enumerating all possible patterns and checking them all for feasibility).

The column construction algorithm that I will use, recursively composes patterns by adding groups one by one, until the pattern becomes infeasible. By the above property, it is useless to add more groups to an infeasible pattern: it will remain infeasible. I use the function GeneratePatterns for the construction of the set of all feasible patterns. In pseudocode, this function looks like this:

GeneratePatterns(P, G)

Input: a (sub)pattern P and a stack G of groups
that we could try adding to P
Output: a set S of all feasible patterns
having P as a proper subpattern

```
1   S = ∅
2   if G = ∅ then
3       return S
4   g = first group from G
5   G' = G\{g}
6   P' = P ∪ {g}
7   if IsFeasible(P') then
8       S = S ∪ {P'} ∪ GeneratePatterns(P', G')          (use g)
9   S = S ∪ GeneratePatterns(P, G')                  (do not use g)
10  return S
```

We see that pattern P is treated as a set of groups (which makes sense). I generate a set of patterns C^p for each period p separately. I call this function with the following parameters

$P = \emptyset$

$G =$ the set of *all* groups that must be taught in period p

Now the function must return a set S consisting of all feasible patterns having the empty set as a proper subpattern. It uses groups from G to add to the empty P, so the set S contains all feasible patterns for period p, as needed.

The algorithm works in such a way that each pattern is checked at most once for feasibility (calling the function IsFeasible), and it does not extend infeasible patterns. As we can see, GeneratePatterns recursively calls itself, unless the set G is empty (in that case, of course no additional patterns can be generated, and an empty S is returned).

In words, the algorithm works as follows. After initializing the (local) set S as empty set (line 1), and checking whether G is empty (if it is, return an empty S; line 2,3) we take the first group g from G (line 4). We remove this group from G, producing G' (a subset of G), and add it to pattern P, producing P' (lines 5,6). Now P' is a pattern having P as a proper subpattern, but we still have to check it for feasibility (line 7). If it is, add it to S and try to extend it even further using groups from G' by recursively calling GeneratePatterns(P', G') adding its return set to S too (line 8). This return set of GeneratePatterns(P', G') thus contains all feasible patterns having P' as proper subpattern, in other words: having P as a proper subpattern *and* containing the group g.

If P' is infeasible, it is useless to try to extend it, because of the property from the beginning of this section: apparently there are no patterns that contain g and have P as a proper subpattern. We also want S to contain patterns having P as proper subset *not* containing g. Therefore, I also add the return set of the recursive call GeneratePatterns(P, G') to S (line 9). Note that the recursive calls both use the smaller set G' as parameter, and therefore the algorithm is guaranteed to terminate. Finally, the complete set S is returned (line 10).

8.2.1 An Example

In Figure 8.1 we can see how this algorithm runs on a simplified example. In this example, there are only 4 groups: A,B,C and D. Every pattern is feasible, except patterns containing both groups B and C (they are given by the same teacher).

The rectangles in the figure represent functions GeneratePatterns with the two displayed sets as parameters. The arcs are the function calls, and the number stands for the corresponding line in the pseudocode. If we look at the pseudocode, only when there is a function call in line 8, a new pattern is added to the total set of feasible patterns: they are in the last column of the figure. The rectangles are drawn in such a way that the top-to-bottom order corresponds to the order in which these functions are called.

The figure shows that indeed all 11 feasible patterns are generated, and that they are generated only once (no redundant work is done). We can also see that there are only two failed feasibility checks, while there are 4 infeasible patterns (BC, ABC, BCD and ABCD). For a more realistic instance (with many more groups, and constraints), the effect of saving feasibility checks (and thus saving time) is even much bigger.

8.3 Algorithm Summary

As a summary, I list what the column generation algorithm does, together with its input and output.

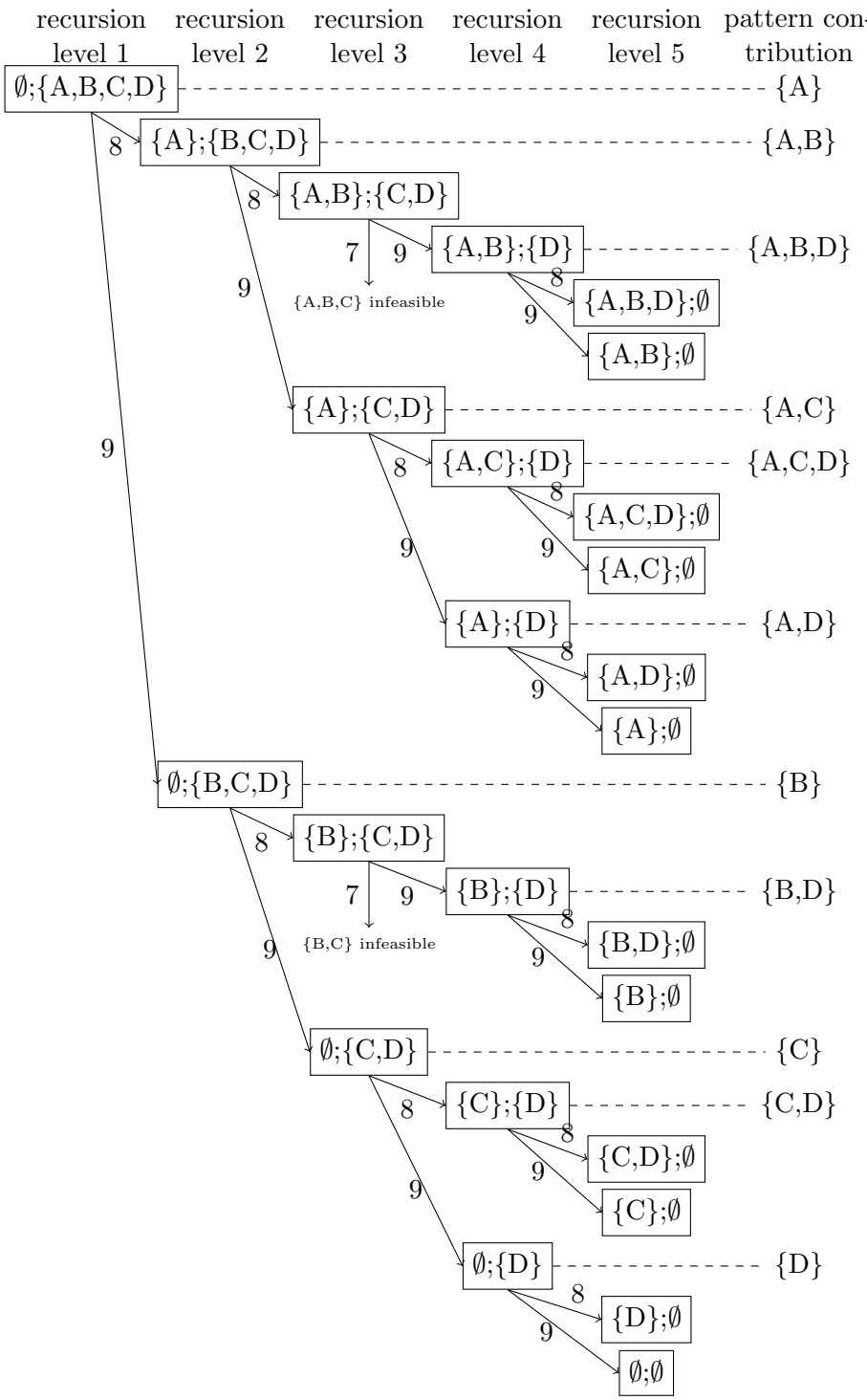


Figure 8.1: Function calls of GeneratePatterns

Input

- The period p for which we want to generate patterns
- A set of stem classes of the same type, with their students
- A set of groups that must be clustered in period p, with their teachers
- Hard bounds on group sizes
- The subscriptions of the students
- The group assignment restrictions (if any) for each student
- Teacher availability
- Classroom information

Objective

- Generate all feasible cluster patterns for period p

Method

- Calculate for each group g the minimum number of students $m(g)$
- Generate patterns using algorithm GeneratePatterns
- Perform feasibility checks using algorithm IsFeasible

Output

- A set of all feasible cluster patterns for period p

Chapter 9

Implementation Matters

Chapters 7 and 8 describe the theoretical model and algorithms for the clustering problem. When I tried to implement these models in my own clustering program, I observed that for big models, the solver took a large amount of time to find a good solution. In this chapter, two techniques for improving the running time are discussed. How well these techniques work is treated in Chapter 11, presenting the computational results (running times, solution qualities etc.).

9.1 Column Generation After All

Because all columns fit in memory, I do not need column generation. Still, I can speed up the solving process by starting solving with a restricted set of columns, and adding more columns when needed. The column generation scheme that I will use is different from that of the standard column generation as described in Appendix B: I generate columns based on solutions of the ILP model, rather than based on reduced costs for LP relaxation solutions.

As a starting point, I generate all feasible columns that contain groups from at most 2 different subjects, and solve the ILP model with this restricted set of columns to optimality. Using this restricted set has two advantages. First of all, this partial ILP model can be solved very quickly (within seconds). Secondly, the solution gives me some information about from which subjects the groups could be combined. The constraint that is restricting the choice of cluster patterns the most, is constraint (5), avoiding overlapping students. Using only patterns with groups from at most 2 different subjects, I keep the matter of overlapping students simple: since students can not be assigned to groups of the same subject, overlapping can only occur in patterns with groups from two subjects, and no complicated overlapping structures involving many groups have to be considered. This is probably the reason why I can solve this ILP model this quickly.

I use this information to generate more columns: I add all columns that have at least one pair of groups (from different subjects) in common with one of the cluster patterns from the solution of the partial ILP model. The reason for this is that a combination of groups that is also in the solution of the partial ILP model is apparently useful. Since I require only one pair of groups to overlap with one of the patterns from the solution, now patterns with groups from more than two subjects are also generated. Then I start the solver again, trying to find a better solution.

As soon as a better solution is found (this does not have to be the optimal solution), I again generate new columns that have an overlapping pair of groups with this new solution. I repeat this procedure of generating extra columns and finding better solutions until I can not find a better solution anymore. If this happens, I add *all* remaining feasible columns, and solve the full model to optimality. Using this approach, I improve feasible solutions more quickly (since I solve partial models with a smaller set of columns), and in some cases solve the clustering problem faster (see Section 11.1.2 for computational results).

9.2 Decomposition

Another way to reduce the running time of the solver is to decompose the problem. However, this usually reduces the quality of the final solution. We already have seen an example of decomposition in the timetabling problem: solving in stages (see Section 5.1.1). I do not want to wait too long for an optimal (or good) solution, so decomposition may be worthwhile.

First of all, I could partition the subjects into two or more sets. I then run the clustering algorithm for each of these subject sets using only patterns with groups from within such a set. Finally, I can collect the partial solutions into one final set of cluster patterns. Consequently, all patterns only contain groups from subjects that are from one of the subject sets. This usually is too much of a restriction, and the resulting solutions are not very good. The only way this approach could be useful is by having all whole-class subjects in one set, and all optional subjects in the other subject set. Since combining groups from whole-class subjects with groups from optional subjects is impossible most of the time, this decomposition produces solutions that are close to optimal.

A decomposition that is more useful is splitting into periods. In Section 6.2.4, I already explained how I could do this (and why this probably produces lower quality solutions): Start clustering for one, or maybe more than one, period, and while fixing the resulting solution, cluster the next period (or maybe all remaining periods). Again, fixing a partial solution is adding more constraints and reduces the quality of the solution. However, if I do this fixing 'carefully' (using some experience), it is possible to improve

running time a lot with little (no guarantees!) quality loss.

There are two things that could be fixed: cluster patterns or the group assignments (or both). The next two sections describe how this can be done.

9.2.1 Fixing Cluster Patterns

If I have a solution for some period, there are probably some patterns in this solution that I like (probably the patterns with a lot of groups). In order to speed up solving the remaining periods, I could fix these patterns. Fixing a pattern does not mean forcing this pattern to appear in the solution of the other periods; this would not be a good idea, since maybe not all groups in this pattern must be taught in an other period. A better way of fixing a pattern is forcing this pattern to be feasible in the final solution: that is, the final solution must have a group assignment that allows this pattern. I can implement this condition by adding the following constraints to the model, for each fixed pattern P^f:

$$y_{ig_1} + y_{ig_2} \leq 1 \quad \text{for all } i,\ g_1,g_2 \in G_i \cap P^f, g_1 \neq g_2 \quad (5^f)$$

This constraint makes sure that there may not be a group assignment that would have overlapping students in the fixed pattern if this pattern would be chosen. I also add these constraints to the ILP feasibility check (see Section 8.1.2), to restrict the set of generated columns. Fixing patterns thus speeds up the solving process in two ways: I restrict the possibilities of group assignments, and I have fewer patterns in the model.

I could also decide to only add the constraints (5^f) to the ILP feasibility check and not to the final model. This still improves running time, since I have fewer columns in the model, but I am more flexible when making group assignments.

9.2.2 Fixing Group Assignments

I could also fix the assignments of students to groups. This can be done by fixing the variables $y_{ig} = 1$. Of course, this also restricts the set of feasible columns, as I add these constraints to the ILP feasibility check too. I could completely fix the group assignments from the solution of one period, but this approach may be too limiting for clustering the other periods: the only thing left to decide by the clustering algorithm is which patterns to use.

The approach that I use builds in some flexibility by allowing to deviate from fixed group assignments. However, I must limit this deviation (otherwise, the fixation would be useless). I have chosen to allow a fixed number n_g^{switch} of students who are fixed to group g to switch to an other group (of the same subject). I need to add the following constraints to the model (of course they are also added to the ILP feasibility check in order to reduce

the number of generated columns):

$$\sum_{i \text{ fixed to } g} (1 - y_{ig}) \leq n_g^{\text{switch}} \quad \text{for all } g \quad (7)$$

If student i is fixed to g, and is still assigned to another group of the same subject as g, then $y_{ig} = 0$ and thus $(1 - y_{ig})$ becomes 1. This means that $\sum_{i \text{ fixed to } g}(1 - y_{ig})$ equals the number of students who have switched from group g to another group, and this number is bounded by n_g^{switch}.

I can be strict by setting the constants n_g^{switch} to low values (setting them to 0 yields a hard fixation which is generally not a good idea), or lenient by setting them to high values (too high values yields complete freedom in assignments). It is also possible to set different values n_g^{switch} for each group g. I have to play around a bit with these values in order to get good and quick results.

9.2.3 Experimenting

There is no clear guideline which fixation method to use, and with which parameters (for instance, which period to solve first). I could even fix both cluster patterns and group assignments. Also, I could keep on solving and fixing periods iteratively, until I am satisfied with the solution. The decisions of what to fix and which period to start solving determines the quality of the final solution, as well as the speed-up factor. Playing around with these choices may seem a bit arbitrary, but if I make these decisions in a smart way, these techniques give good results. Note that I only have to use these techniques for larger instances: for small instances I find good solutions quickly enough, so I do not need to sacrifice solution quality for a lower running time.

Part III

The Big Picture

Chapter 10

A New Distribution of Work

Introduction In this chapter I will treat a new approach to the the timetabling and the clustering problem. More precisely, I will make a different distribution of work between the two problems: the timetabling algorithm takes over some of the responsibility from the clustering algorithm. I think this new approach ultimately results in better timetables and may even resolve infeasibility problems. This chapter begins with explaining the general idea. In Section 10.2 and 10.3 I explain how I have to adapt the clustering and timetabling models, respectively.

10.1 General Idea

In the approach described in Chapter 2 and 6, it is the responsibility of the clustering algorithm to assign students to groups and to make feasible ready-to-use cluster patterns, which are input to the timetabling algorithm. Initially, this worked well but when I added more and more constraints to the timetabling problem (classrooms, double hours, spreading) some problems arose.

10.1.1 The Problem

Suppose for example that subjects A and B must be given twice a week, and the clustering algorithm produces two cluster patterns A-B. Suppose subject A must be given as a double hour: this means that the two patterns A-B must be scheduled consecutively, while it is probably desirable to spread the lessons of subject B over the days in the week. The double hour constraint for subject A thus conflicts with the preference to spread lessons of subject B, because the clustering algorithm grouped the corresponding groups into cluster patterns.

In this example, I am forced to accept some penalty points for not spreading lessons of subject B. Although this is of course an undesirable situation,

it is not a very big problem. However, more complicated situations could arise involving several conflicting constraints causing many penalty points or even infeasibility. In particular, this can happen when the cluster patterns consist of many groups (for instance, the patterns for 4HAVO may consist of up to 7 groups). Also, when a partial schedule is fixed (because I schedule in stages, see Section 5.1.1), there is a higher risk of such conflicting situations.

10.1.2 The Solution

The main cause of these issues is that the clustering algorithm *fixes* combinations of groups into cluster patterns. The timetabling algorithm must use these patterns, unable to recombine groups if a conflict occurs: the timetabling algorithm lacks some flexibility. As we have seen in Section 6.3.2, the currently used software does not have this problem, as it combines groups in the timetabling phase, and not in the clustering phase. By using the so-called lines in the clustering phase, they make sure that good combinations of groups exist.

Inspired by this approach, I decided to try something comparable. The responsibility of the clustering algorithm is to assign students to groups in such a way that good combinations of groups could be made *in the timetabling phase*. This means that I have to adapt the clustering model, since it now has a new objective. Moreover, I also have to extend the possibilities of the timetabling algorithm: it now has to be able to choose how to combine the groups into cluster patterns. How to adjust the two models is described in Section 10.2 and 10.3.

10.2 Adjustments to the Clustering Model

The primary objective of the clustering problem in the old approach, was to minimize the total number of cluster patterns, covering all multiplicities in the lessons table. In the new approach, I do not need a set of patterns covering the lessons table, but a large set of *all* cluster patterns that are feasible given the group assignments: the timetabling algorithm now has the responsibility to choose patterns from this set such that each group is given the correct number of times in a week. The clustering objective now is to make group assignments in such a way that I have a *good* set of feasible patterns.

When do I regard this big set of feasible cluster patterns 'good'? First of all, it has to contain a large number of patterns, because this increases the flexibility in the timetabling phase. Secondly, it is desirable to have patterns that have a large number of groups: using such large patterns reduces the number of time slots needed to cover all multiplicities from the lessons table. Usually, I have to balance these two properties, since optimizing one property restricts the possibilities for the other one.

I could still use the old objective of minimizing the total number of patterns covering the lessons table: I then disregard the set of patterns of the solution, but use the group assignment of the solution to generate the big set of feasible cluster patterns and feed it to the timetabling problem. This big set probably contains some large patterns, since the old objective tries to fit a fixed amount of groups in a minimum number of patterns. However, the number of feasible patterns is not necessarily large. Also, constraint (1) (respecting the lessons table) is not needed in the clustering phase anymore, since the timetabling algorithm now gets the responsibility to respect the multiplicities of the subjects: this constraint is unnecessarily restricting. Therefore I

remove constraint (1): $\sum_{c \in C^p} x_c c_g = m_g^p \quad$ for all g, p.

Without this constraint, the old objective becomes useless (minimizing the number of patterns would then result in no patterns at all). In the next sections I describe some possibilities for objectives I could use to obtain good pattern sets. These sections have titles taken from graph theory. I can translate the problem to a set of graphs (one for each period) as follows: for each group that must be given in the corresponding period, introduce a node in the graph. Two nodes are connected by an edge if they could be given at the same time (that is, if they could be part of the same cluster pattern). This of course depends on the group assignments, so the edges of this graph are not predetermined. However, this translation gives some insight in what I try to achieve.

10.2.1 Minimum Clique Cover

This objective is inspired by the currently used approach of minimizing the number of 'lines' (see Section 6.3.2). I try to form, for each period, a set of cluster patterns (lines) that cover all groups *once*, and the objective is to minimize the number of such patterns needed. In the corresponding graph, this means that I am minimizing the number of cliques (cliques correspond to feasible patterns) that entirely cover the graphs. More precisely, I want the group assignments (determining the edges of the graphs) to be in such a way, that the graphs allow a minimum clique cover (MCC). This approach differs from the currently used approach in that it is period dependent: each period may have different patterns. Therefore, I do not need to care about alternating subjects (see Section 6.3.2). This approach meets at least one of the desired properties: I produce (at least a few) large cluster patterns. A large *number* of feasible patterns (cliques) is however not guaranteed.

To implement this approach to the model, I only have to add a constraint that makes sure each group has to be covered once:

$$\sum_{c \in C^p} x_c c_g = 1 \quad \text{for all } g, p \quad (1')$$

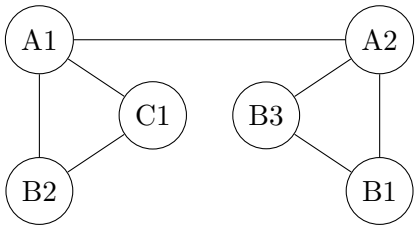


Figure 10.1: Example solution of the MCC objective

I can still use constraints (3) to balance the number of lines over the periods:

$$\sum_{c \in C^p} x_c \leq \hat{n}^p \quad \text{for all } p \quad (3)$$

The objective is to minimize the total number of cluster patterns (lines) needed. I can still try to keep the group sizes as close to their ideal size as possible as a secondary objective. This yields the exact same objective as I used in the old approach:

$$\min \sum_c x_c + c_z \sum_g (z_g^+ + z_g^-) \quad \text{(obj)}$$

Figure 10.1 gives an optimal solution for this objective of a simple example with 3 subjects (only one period is shown). Subject A has two groups (A1 and A2), subject B has three (B1, B2 and B3) and subject C has only one (C1). The optimal group assignment allows this graph to be covered by only two cliques: A1-B2-C1 and A2-B1-B3.

10.2.2 Maximum Number of Edges

The previous objective focused on the size of the patterns, this objective concentrates on the number of feasible patterns. One easy and natural way to allow a large number of patterns, is to make the graph as large as possible, that is, to have as many edges as possible. Of course, this is the same as maximizing the number of feasible patterns consisting of two groups, but it hopefully also creates larger cliques (larger cluster patterns) in the graph.

This objective is very easy to implement. First of all, I only need to consider patterns consisting of two groups. Therefore I have to

> adapt the column generation algorithm to only generate cluster patterns with two groups.

Secondly, it is useless to impose upper bounds on the number of patterns per period, since I am *maximizing* the number of patterns. Therefore, I

> remove constraints (3).

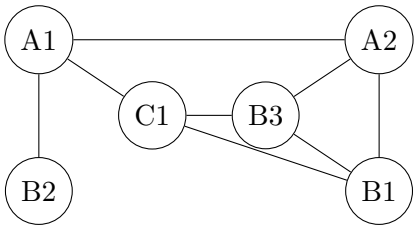


Figure 10.2: Example solution of the MNE objective

I could however impose *lower* bounds on the number of feasible edges in order to balance this number over the periods:

$$\sum_{c \in C^p} x_c \geq \check{n}^p \quad \text{for all } p \quad (3')$$

Finally, I have a new objective. The main goal is to maximize the number of feasible cluster patterns, and I still have the secondary objective to keep the group sizes as close to their ideal size as possible. Since I am now maximizing instead of minimizing, I have to *subtract* the term $c_z \sum_g (z_g^+ + z_g^-)$:

$$\max \sum_{c \text{ with 2 groups}} x_c - c_z \sum_g (z_g^+ + z_g^-) \quad (\text{obj'})$$

Because I restrict the column set to columns with two groups, the model becomes relatively small. This results in low running times, which is a big advantage.

Figure 10.2 shows an optimal solution of the same example as in Figure 10.1, now using the maximum number of edges (MNE) objective. As we can see, I have 8 edges in this solution instead of 7 in the solution of the MCC objective. Just like in the MCC solution, there are two feasible patterns consisting of three groups (A2-B1-B3 and B1-B3-C1). However, we now need three cliques to cover the graph, instead of two in the MCC objective.

10.2.3 Other Remarks

The previous two objectives mainly focus on one of the two desired properties for the resulting set of feasible patterns. There are lots of possibilities that try to balance the two. For instance, I could maximize the number of 3-cliques, 4-cliques or maybe n-cliques (instead of the number of edges: 2-cliques). Another option is to maximize the total number of feasible patterns, possibly weighing them with the number of groups per pattern. Which approach works best depends on the situation, and on what exactly I am trying to achieve. Let us regard this as further research.

pattern	#parents
A2-B1-B3	3
A1-A2	2
A1-B2	
A1-C1	
A2-B1	
A2-B3	
B1-B3	
B1-C1	
B3-C1	
A1	1
A2	
B1	
B2	
B3	
C1	

Table 10.1: Feasible patterns: children

parent	children
A1	A1 A1-A2 A1-B2 A1-C1
A2	A2 A1-A2 A2-B1 A2-B3 A2-B1-B3
B1	B1 A2-B1 B1-B3 B1-C1 A2-B1-B3
B2	B2 A1-B2
B3	B3 A2-B3 B1-B3 B3-C1 A2-B1-B3
C1	C1 A1-C1 B1-C1 B3-C1

Table 10.2: The parents with their children

Also note that I could try different strategies (objectives) iteratively, and use decomposition techniques (see Section 9.2) in order to improve solution quality and running time.

10.3 Adjustments to the Timetabling Model

The timetabling model has to be adapted to be able to handle a large set of feasible cluster patterns, instead of a set of patterns that serve as ready-to-use lessons. Therefore, I introduce the concept of *parents* and *children*. A parent corresponds to a single group of a subject, for which I know its teacher, classes (from the group assignments), and multiplicity (from the lessons table), plus properties like double hours, location and classroom type. The children correspond to feasible patterns. A parent usually has several children (all patterns in which this group appears), and a child has one or more parents (one for each group in this pattern).

Consider the example from Figure 10.2. Table 10.1 shows all feasible patterns (the children, this is input from the clustering phase), and Table 10.2 shows all 6 parents with their children.

Parents and children can both be seen as schedulable objects, that are similar (but not exactly) to lessons. I can schedule at most one (from the set of all feasible patterns) child at some time slot i: then automatically all its patterns are also scheduled at time slot i. In the ILP timetabling model, there are only variables y_{li} for scheduling lessons:

$$y_{li} = \begin{cases} 1 & \text{if } l \text{ is scheduled at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

To extend functionality to also be able to handle children and parents, I need additional variables y^c_{li} for children and y^p_{li} for parents (abusing notation a

bit, l now can represent a 'normal lesson', a child and a parent):

$$y^c_{li} = \begin{cases} 1 & \text{if child } l \text{ is scheduled at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

$$y^p_{li} = \begin{cases} 1 & \text{if parent } l \text{ is scheduled at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

These variables together replace the old variables y_{li} for the ready-to-use cluster patterns l, which could simply be treated as normal lessons. The following additional constraint links the two types of variables.

$$y^p_{li} = \sum_{\tilde{l} \text{ child of } l} y^c_{\tilde{l}i} \quad \text{for all parents } l, i \quad (10)$$

These constraints make sure that if some child $\tilde{l}$ is scheduled at time slot i, so must be all its parents. Conversely, if a parent l is scheduled at time slot i, so must be exactly one if its children.

But how to install these variables in the existing timetabling model (see Section 3.3)? The concepts multiplicity (constraints (3)), fixing (constraints (4)), double hours (constraints (5abc)), spreading (constraints (6ab)) and classroom constraints (constraints (7)) apply to single groups, that is, to the parents. Therefore, I

add constraints (3), (4), (5abc), (6ab) and (7) for variables y^p_{li}.

On the other hand, I can not include parent variables y^p_{li} to constraints (1) (recall from chapter 3 that these constraints link variables x_c and y_{li}):

$$\sum_{c \in C_{jd}} x_c c_{ib} - \sum_{l \in L_{jb}} y_{li} = 0 \quad \text{for all } j, d, i \in I_d, b \quad (1)$$

This is because more than one parent may be scheduled at the same time if they are grouped in a pattern/child. This is no problem for the teachers (since all groups in a pattern must be given by different teachers), but it is for classes: the classes sets of the groups/parents in a pattern usually overlap and subtracting more than one positive y^p_{li} in the left-hand side of this constraint would make it infeasible. This is why I split these variables in parents and children in the first place. Since only one child/pattern may take value 1 per time slot, I

include variables y^c_{li} in the corresponding constraints (1).

The term 'corresponding constraints' means the constraints for all teachers and classes involved in this pattern (that is, the union of the teachers and classes sets of all its parents). By installing the parent and child variables into the correct constraints, and linking them by the new constraints (10), I have achieved that the timetabling algorithm is able to choose how to combine the single groups into cluster patterns. Of course, this makes the model bigger, since there are a lot more variables, but also increases flexibility.

Chapter 11

Results

Introduction The models described so far look good in theory, but how do they perform in practice? Of course, I have done a lot of experiments to find out how good the produced solutions are and how long it takes to achieve such a solution (that is, the running times). By means of a number of tables and figures with accompanying texts, the results of these experiments are presented in this chapter. I only give factual information in this chapter, conclusions are drawn in Chapter 12.

All experiments were done on a system with an Intel Core2Duo E6850 (2×3 GHz) processor and 2 GB of memory. Although my modeling programs were programmed in JAVA (see Appendix A), the actual computations were performed by ILOG CPLEX 11 (see [4]), a high performance mathematical programming optimizer written in C.

11.1 Clustering

11.1.1 Input

As I explained in Chapter 6, clustering is done for each set of classes of the same type. At the HLC, there are 7 types. To give an idea of the sizes of the instances, Table 11.1 gives for each type the number of students and stem classes. Moreover, I specify per period (there are 5 periods in a year at the HLC) the total number of groups (of all subjects) and feasible patterns.

As we can see, the number of feasible patterns of course depends on the number of groups, but even more on the number of students. This is because a larger number of students allows patterns containing a larger number of groups, and more combinations can be made with this larger number of groups. The biggest data set is 4HAVO with 6 stem classes. For periods 2,4 and 5, the patterns generation algorithm failed to produce all patterns within a reasonable amount of time (one hour) for this instance and eventually crashed, so these numbers are unknown but surely exceed 100000. The number of groups and feasible patterns for 4VMBO is the same for each

			#groups					#feasible patterns					
type	#classes	#students	p1	p2	p3	p4	p5	p1	p2	p3	p4	p5	total
3VMBO	4	101	14	14	10	14	19	263	826	215	263	2331	3898
4VMBO	4	95	18	18	18	18	18	1470	1470	1470	1470	1470	7350
4HAVO	6	147	28	29	22	32	35	86291	?	26200	?	?	?
5HAVO	5	132	26	25	25	27	23	10342	8778	8740	13333	9285	50478
4VWO	2	56	12	13	15	10	11	125	157	266	42	89	679
5VWO	2	54	18	17	20	20	16	53	409	297	521	233	1513
6VWO	3	77	20	22	21	24	21	281	1080	558	1318	1070	4307

Table 11.1: Instances

period. This is because in each period, all subjects must be given (that is, the lessons table does not contain any zeros). Although the multiplicities for period 5 are a bit different to those of periods 1 through 4, this does not affect the number of feasible patterns, of course.

Almost all input information from the problem definition in Section 6.4 was provided by the HLC. However, there were still a few things that I had to guess:

- The ideal group sizes: I tried to balance all groups evenly.
- The hard bounds on the group sizes: I chose to allow a deviation from the ideal group sizes of a few students (about 4, this differs per subject depending on the total number of subscribed students to this subject).
- Amount of overlapping teacher availability: I decided that the teachers in a cluster pattern must have at least 1 overlapping time slot at which they are all available.
- Classroom constraints: I decided that at most two groups of the same subject may appear in one cluster pattern.
- Cluster patterns per period: I did not use any bounds on the number of patterns.

11.1.2 Minimum Number of Patterns

The primary objective described in Chapter 7 is to minimize the total number of patterns covering all multiplicities of the groups. There is also a secondary objective: to minimize the deviation from the ideal group sizes. In order to make this a secondary objective, I multiplied each unit of group size deviation (GSD) by $c_z = 0.01$ (assuming their sum does never exceed 100, which is always the case): the decimal part of the objective value now represents the group size deviation. Table 11.2 shows for each type the best solution that I could find using this objective.

type	objective	LB	#patterns					GSD
3VMBO	57.21	55.13	12	12	9	12	12	21
4VMBO	79.39	=79.39	15	15	15	15	19	39
4HAVO	121.14	?	24	24	19	26	28	14
5HAVO	114.09	101.09	23	23	24	27	23	9
4VWO	97.06	=97.06	19	19	21	19	19	6
5VWO	133.01	=133.01	28	27	29	29	30	1
6VWO	120.04	116.04	21	25	25	26	23	4

Table 11.2: Best Solutions

			running time (seconds)			
type	period	objective	building	patt. gen.	solution	opt. proof
3VMBO	1	12.12	0.100	0.874	< 1	< 1
4VMBO	1	15.39	0.064	9.162	811	1483
4HAVO	3	*19.14*	0.736	504.246	4765	n/a
5HAVO	1	23.09	0.172	78.015	75909	75909
4VWO	1	18.04	0.542	0.611	6	35
5VWO	1	28.00	0.148	0.049	< 1	< 1
6VWO	1	21.02	0.080	0.456	< 1	< 1

Table 11.3: Running times, solving 1 period

I obtained these solutions by using several iterative runs with different objectives, fixations etc. Therefore, it is impossible to have useful running times that can be used in comparisons. I can roughly say that they lie between 1 hour (for 4VWO) and 1 day (for 4HAVO), although obtaining reliable lower bounds could take longer. The lower bounds in the third column of the table are simply obtained from the CPLEX log. We can see that three of the problem sets are solved to optimality, 3VMBO and 6VWO are at least close to optimality and the solution of 5HAVO is still some distance away from its lower bound. For 4HAVO, I do not have a lower bound since it so happened that not all patterns could be generated. The solution with objective 121.14 thus had to be found using a restricted set of columns, and decomposition techniques.

One Period

The first timed experiments were to solve only one period of each type. I chose to solve period 1, except for the large instance 4HAVO: period 3 has much fewer feasible patterns (see Table 11.1) and therefore runs faster. It is also the only one I did not solve to optimality: after about 20 hours the search tree of CPLEX became too big, and the solving process stopped. Long before then (after 4765 seconds) it found the solution with objective value 19.14.

Table 11.3 shows several running times. First of all, we see that building the models (initializing variables and constraints), can be done very quickly:

type	period	objective	running time (seconds) normal	CG
3VMBO	1	12.12	< 1	1
4VMBO	1	15.39	811	360
4HAVO	3	*19.14*	4765	6308
4HAVO	3	17.14	n/a	18719
5HAVO	1	*25.09*	3812	1177
5HAVO	1	23.09	75909	73275
4VWO	1	*20.04*	5	< 1
4VWO	1	18.04	6	20
5VWO	1	28.00	< 1	< 1
6VWO	1	21.02	< 1	< 1

Table 11.4: Running times, solving 1 period

within a second. The generation of all feasible patterns can take longer for the bigger models: it took more than 8 minutes to generate all 26200 feasible patterns of period 3 of 4HAVO. The time CPLEX needed to find the solution varies a lot: from under a second to over 21 hours (for 5HAVO). Sometimes, the optimizer needed extra time to proof optimality of the solution, as is the case with 4VMBO and 4VWO.

Comparing these objectives with the number of patterns in the solutions for the complete models (see Table 11.2), we see that only the optimal 18 patterns of 4VWO, when I only solve period 1, is too optimistic if I want to solve the complete model: in the optimal solution of the complete 4VWO instance, I need 19. Also, the group size deviations do not deviate that much.

Column Generation

In Section 9.1, I described a column generation scheme intended to find good solutions more quickly. Table 11.4 compares the running times from Table 11.3 with the running times using this column generation (CG) scheme.

We can see that for the smaller models, the CG is probably not a good idea: sometimes the normal optimization process finds an optimal solution even quicker (e.g. 4 VWO). This is because in the CG scheme, every time a new solution is found new patterns have to be generated. This relatively takes a lot of time for the smaller models. A solution for this issue could be to generate all feasible columns once in the beginning, put them in a column pool, and add columns from this pool (which is in the memory) to the model when needed. We can see that for the bigger models (e.g. 4VMBO, 5HAVO), the CG scheme *does* pay off: it finds the optimal solution quicker. This effect is even bigger for *nearly* optimal solutions: see objective 25.09 for 5HAVO and objective 20.04 for 4VWO. In the biggest model 4HAVO,

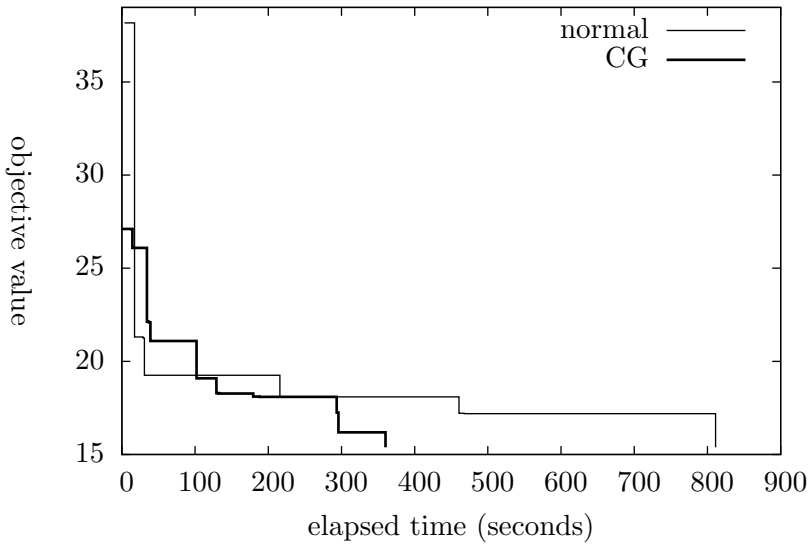


Figure 11.1: 4VMBO objective value against running time

something interesting happened: using the CG scheme I did find the optimal solution with objective value 17.14, while the normal solving process ran out of memory.

Figure 11.1 shows how the objective value of the 4VMBO instance improved in the course of time. Timings for both the 'normal'- and the column generation strategy are displayed. I included this plot to show how the solving process works, and of course to compare the two methods. Obviously, the first feasible solution is found very quickly (in fact, within a second), but the quality of the first solution of the CG approach is much better. However, in the normal approach the solution improves much more quickly than the CG approach and soon (after about 17 seconds) takes the lead. Later on, the CG approach catches up and ultimately finds the optimal solution much quicker than the normal approach.

Fixing Student Assignments

I can use the solution for one period to cluster all periods. I have tried to do this by fixing the student group assignments, see Section 9.2.2 (still using the same objective). In this experiment, I did not allow any switching ($n_g^{\text{switch}} = 0$ for all g), except for 4VWO: I used the 4VWO instance to show the effect of allowing to switch on solution quality and running time. Table 11.5 shows the results. All objectives in this table are proven to be optimal solutions.

The performance of this method is quite good and quick for most of the instances: it even finds the *optimal* solution for 4VMBO and 5VWO. For

type	n_g^{switch}	objective [best known]	#feasible patterns [total]	running time (seconds)
3VMBO	0	59.13 [57.21]	1577 [3898]	310
4VMBO	0	79.39 [79.39]	565 [7350]	< 1
4HAVO	0	-	-	-
5HAVO	0	114.15 [114.09]	1344 [50478]	8
4VWO	0	103.06 [97.06]	236 [679]	1
4VWO	1	101.06 [97.06]	238 [679]	64
4VWO	2	100.10 [97.06]	282 [679]	47
4VWO	3	100.06 [97.06]	315 [679]	34
5VWO	0	133.01 [133.01]	1513 [1513]	25
6VWO	0	126.04 [120.04]	1876 [4307]	101

Table 11.5: Fixing group assignments of one period, solving all

4VMBO, this can be explained by the fact that in each period all subjects are given, so a good solution of one period is also good for the other periods. The performance of 5VWO has another reason: in this case there are only few courses with more than one group, so fixing assignments does not do that much. This is also reflected in the fact that all 1513 patterns are generated. For all other instances, the number of patterns generated is less than without the fixed assignments. This is because the student fixations constraints are also added to the ILP feasibility check (see Section 8.1.2).

Unfortunately, I could not solve 4HAVO this way: there were too many subjects that are *not* given in the fixed third period, so the group assignment for these subjects still had to be determined. This resulted in a set of feasible patterns that was still too large to handle.

Allowing more flexibility by increasing n_g^{switch} clearly shows its effect for 4VWO: the solution quality increases together with the number of feasible patterns. Of course, completely fixing all student assignments results in a far lower running time than allowing the solver to 'play around' with the variables y_{ig}. Compared to $n_g^{\text{switch}} = 1$ however, we can see that increasing flexibility to $n_g^{\text{switch}} = 2, 3$ has the effect that I found better solutions *in less time*.

Fixing Patterns

Besides fixing group assignments, I could also fix patterns. I tried to use this approach to again cluster all periods, fixing some patterns from the solution of the first period. Recall from Section 9.2.1, that by adding these fixed patterns to the ILP feasibility check of the CG algorithm, I make sure that these patterns remain feasible. I chose to fix patterns in which at least 85% of the students have class. Using this approach, solving to optimality took too long for some of the instances, so I decided to abort the optimization

type	#patterns fixed	objective [best known]	#feasible patterns [total]	running time (seconds)
3VMBO	2	59.15 [57.21]	2949 [3898]	493
4VMBO	4	79.39 [79.39]	600 [7350]	< 1
4HAVO	-	-	-	-
5HAVO	-	-	-	-
4VWO	2	101.10 [97.06]	532 [679]	294
5VWO	2	133.01 [133.01]	1513 [1513]	25
6VWO	2	125.04 [120.04]	4307 [4307]	190

Table 11.6: Fixing patterns of one period, solving all

type	objective	LB	#patterns				
3VMBO	20.21	18.13	4	4	4	3	5
4VMBO	20.39	=20.39	4	4	4	4	4
4HAVO	39.14	?	7	8	6	9	9
5HAVO	36.09	30.09	8	7	7	7	7
4VWO	31.06	28.06	6	6	7	6	6
5VWO	43.05	=43.05	10	8	9	9	7
6VWO	40.04	39.04	8	8	8	9	7

Table 11.7: Minimum clique cover objective

process after 10 minutes. Unfortunately the 4HAVO and 5HAVO instances appeared to be too big to produce a reasonable solution within this time.

The results in Table 11.6 are comparable to those fixing group assignments. The solving process takes a bit longer, but produces slightly better solutions for 4VWO and 6VWO. Generally, the number of feasible patterns is a bit higher, especially for 6VWO: fixing the 2 patterns with the highest load factor, all 4307 patterns are still feasible. Apparently, fixing these two patterns did not reduce flexibility at all in this case.

11.1.3 Minimum Clique Cover

In Chapter 10, I described a new approach and proposed some new objectives that I could use. One of them was to minimize the number of cliques covering the graphs that I could construct from the instances (see Section 10.2.1). Solving the complete models (all periods) with this objective takes a lot of time, especially for the big models. Therefore, I again used several iterative runs with different objectives, fixations etc. to obtain these solutions, and I do not have useful running times. I also do not have a lower bound for 4HAVO, since it was impossible to add all patterns to the model. Table 11.7 shows the results.

Except for 4HAVO and 5HAVO, we can see that I can get optimal—or at least near-optimal—solutions for this objective. The solutions of the currently used clustering software have 7 lines for 4HAVO and 6 for 5HAVO (not period dependent!), implying that a solution with at most 35 and 30

type	objective	#patterns [total]	running time (seconds) solution	optimality proof
3VMBO	281.73	464 [3898]	19	136
4VMBO	364.65	690 [7350]	342	1919
4HAVO	*1090.64*	2018 [?]	3600	n/a
5HAVO	*639.75*	1330 [50478]	3600	n/a
4VWO	118.74	327 [679]	225	2686
5VWO	192.87	483 [1513]	97	200
6VWO	383.86	747 [4307]	9	74

Table 11.8: Maximum number of edges objective

patterns respectively must exist for the MCC objective.

11.1.4 Maximum Number of Edges

An other proposed objective was to maximize the number of feasible patterns with two groups (or, equivalently, the number of edges in the graph). Unlike for the MCC objective, I was able to solve most instances to optimality within a reasonable amount of time. The reason for this is that only patterns with two groups are included in the model. Table 11.8 shows the results.

The 4HAVO and 5HAVO solving processes were aborted after one hour, and the displayed solutions may not be optimal: they are within 33% and 20% of the optimal solution, respectively. From the table, we can read that generally the optimal solution is found quite quickly, but the optimality proof takes much longer. Extrapolating this observation to the 4HAVO and 5HAVO instances, I am confident that the displayed solutions are optimal or close to optimal.

11.1.5 Objective Comparison

We have seen the results for the different objectives, but which solution do I use as input for the timetabling model? In Section 10.2, I argued that a solution is good if the corresponding group assignment allows a feasible set of cluster patterns that is big and contains large patterns. Figure 11.2 shows the number of feasible patterns for the three objectives (MNP stands for 'minimal number of patterns', corresponding to the solutions from Table 11.2). Figure 11.3 shows the number of *large* feasible patterns for the three objectives. When to call a pattern 'large' is a bit arbitrary and of course depends on the instance. How many groups a pattern must consist of to be regarded as a large pattern is also displayed in the figure: e.g. a 3VMBO pattern is large if it has at least 4 groups.

Like I could expect, the MNE objective produces the largest number of feasible patterns in all instances, whereas the MCC objective allows the lowest number of feasible patterns. Also, if we look at big patterns, MNE

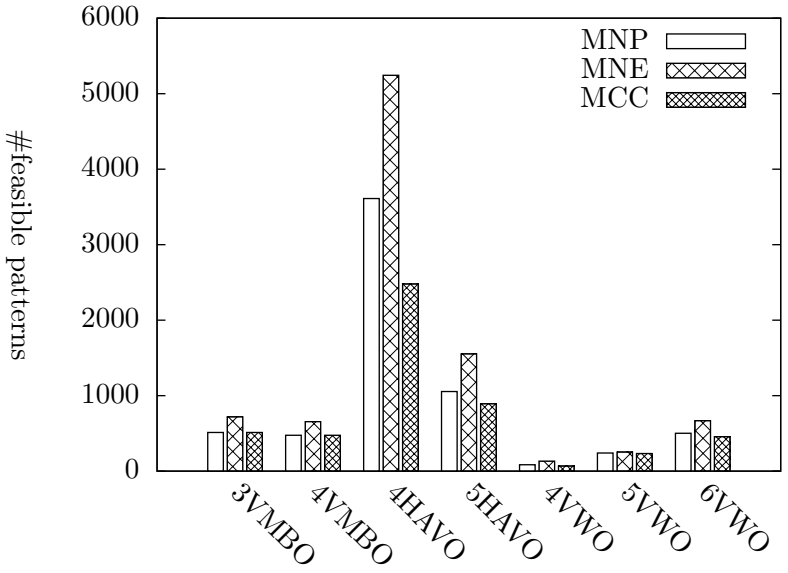


Figure 11.2: Number of feasible patterns

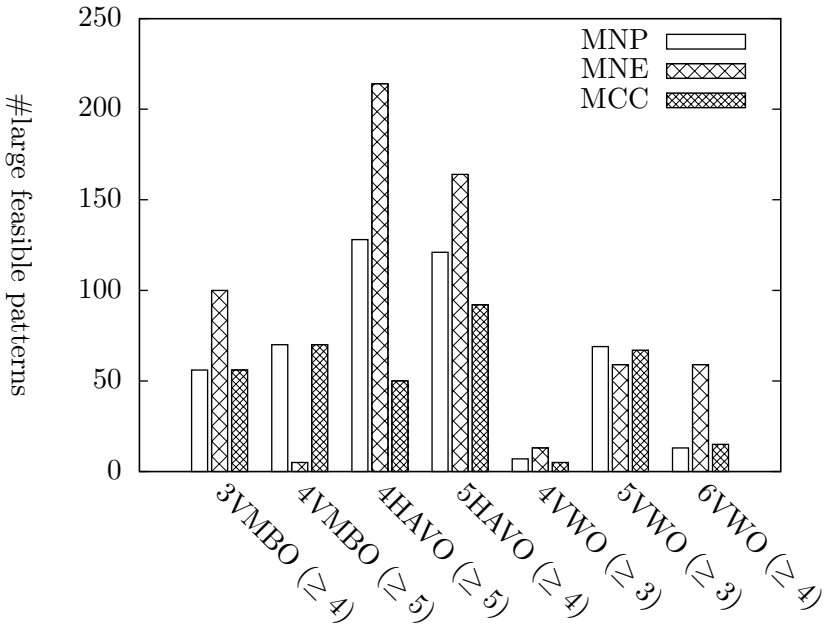


Figure 11.3: Number of large feasible patterns

	number of patterns		group size deviation	
type	HLC	MNE	HLC	MNE
3VMBO	20	20	27	21
4VMBO	20	20	38	39
4HAVO	35	39	34	14
5HAVO	30	36	9	9
4VWO	40	31	11	6
5VWO	45	43	14	5
6VWO	40	40	5	4

Table 11.9: Quality comparison

is the winner in 5 of the 7 instances. For 5VWO, it is not that bad, but for 4VMBO it is. Since there are not too many time slots available (in the timetabling problem) for scheduling lessons for VMBO classes (for instance, they are unavailable the 8th and 9th time slot each day), I really need large patterns. Therefore, I have chosen to use the set of feasible patterns from the MNP objective in the 4VMBO instance. In all other instances, I think it is best to use the pattern set from the MNE objective. One could expect that the MCC objective would produce large patterns, but the figure shows otherwise. The reason for this is that for the MCC objective, I only need a *few* large patterns (cliques) that cover all the groups.

11.1.6 Solution Quality

It is difficult to compare my solutions with the solutions from the clustering program of the HLC for several reasons. One of them is of course that they use a different objective (minimizing 'lines'). Another reason is that their solution sometimes is based on other input data than mine: they are from different moments in time, apparently. For instance, my 5VWO instance included the subject 'Greek', whereas this subject did not appear in the solution from the the HLC's clustering program. Also the number of students often differed.

To give an idea of how well my clustering software performs, I still compared the HLC solutions with my solutions. I used the outcomes of the MCC objective for this purpose, because this objective comes closest to the objective of the HLC. To make a fair comparison between the number of 'lines' and the number of patterns, I multiplied the number of lines by 5.

As we can see, the solutions of the smaller models are comparable as far as the number of patterns is concerned. My 4VWO solution is significantly better. Unfortunately, there is much room for improvement for the HAVO instances. For the group size deviation, my solutions are at least as good as the HLC's.

no.	description	#classes	#teachers	#lessons	#lesson hours	#children
1	PE, difficult lessons	59	141	97	237	
2	4HAVO	6	34	58	158	807
3	5HAVO	5	24	41	112	352
4	34VMBO	8	29	75	209	241
5	456VWO	7	33	68	188	240
6	1st classes	12	54	132	402	
7	2nd classes	13	24	145	377	
8	3HAVO/VWO	8	21	95	234	
complete model		59	145	711	1917	

Table 11.10: Stages with characteristics

11.2 Timetabling

11.2.1 Input

Because it is (yet) impossible to solve the complete HLC instance with all its lessons at once, I had to decompose the model into stages (see Section 5.1.1). When a stage is being solved, all lessons scheduled in previous stages are fixed, so I end up with a complete schedule in the end. It is wise to schedule difficult lessons in an early stage, to avoid infeasibility problems. Deciding which lessons to regard as 'difficult', requires some experience. In general, lessons that have a large classes or teachers set fall into this category. Therefore, it is a good idea to schedule upper school classes (that have a lot of cluster patterns) before lower school classes.

The first stage consists of all the meetings (large teachers set) and some (rare) lower school cluster patterns that involve a large number of lower school classes. I also included the PE lessons in this first stage, since there are so many of them, that the PE classrooms almost never have an empty time slot. This can also cause infeasibility problems later on. In total, I partitioned the model into 8 stages. It may be possible to use less than 8 stages, but this of course increases the size of the instances and therefore the running time. After experimenting with different partitionings, I ultimately found that these eight stages were large, but still manageable. Table 11.10 shows these stages with some characteristics indicating the problem sizes.

The complete model thus has 59 classes, 145 teachers and 711 lessons (I do not count children (feasible cluster patterns) as lessons) that must be given about 2.7 hours per week on average (yielding a total of 1917 lesson hours). The presence of children shows that I used the new approach from Chapter 10. In fact, I used the pattern sets from my own clustering, using the MNP objective for 4VMBO and the MNE objective for the other upper school classes (see Section 11.1.5).

All input information from the problem definition in Section 6.4 was provided by the HLC, except the timetable layout preferences of all teachers

and classes. This is because they use a less exact method of penalizing undesirable timetable properties (using sliders, see Figure 2.3). Therefore, I had to devise the penalty vectors myself, making use of some known information (for instance that upper school VMBO classes are preferred to have the first hour off) and my own instinct. I kept the cost model simple (by having a lot of all-zero penalty vectors), but it is easy to extend the cost model when needed. The following costs are incurred:

- for having n consecutive intermediate hours, incur 2^n penalty points.
- for having class the ninth (last) time slot of the day, incur 2 penalty points.
- for having class the eighth time slot of the day, incur 1 penalty point.
- for VMBO upper school classes having class the first time slot of the day, incur 1 penalty point.
- for a positive variable z_{ld} (bad spreading, becomes positive if more than one hour group of lesson l is scheduled on day d), incur $2 * z_{ld}$ penalty points (thus $c_z = 2$).

It is straightforward to translate this cost model to the penalty vectors from Table 4.1: only p^{L}_{jd} and $p^{\#\mathrm{CIH}}_{jd}$ have positive components. I do not use week layout preferences, so constraints (8) are not in my model. I also do not include fictitious classes (see Section 5.1.2) to deal with empty lessons (this will be tested in Section 11.2.4).

11.2.2 Timetabling the Complete Model

This section describes the results and running times of scheduling the complete school in stages. Before I do that, I have to specify how exactly the columns are generated. It is important to start the ILP phase with a set of columns that is big enough to allow a good solution, and small enough to fit in memory and to allow the solver to run at an acceptable speed. From experience, I learned that having about 20000 columns is ideal (although this number of course depends on the instance I want to solve). My column generation scheme is as follows:

- Generate the 10 columns for each teacher/class and each day with lowest cost and use this column set as input for the LP phase.
- Generate during the LP phase columns with lowest (and negative) reduced cost (every iteration, for each teacher/class and day). For every generated column, store neighbor columns in a column pool (see Section 5.2.3).

		#columns				running time (seconds)			solution		
stage	#it.	init.	LP	CP	ILP	LP	solution	ILP	LP	ILP	LB
1	6	20903	21336	23372	34256	130	56	56	553.00	582	582
2	4	14879	14910	15718	22770	115	51773	72000	640.90	725	664
3	8	12698	12841	13522	18964	81	673	1800	851.09	950	891
4	5	13680	13700	14266	19849	76	28570	28570	1042.10	1141	1061
5	4	14215	14267	14683	21069	52	1871	1871	1170.24	1265	1204
6	2	18563	18574	19594	29295	80	35512	66689	1191.00	1293	1193
7	4	13693	13709	14220	19769	71	13950	80804	1429.88	1535	1450
8	4	9177	9191	9454	12554	38	291	496	1619.04	1672	1641

Table 11.11: Results of timetabling the complete HLC model

- After finding an optimal solution to the LP relaxation, add the unique columns from the column pool that are not yet in the model.
- Generate the 20 columns for each teacher/class and each day with lowest cost that are not yet in the model. Use the complete column set as input for the ILP phase.

The numbers 10 and 20 are determined by experience, and produce good column sets. The only purpose of generating columns before the LP phase is to allow a feasible solution (otherwise, the column generation algorithm can not start). In most cases, I could probably lower this value of 10, and still allow a feasible solution: while solving, I observed that the first feasible solution of the LP phase was already very close to the optimal solution, and only few columns had to be generated to obtain this optimal LP solution.

Table 11.11 shows a lot of data. From the second column, we can read that in all instances I needed less than 10 iterations to obtain an optimal solution to the LP relaxation. Columns 3-6 show the number of columns in the different phases of the solving process. They show that during the column generation phase only a few columns had to be generated before it found an optimal LP solution. Also, the column pool contained only a few hundred columns. Adding the 20 best columns for each teacher/class and day determines the final size of the column set: typically about the ideal 20000. Only stages 1 and 6 have significantly more (this can be explained by the number of classes and teachers involved, see Table 11.10), and stage 8 has only 12554.

If we look at the running times (columns 7-9), we see that the LP phase only takes a couple of minutes; this includes generating the columns. Column 8 shows the running time of the ILP solver before I found the solution I ultimately accepted, and column 9 shows the elapsed time before I aborted the ILP solver (except for stage 1, which was solved to optimality in only 56 seconds). These running times look quite erratic. This is because for some stages I decided to abort after a short period of time because I thought the

solution the solver found was good enough (e.g. stage 3), and sometimes I let it run all night (e.g. stages 2 and 7). In any case, it was impossible to solve stages 2-8 to optimality, since this would probably take weeks and the size of the search tree probably exceeds the size of the memory long before then (actually, this happened already after 80804 seconds (over 22 hours) while solving stage 7).

Columns 10, 11 and 12 show the objective value of the (optimal) LP solution, the best ILP solution I found and the lower bound on the ILP solution, respectively. In all cases, the solution found is within 10% of its lower bound. There is probably still some room for improvement, especially for the stages I aborted early (e.g. stages 3, 5), but I did not have the patience to wait for slightly better solutions.

Relaxations

Unfortunately, there were two situations where I had to 'cheat' in order to maintain feasibility. This is nothing to be ashamed of: the currently used timetabling software is often also unable to schedule all lessons, and the timetabler has to relax some constraints to restore feasibility.

The first feasibility problem that I encountered had something to do with lower school classroom constraints. There were three classroom groups in which so many lessons had to be given that they simply could not fit in their rooms. The most obvious example is classroom group NT with 2 classrooms, in which 24 lower school lessons have to be given, totaling 86 lesson hours. Since lower school classes are unavailable the ninth (last) time slot each day, there are only 40 available time slots per week. This means that even when both NT classrooms are occupied all 40 time slots, at most 80 lesson hours can be scheduled in these rooms, showing it is impossible to meet this constraint. In fact, at some time slots, only one of the NT teachers is available, making things even worse.

The solution I used is to allow one extra room to be used for NT lessons. Every time slot this extra room is needed, a penalty is incurred. Recall constraint (7) from the timetabling model:

$$\sum_{l \in L^r} y_{li} \leq n^r \quad \text{for each } r, i \quad (7)$$

Here, r is a classroom group and n^r its number of rooms. I adapt this constraint, by introducing a 0-1-variable q_{ri} for such classroom group r and time slot i:

$$\sum_{l \in L^r} y_{li} - q_{ri} \leq n^r \quad \text{for infeasible group } r \text{ for all } i \quad (7')$$

Adding bounds $q_{ri} \in \{0, 1\}$ and adding the term $c_q \sum_r \sum_i q_{ri}$ to the objective completes the adaptation. This approach is basically the same as constraints

	1	2	3	4	5	6	7	8	9
Mon	unav.	unav.	unav.	unav.	unav.	unav.	unav.	unav.	unav.
Tue	1	1	1	1	**0**	1	1	1	1
Wed	2	2	2	1	1	2	2	**0**	**0**
Thu	2	**0**	1	1	**0**	1	1	1	**0**
Fri	2	2	**0**	1	1	1	1	**0**	**0**

Table 11.12: Tight schedule of teacher Bal

(6), where I allowed deviation of the ideal spreading incurring a penalty c_z. I had to do this for three lower school classroom groups, and I used $c_q = 2$ as unit penalty. When assigning lessons to physical rooms, I would then sometimes have to assign some lessons to inappropriate rooms. There are always enough rooms in the building to fit all classes, so this should not be a big problem.

The other feasibility problem was that after stage 7, there were three teachers whose schedule was so tight that some of the remaining lessons of stage 8 could not be fitted in. I had to remove 4 lessons from my model to restore feasibility. One of them is given by teacher Bal, and the Table 11.12 shows its schedule after the last stage 8 (without the infeasible lesson).

The zeros in the table represent the free hours, the ones and twos stand for scheduled lessons in building 1 and 2, respectively. This teacher is unavailable on Monday. The unscheduled lesson has a multiplicity of 2, is a double hour and must take place in location 1. Since it is a double hour, I need two consecutive free slots. Time slots 8 and 9 on Wednesday is not an option, since the teacher than would have to travel from building 2 to 1 between time slots 7 and 8, but there is no break between these time slots. The only possibility left is Friday, time slot 8 and 9. Unfortunately, it is school policy that no class has a lesson on Friday the ninth time slot. This way, we can see that this lesson can not be scheduled, fixing the lessons from the solutions of previous stages.

The same thing happened to three other lessons. Apparently, the schedule so far for these busy teachers was not good enough, but can not be changed anymore (since all its lessons are fixed at this stage). However, by moving and switching some lessons manually, these four lessons could still be fitted in, resulting in a complete schedule that only violated some of the classroom constraints (but this was unavoidable).

Penalty Partitioning

The costs from the 11th column of Table 11.11 can be partitioned in four parts: the timetable layout costs for the classes, the timetable layout costs for the teachers, the spreading deviation penalties and the classroom deviation penalties. Figure 11.4 shows how these are partitioned for each stage.

The timetable layout penalties for the teachers obviously is the largest

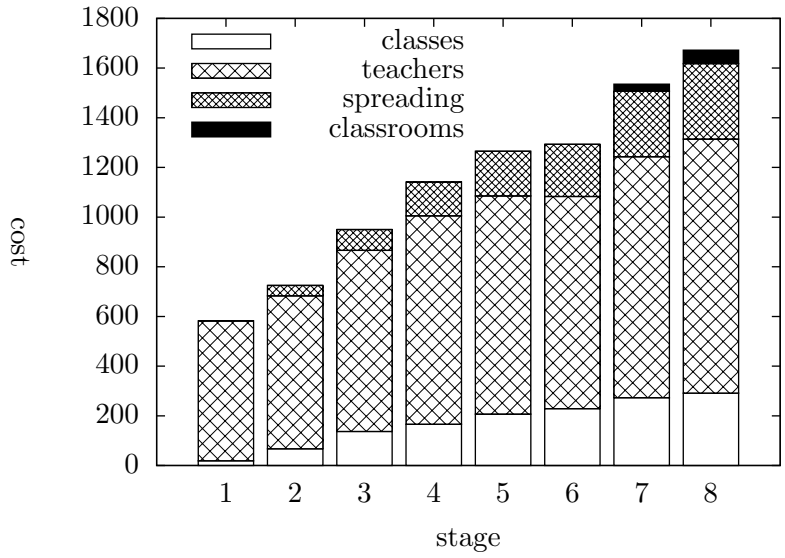


Figure 11.4: Penalty partitioning

part of the total cost. One interesting thing to note is that this cost in stage 5 (878) is higher than in stage 6 (854). This can happen, because a number of intermediate hours of teachers get filled up by the newly scheduled lessons in stage 6. Apparently, this effect was bigger than the newly created teacher costs in this stage. All other costs are constantly increasing. I had to relax classroom constraints only in stages 7 and 8.

Solution Quality

Again, it is difficult to compare my results with the results from the HLC. First of all, the HLC uses a different cost model. Moreover, their schedule includes kwt-hours (see Section 2.1.3), mine does not. Finally, it appeared to be almost impossible to extract information from their schedule about how well the lessons were spread, and when a lesson was scheduled in an inappropriate room. In order to make some kind of fair comparison, I fed all timetable layouts from their schedule to my own program, and extracted their cost, the number of trips and the number of intermediate hours from it. The results are in Table 11.13.

As we can see, my program has much fewer intermediate hours and a much better timetable cost. We must however keep in mind that this comparison is done for the cost model that I invented myself. The number of trips is lower in the HLC solution. This is no surprise since I do not penalize trips at all, nor do I bound the number of them. Also, if I only consider the classes, my program produces a solution with more intermediate hours and a higher timetable cost. The reason for this is probably because

		HLC program	my program
Total	timetable cost	1850	1314
	#trips	87	106
	#intermediate hours	501	325
Classes	timetable cost	249	291
	#trips	2	2
	#intermediate hours	9	40
Teachers	timetable cost	1601	1023
	#trips	85	104
	#intermediate hours	492	285

Table 11.13: Quality comparison

	old approach costs			new approach costs		
	layout	spreading	total	layout	spreading	total
3VMBO	5	6	11 [10]	5	0	5
4VMBO	0	16	16	0	0	0
4VWO	0	4	4	0	2	2
5VWO	0	16	16	3	4	7
6VWO	18	12	30 [28]	0	4	4 [2]

Table 11.14: Approach comparison

I penalize classes the same as teachers, whereas at the HLC, intermediate hours of classes are penalized much heavier.

A quick manual check of the HLC schedules for spreading and classroom violations indicate that they are comparable to the violations in my solution. However, I can not show this with hard numbers.

11.2.3 Performance of the New Approach

The complete school was scheduled using the new approach from Chapter 10, where it is the responsibility of the timetabling solver to combine the cluster groups into feasible cluster patterns. I claimed that this improves flexibility and therefore could avoid spreading penalties (or even classroom group penalties). The old approach produced ready-to-use cluster patterns. This section compares the two approaches. To show that the new approach is indeed an improvement, I tried both methods on small data sets (the lessons for 4VWO, 5VWO, 6VWO, 3VMBO and 4VMBO). I used the patterns from the MNP (minimal number of patterns) objective for the old approach. For the new approach, I used the same large pattern sets as used for scheduling the complete HLC.

Table 11.14 compared the costs for these 5 instances. Because these instances are small, there is of course no effect on classroom constraint violations. Almost all instances are solved to optimality. If not, the lower bound is displayed between brackets. The solving process was aborted after

	no fict. classes		50%		{33%,67%}	
	cost	#SIH	cost	#SIH	cost	#SIH
3VMBO	5	248	9	77	12 [10]	96
4VMBO	0	171	0	61	0	46
4VWO	2	308	2	180	2	178
5VWO	7	466	8	243	6	215
6VWO	4 [2]	560	8 [2]	334	6 [2]	240

Table 11.15: Fictitious classes

half an hours (unless of course the optimal solution was found before then). In all cases, the new approach produces better solutions. This is mainly due to the reduced spreading costs. In the 6VWO instance, also the timetable layout costs are reduced! For 5VWO, the timetable layout costs increased a bit in the new approach, allowing a solution with much less spreading violation.

11.2.4 Fictitious Classes

Section 5.1.2 described a way to deal with 'empty' lessons (cluster patterns with only few students; note that even in the new approach form chapter 10, this is still an issue). I proposed to introduce fictitious classes that only have the 'full' lessons. I did not use this method yet for timetabling the full HLC, but I did some experiments on the same instances as used in Section 11.2.3.

I tried two different partitionings of the lessons. The first partitioning has one fictitious class for each stem class, that is added to the classes set of only the lessons in which at least 50% of the students of the corresponding stem class participate. The second partitioning is finer and has two ficticious classes per stem class: one has all lessons with at least 33% participation, the other one only has lessons with 67% participation. Table 11.15 compares the costs and the number of student intermediate hours (SIH): the sum of the number of intermediate hours of each individual student. Again, when the solution is not (proven) optimal, the lower bound on the cost (consisting of spreading penalties and timetable layout costs for teachers and classes, including the fictitious classes(!)) is displayed between brackets. The solving process was aborted after 12 hours (unless of course the optimal solution was found before then).

The effect is obvious: using fictitious classes reduces the number of intermediate hours for students. The finer partitioning performs slightly better than the 50% partitioning (with the exception of 3VMBO). Sometimes, the cost increases a bit if I use fictitious classes; this is due to the fact that there are of course more classes, all contributing to the total cost.

Chapter 12

Conclusion

Introduction In the introductory Chapter 1, I stated what I tried to achieve with my research. Now that I have covered all theory and presented the computational results, I am ready to draw some conclusions: to what extent did I reach the goals that I set myself? This question will be answered in this chapter, firstly for the clustering problem, and secondly for the timetabling problem. The final Section 12.3 describes some further research that could be done in this area.

12.1 Clustering Problem

We have seen that my clustering algorithm works, and is able to handle all constraints that occur in real-life situations. In fact, classroom and teacher availability constraints are taken into account: this is not the case in the current HLC model. I clustered using three different objectives, differentiating between the 'old' approach, where the objective was to minimize the number of cluster patterns covering the lessons table (MNP objective), and the 'new' approach from Chapter 10, for which I proposed two possible objectives: maximizing the number of edges (MNE) and finding a minimum clique cove (MCC).

Clustering using the MNP objective can be done quickly (within a few hours) for the smaller models. The bigger models (4HAVO and 5HAVO) caused some problems: very large column sets, long running times and solutions that may not be very close to optimality. I had to use several tricks to obtain solutions for these larges instances. One of these tricks was a (non-standard) column generation technique described in Section 9.1. We have seen that this technique could be useful for these larger instances (and is useless for smaller instances). Other tricks I could use are decomposition techniques, where I fix (part) of the student assignment to groups and/or cluster patterns that I like. In particular, (partially) fixing the student assignments proved to be useful, and I used this technique repeatedly to find

good solutions for 4HAVO and 5HAVO.

The MCC objective comes closest to the objective of the HLC minimizing the number of 'lines'. Therefore, I used this objective to compare my solutions to the solutions of the HLC. The group balancing (the secondary objective) was at least as good in my solution as in theirs. The number of patterns was comparable for the smaller models. For the bigger models I experienced the same problems as with the MNP objective; my 4HAVO and 5HAVO solutions were significantly worse.

Fortunately, there is a third objective: maximizing the number of edges (MNE), which is the best of the three objectives, in my opinion: it produces the largest feasible pattern set and also produces enough *large* patterns, in most instances. Also, since it only has to consider patterns with two groups, its running time is quite short: within an hour, it finds an optimal—or near-optimal—solution. However, this running time is no competition for the local search algorithm used at the HLC, which runs in minutes, or even seconds. I still think, one hour is an acceptable running time, given the fact that clustering only has to be done once a year.

Summarizing, I propose to use the MNE objective (or an improved version, see Section 12.3.1) for clustering: it produces good solutions within a reasonable amount of time. Moreover, I think using this objective gives better feasible cluster pattern sets than the objective currently used at the HLC. However, this objective can only be used in the 'new' approach from Chapter 10. For the old approach, I am committed to the MNP objective, which performs well for small instances. For large instances, I need tricks like column generation and group assignment fixation to obtain good solutions.

12.2 Timetabling Problem

The timetabling model covers all important constraints that arise in real-life timetabling. In particular, I think my cost model is versatile enough to include all preferences that a teacher of class could have. However, given the extensiveness of the timetabling software that is currently used, there are probably some fine-tuning issues that are not included in my model, but do play a (minor) role in the HLC's timetabling.

Again, I need to differentiate between the old- and new approach. Some experiments on small instances clearly show that using the new approach improves the quality of the solution dramatically: in particular, costs caused by bad spreading of lessons can be avoided by using this new approach.

I tried to schedule the complete HLC using this new approach. Unfortunately, it is impossible to schedule all 711 lessons at once: I had to decompose the model into stages (I used eight), each stage fixing the result of all previous phases. I have chosen a simple cost model that was easy to work with, but in the end appeared to be simplified too much (for example,

I did not penalize trips and did not penalize intermediate hours for classes heavier than for teachers).

Still, after a total running time of about 70 hours, I managed to produce a complete schedule for the HLC of good quality, in my opinion. The words 'good quality' can be justified by the fact that my schedule contained considerably fewer intermediate hours, and my timetable layout costs are significantly lower. On the other hand, my schedule contained more trips, and if I consider the classes alone, my solution is slightly worse. However, I am confident that these issues can be improved by using a better cost model.

There were a few infeasibility issues along the way. The timetabler at the HLC, Victoire Camps, assured me that this always happens, and some constraints always have to be relaxed. In this case, the classroom constraints proved to be too tight, and I had to allow an extra room for three classroom groups. Also, four lessons could not be fitted in the fixed schedules of the corresponding teachers, but this issue was easily resolved by manually moving some lessons around.

For the complete HLC model, I did not use the technique of adding fictitious classes in order to reduce the number of student intermediate hours (in other words: moving emptier cluster patterns to the beginning or end of the day). I did experiment with this technique on smaller instances, and the results are very promising. Adding fictitious classes decreased the number of student intermediate hours by up to 70%! We must however keep in mind, that adding these classes increases the size of the model, so I can not add too many of them.

Summarizing, it is possible to solve high school scheduling problems using ILP and column generation. Using the new approach is a good idea, just like thinking of a good cost model. The 70 hours it took me maybe seems like a long time at first sight, but is in my opinion still acceptable, keeping in mind that computers get faster and faster. Moreover, the HLC instance is quite big. Although not yet tested on large instances, adding fictitious classes is a good way to avoid having too many student intermediate hours.

12.3 Further Research

12.3.1 Clustering

Let us start with the clustering model: what could be improved? Although the MNE objective works well, I think there must be better objectives that find a better balance between the size of the feasible pattern set and the amount of large cluster patterns in this set: we have seen that in one instance (4VMBO) the optimal solution using the MNE objective produced too few large patterns (see Figure 11.3).

Another thing that could be investigated is how to remove some symmetry from the clustering model that is still present. Suppose for instance that

there is a subject A with two groups A1 and A2, and in some solution a set of students S1 is assigned to A1 and a set of students S2 is assigned to A2. Then there is probably (if there are no group assignment restrictions) also a solution with the same objective value in which student set S2 is assigned to group A1 and S1 to A2, and in every pattern group A1 is replaced by A2 and vice versa. How to lose this kind of symmetry?

12.3.2 Timetabling

One thing I encountered during the timetabling in stages, is that in early stages the schedules for teachers contain a lot of intermediate hours, and they are of course penalized as such. When it is known that in later stages a lot of lessons will be added to this schedule, these intermediate hours are no big problem because they will probably be filled. Maybe we could do something with this information, and use different cost functions for the same teacher in the different stages?

Another thing that could be investigated is whether it is possible to put more information in the columns than only in which building a lesson must take place. Maybe we could include information about whether a double hour has to be scheduled at the corresponding slots (then there must be two such column entries consecutively). Another possibility is to include the concept of full and empty lessons in the columns (see Section 5.1.2). We could be even more specific like including which class to teach. The more specific we are, the more feasible columns we have: this is something to be careful about.

Finally, I am curious about the HLC's method of avoiding too many trips. If a teacher teaches in both buildings, part of his schedule (say Monday and Tuesday) is reserved for lower school lessons, and the remainder is reserved for upper school lessons. We have to be careful with these reservations, because we do not want the situation where too many teachers have reserved time slots on Monday and Tuesday for lower school lessons; then there could be too few teachers to teach lower school classes the rest of the week. I think that this approach is too limiting for the ILP approach (but maybe I am wrong), and only works at the HLC because the local search approach allows them to quickly move some lessons around if such infeasibility issues occur. This is not possible (or at least difficult) in an ILP approach.

12.3.3 Pre- and Post-processing

Before the clustering phase actually starts, some pre-processing has been done: determining the lessons table, which is basically answering the question 'which subjects are taught in which period and for how many hours per week?'. There are several things to consider. Firstly, the total number of lesson hours per class must be balanced evenly over the periods. To a lesser

extent, the same holds for the teachers. Of course, the legal total number of lesson hours per subject must be satisfied. Also, things like classroom types play a role: we can not have too many lessons of the same type in a period, because they have to fit in appropriate classrooms. Solving this problem is currently done by hand, but I think it would be interesting to develop a model that allows us to solve this problem more efficiently.

Another pre-processing matter is the formation of stem classes. For first year classes, this is done based on the preferences of the students (of course, they want to have some friends from primary school in their own stem class). Later on (in particular, before the first year in upper school), new stem classes could be formed, based on the subject set choices of the students: Doing this in a smart way could make clustering a bit easier. I think it would be interesting to investigate how the formation of stem classes is currently done, and how to improve this process.

After solving the timetabling problem, there is one thing left to decide: which lessons must be given in which rooms? The classroom constraints made sure that every (or almost every) lesson can be given in an appropriate classroom, but within a classroom type there may be differences between individual rooms. It is not uncommon that a teacher has his 'own room', in which this teacher gives almost all its lessons. Now that we have a large number of teachers, it may happen that several teachers regard a classroom as their own, an this may lead to conflicting interests. As further research, one could develop a model that assigns the lessons to individual classrooms keeping the teachers as happy as possible (we thus again need cost functions). Note that at the HLC this assignment is done *during the timetabling phase* instead of afterwards.

Part IV

Appendices

Appendix A

Screen Shots from my JAVA Programs

Introduction In order to enter the data in a user-friendly way, and to be able inspect results without having to read long and ill-digested text files, I made a graphical shell for the two JAVA-programs that I wrote. I decided to include some screen shots with comments of both the timetabling- and the clustering program in this appendix, because this gives us some insight in the extensiveness of such programs. We still have to keep in mind that (naturally) my programs are not even nearly as extensive as todays commercial software packages.

A.1 My Timetabling Program

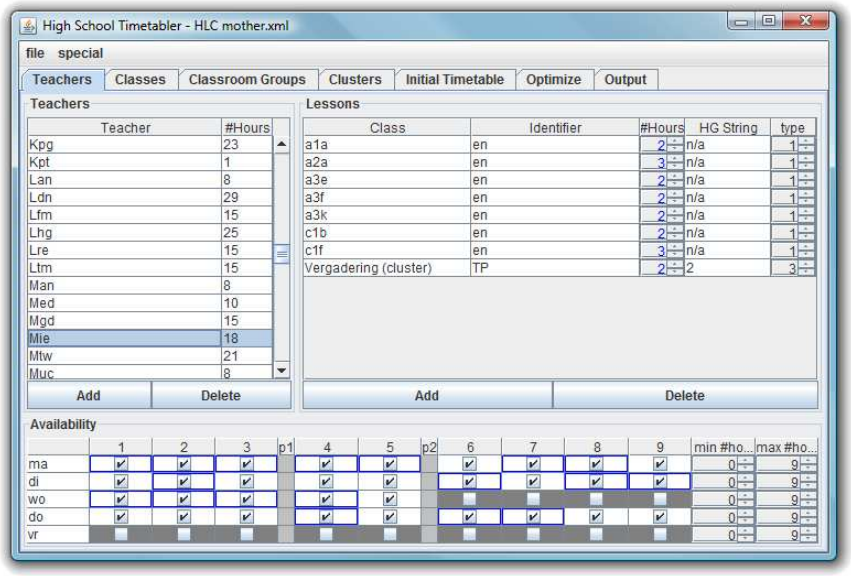


Figure A.1: Teachers tab showing the lessons of (English) teacher Mie. This teacher also has a meeting ('Vergadering'), which is a double hour (according to its hour groups entry '2'). The bottom part shows that he is unavailable on Wednesday afternoon and on Friday.

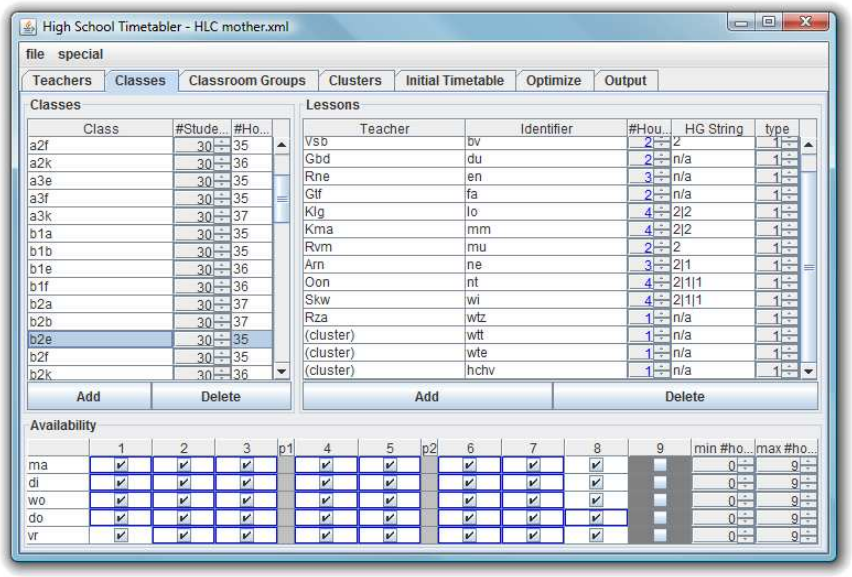


Figure A.2: Classes tab showing the lessons of class b2e. We can see that, although this is a lower school class, it still has some cluster lessons (three to be precise). It also has several double hours.

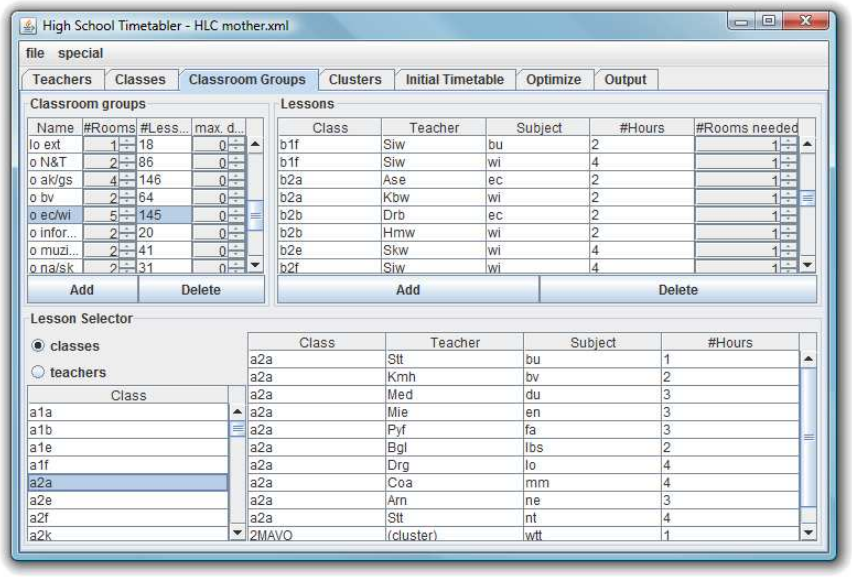


Figure A.3: Classroom Groups tab showing the lower school Economics/Mathematics classrooms: there are 5 of them. Using the lower part ('lesson selector') I can choose the lessons that must be scheduled in one of the rooms from this group.

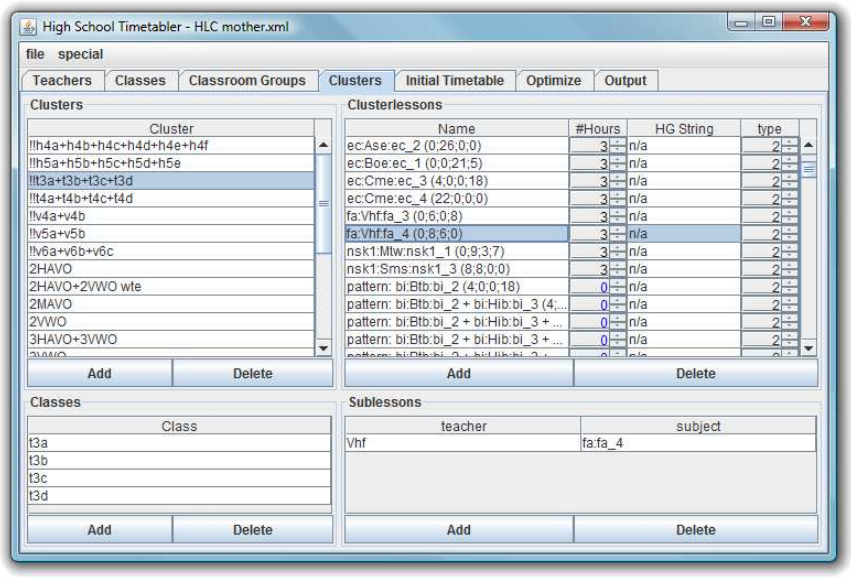


Figure A.4: Clusters tab showing the cluster lessons of the 4 3VMBO classes t3a, t3b, t3c and t3d. Here the new approach from Chapter 10 is implemented: the currently selected cluster lesson fa_4 is one of the parents.

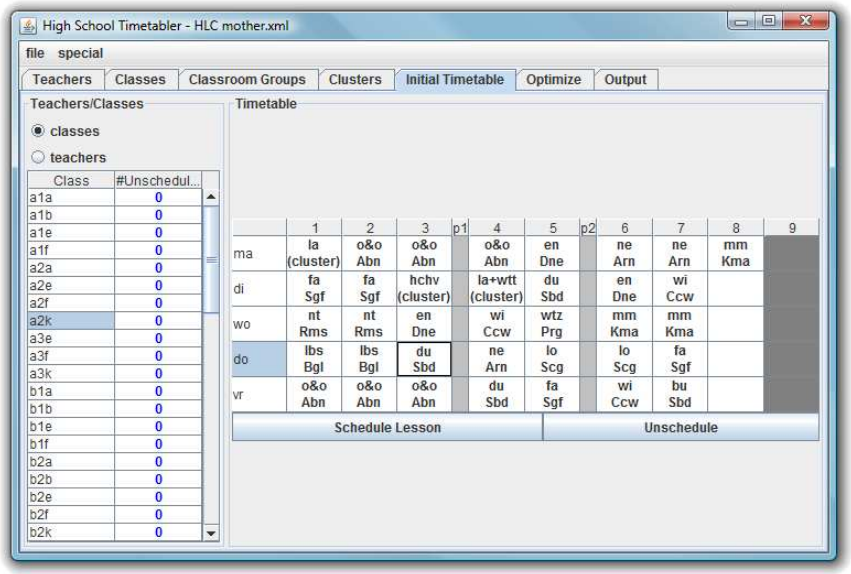


Figure A.5: It is possible to provide an existing schedule to the program that can act as a starting point for the optimization algorithm. Using this tab, the timetable that is currently being used at the HLC has been entered.

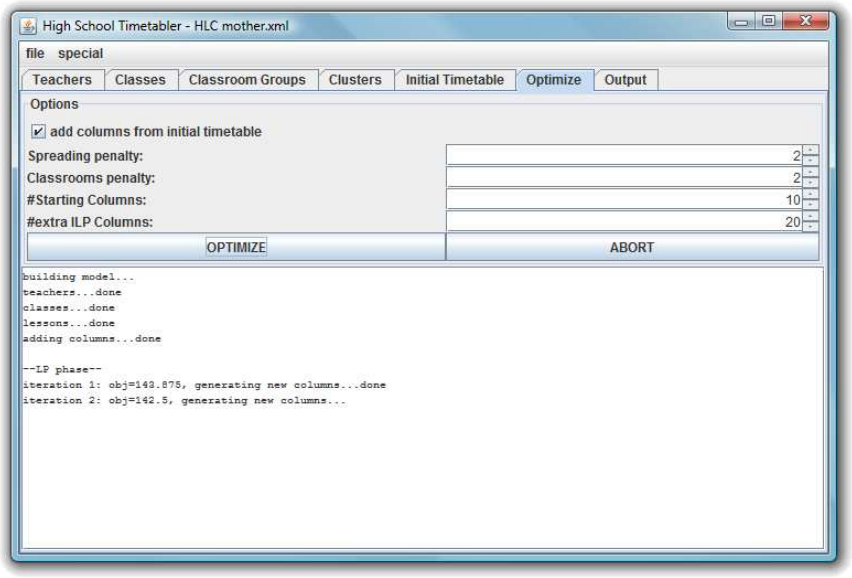


Figure A.6: The Optimize tab can be used to set some of the options for the optimization algorithm. The large text field at the bottom shows the progress after hitting the 'Optimize' button. If something goes wrong, I can also abort the optimization.

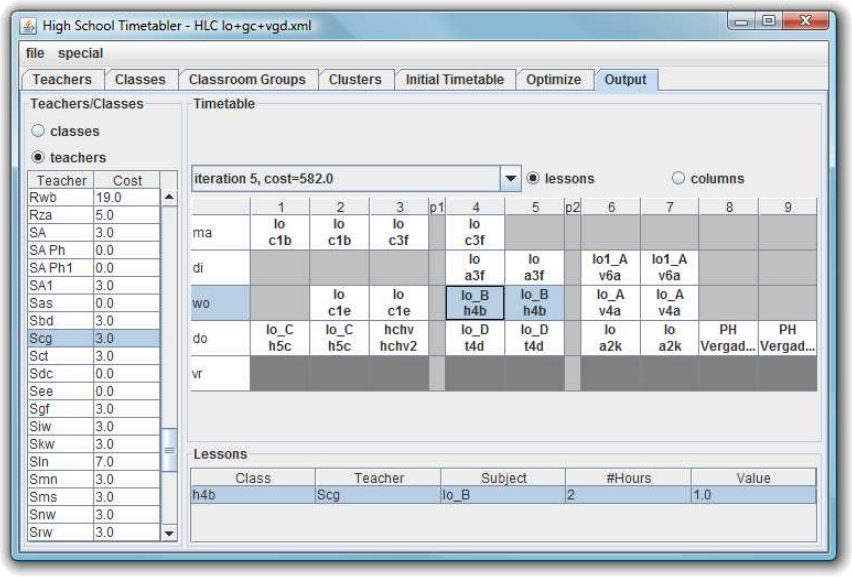


Figure A.7: When the solver has finished, the results can be viewed in the Output tab. In this case, we see the resulting timetable for PE teacher Scg, without any intermediate hours.

A.2 My Clustering Program

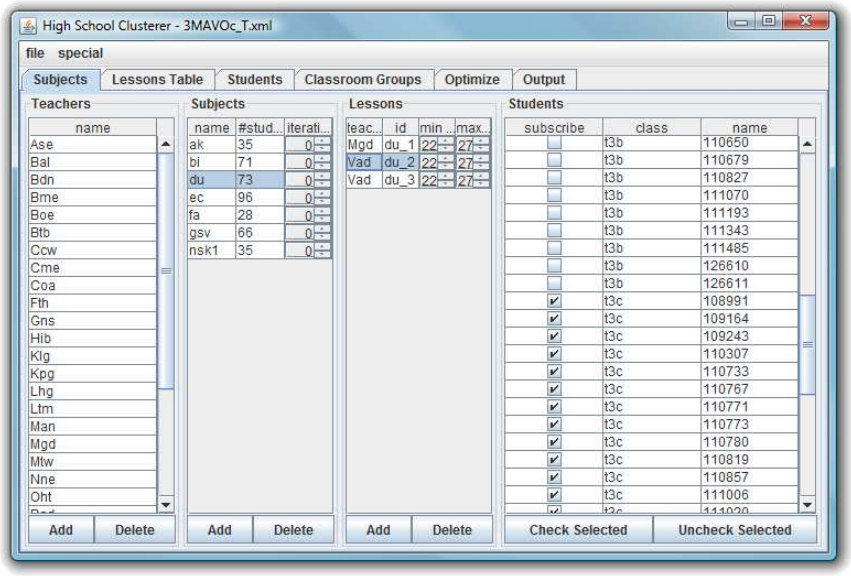


Figure A.8: Subjects tab showing all teachers, subjects and groups (inconsistently called 'lessons' in my program) that have to be clustered for this set of classes (the 3VMBO classes, in this case). The right most section is used to enter the Group Assignment Restrictions. As we can see, not all students who have subscribed to this subject 'du' may be assigned to group 'du_2'.

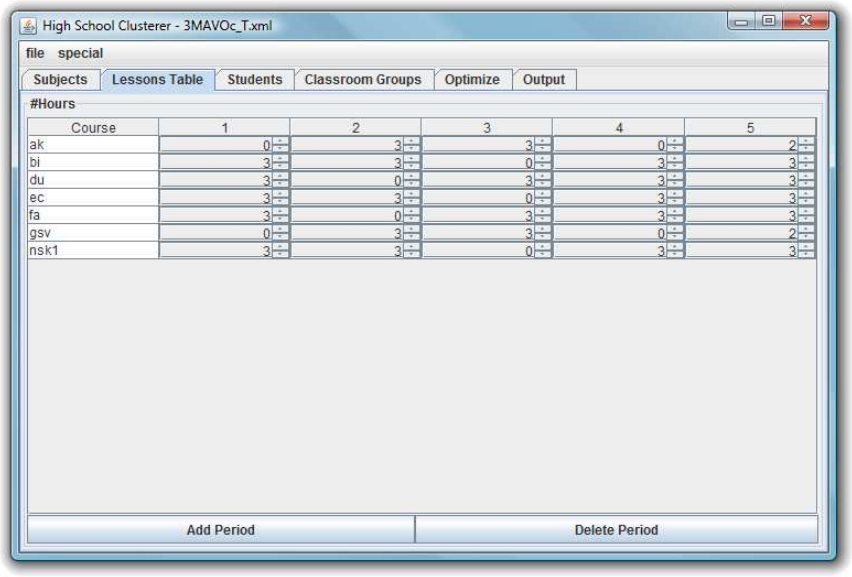


Figure A.9: Lessons Table tab showing the lessons table for the 3VMBO classes.

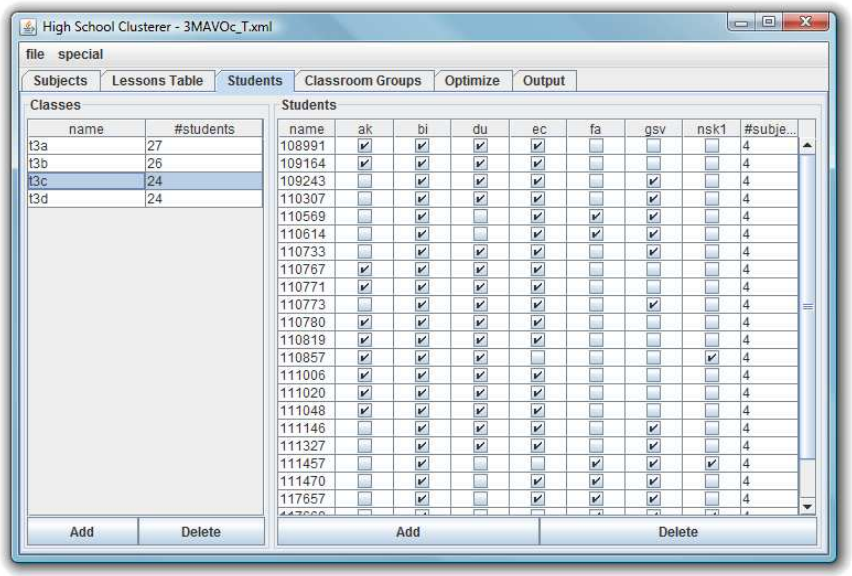


Figure A.10: Students tab showing the students of class t3c with their subscriptions.

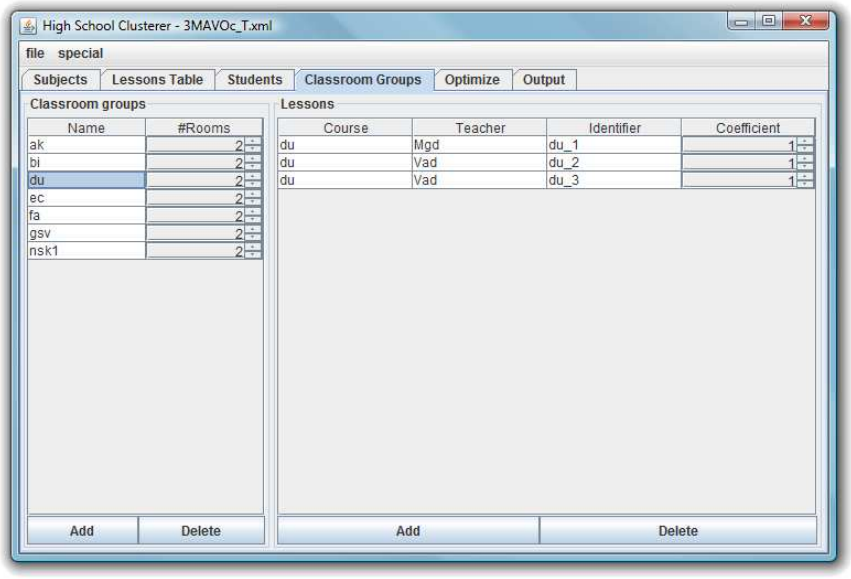


Figure A.11: The Classrooms tab is used to enter the classroom restrictions. In this case only two out of three groups of the subject 'du' may be in a pattern.

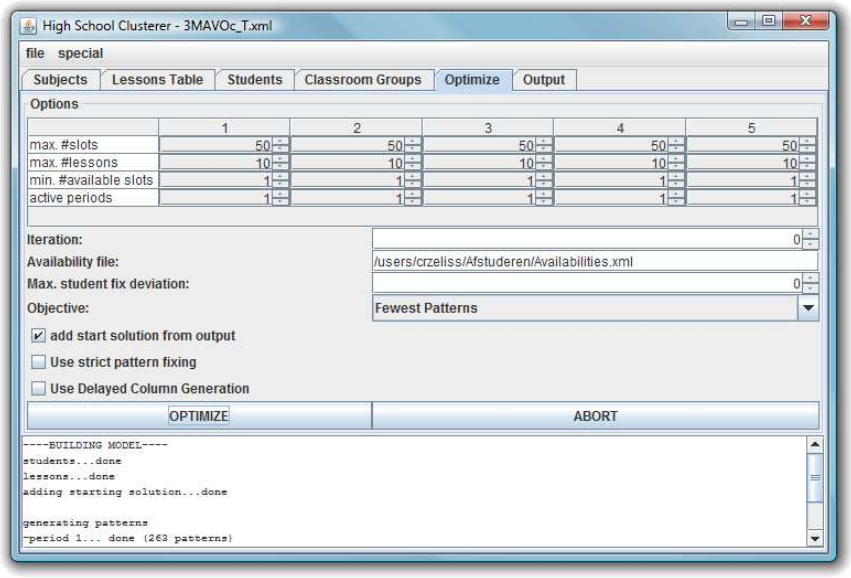


Figure A.12: This Optimize tab is similar to the Optimize tab of the Timetabling program: it is used to set options and to view information of the solving process.

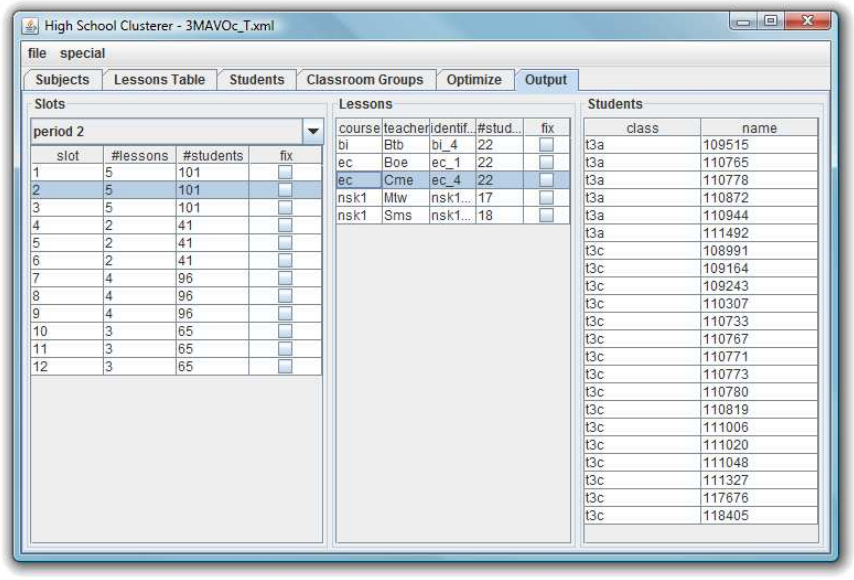


Figure A.13: The Output tab shows the results of the optimization process. In this case, the five groups (again inconsistently called 'lessons' in my program) of a cluster pattern in period 2 are shown in the second column. The students who have been assigned to one of these lessons, ec_4, are displayed in the third column.

Appendix B

Optimization Techniques

Introduction The main technique that I use to solve the timetabling- and clustering problem is Linear Programming (LP). To be able to deal with the large number of variables in the timetabling problem I also use the Column Generation (CG) technique. Finally, the column generation algorithm uses simple shortest path algorithms to find good columns. For those who are unfamiliar with these algorithms, this chapter explains the basic ideas behind them.

B.1 Linear Programming

Linear Programming is a technique that enables us to optimize a linear objective function subject to a set of linear constraints. This is best explained by a simple example: the *diet problem*. Suppose we have m different foods each containing a certain mixture of n different nutrients. Each unit of food j has a certain cost c_j, and a nutritional content of a_{ij} for nutrient i. The diet prescribes that we must eat a fixed amount b_i of nutrient i. Of course, we want to spend as little money as possible on buying the food items, while meeting the diet prescriptions. Table B.1 contains all the data we need for a general diet problem.

The decision we have to make is how many units we have to buy, for each food type. Therefore we introduce so-called *decision variables* $x_1, \ldots, x_m$,

	food 1	food 2	$\cdots$	food m	diet
cost	c_1	c_2	$\cdots$	c_m	
nutrient 1	a_{11}	a_{12}	$\cdots$	a_{1m}	b_1
nutrient 2	a_{21}	a_{22}	$\cdots$	a_{2m}	b_2
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
nutrient n	a_{n1}	a_{n2}	$\cdots$	a_{nm}	b_n

Table B.1: Diet Problem Data

where x_j represents the number of units of food j we buy (and eat). Since we are trying to minimize the total cost, the objective becomes:

$$\min c_1x_1 + c_2x_2 + \ldots + c_mx_m$$

Note that the objective function is indeed linear in the variables x_j. The constraints in this case are that we have to meet the diet: we must eat a fixed amount of each nutrient. This can be modeled by the following set of constraints:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{im}x_m = b_i \quad \text{for each } i = 1, \ldots, n$$

Finally, we must of course have the constraint that the variables must all be non-negative:

$$x_j \geq 0 \quad \text{for each } j = 1, \ldots, m$$

If we store the constants a_{ij} in a $n \times m$ matrix A, constants c_j in a vector c, constants b_i in a vector b and variables x_j in a vector x, we can write the complete problem as:

$$\begin{aligned} &\min c'x \\ &\text{subject to } Ax = b \\ &x \geq 0 \end{aligned}$$

This is a linear program in *standard form*: all variables must be nonnegative and all constraints are equality constraints. A general linear program may also contain inequality constraints (replace the =-sign by $\leq$ or $\geq$) or free (unbounded) variables (they are allowed to become negative). It may also have the objective to *maximize* instead of minimize the objective function. However, it is always possible to eliminate inequality constraints and free variables by a series of transformations (see [1], Chapter 1). Also, maximizing $c'x$ is essentially the same as minimizing $-c'x$. Therefore, a general linear program can always be transformed to the standard form.

There are several algorithms that solve linear programming problems. The most popular algorithm is the simplex method. This method is fast in most practical cases, but has been proven not to run in polynomial time in a worst-case scenario. There are, however, some algorithms that do solve linear programming problems in polynomial time, for instance the affine scaling algorithm (see [1], Chapter 9).

B.1.1 Integer Linear Programming

The models of the timetabling and clustering problem (see Section 3.3 and 7.4) are obviously not in standard form: they contain a lot of inequality constraints. As I mentioned before, this is not a problem: these constraints

can be transformed into equality constraints. Nevertheless, the models do contain so-called *integrality constraints*, like:

$$x_c \in \{0, 1\}$$

This constraint does two things. Firstly, it bounds the variable x_c: it may only take a value that is at least 0 and at most 1 (this type of constraint is still allowed in linear programming). Secondly, it only allows integer values for this variable (together with the bounds, this results in only two feasible values: 0 and 1). This is not allowed in linear programming, and makes the model an Integer Linear Program.

In the diet problem it is quite natural to include such integrality constraints $x \in \mathbb{Z}^m$: it is usually impossible to buy half a banana although the corresponding variable x_{banana} could be 1/2 in the optimal solution. Requiring integer values for x_j may make the problem infeasible: we required the diet prescriptions to be met exactly, and this may well be impossible for integer x_j. Therefore, we relax the diet constraints: we must buy the food items such that we have *at least* the required amounts of nutrients. The integral diet problem becomes:

$$\begin{aligned} &\min c'x \\ \text{subject to } &Ax \geq b \\ &x \geq 0 \\ &x \in \mathbb{Z}^m \end{aligned}$$

Unlike linear programs, ILP models (like the timetabling or clustering model) are generally not solvable efficiently (in polynomial time): integer linear programming is an NP hard problem. There are some exact algorithms that solve ILP models to optimality but take an exponential number of iterations, for instance cutting plane methods or branch-and-bound.

B.2 Column Generation

Unfortunately, the models that I try to solve in this thesis, are ILP models. Therefore, I am committed to inefficient algorithms and powerful solving software (like CPLEX, the software package that I have used). In case of the timetabling problem, there is another issue: the complete model does not fit into memory. Column Generation (CG) is a technique that could be used for this kind of large scale optimization. The term 'column' in this case corresponds to a variable in the model. The main idea behind CG is to delay adding a variable to the model until we know it is beneficial to do so. If there is no variable that is worthwhile to add, we stop. Because not all variables have to be added (usually, there are lots of useless variables), we keep the model small enough to fit in memory.

B.2.1 The Cutting Stock Problem

So how does this technique work? The classical example used to explain CG is the *cutting stock problem*. It is about a paper cutting company that cuts large paper rolls of width W into rolls of smaller width, meeting customer demand. This demand can be expressed in two vectors b and w: the customer wants b_i rolls of width w_i $(i = 1, \ldots, n)$. Of course, the paper cutting company wants to minimize the number of large rolls used, satisfying the customer order. The company has to decide for each large roll, how to cut it into smaller rolls; this is called a *cutting pattern*. Such a pattern can be expressed by a vector A_j, where its ith component a_{ij} represents the number of rolls of width w_i that are cut from this large roll. Decision variable x_j represents how many large rolls are cut according to pattern j $(j = 1, \ldots, m)$.

This problem description leads to a simple integer linear program. The objective is to minimize the total number of large rolls used, that is, we minimize $\sum_j x_j$. Of course, the variables x_j must be nonnegative and integer. Finally, we must satisfy the customer demand: the number of produced paper rolls of width w_i, that is $\sum_j a_{ij}x_j$, must be at least the demand b_i. We now have the following ILP (storing the constants a_{ij} in a big matrix A):

$$\begin{aligned} \min & \sum_j x_j \\ \text{subject to } & Ax \geq b \\ & x \geq 0 \\ & x \in \mathbb{Z}^m \end{aligned}$$

The number of feasible patterns m can be very large, implying a huge number of variables and columns of the matrix A. However, the vast majority of these patterns are probably useless for this problem (and the corresponding variables x_j will always remain 0). Therefore, even if the model does fit in memory, using column generation may be a good idea in this case.

step 1 First of all, we have to transform this problem into a linear program, because the CG algorithm needs information from the simplex algorithm (which only solves LP problems, not ILP). This transformation is easy: simply remove the integrality constraints $x \in \mathbb{Z}^m$. The resulting problem is called the *LP relaxation* of the cutting stock problem. Of course, we are cheating with this relaxation, since this simplifies the problem: the optimal solution of the LP relaxation may not be integer. However, this will be fixed in step 4.

step 2 The second step is to solve this LP relaxation (using the simplex method) for a small set of columns (patterns, in this case). This set of patterns may not be too small: we have to make sure that a feasible solution exists. In this problem it is easy to construct a small set of columns that allow a feasible solution: for each w_i include a pattern that only consists of one roll of this width. This results in an identity matrix A and an optimal solution $x = b$ with value $\sum_i b_i$.

step 3 From the theory of the simplex method (see Chapter 3 of [1]) we know that including a variable could improve the quality of a feasible solution if its *reduced cost* is negative (assuming we are *minimizing* the objective function). Conversely, if all reduced costs are nonnegative, we know that we have an optimal solution. In this case, the reduced cost of a pattern j equals $c_j - p'A_j$, where p is the vector of so-called *dual multipliers* (also called simplex multipliers) and c_j is the coefficient of x_j in the objective function (in this case, $c_j = 1$). How these dual multipliers can be computed is beyond the scope of this chapter. For now, it is important that p is available, given a solution. The third step is to try to find a pattern that we can add to this model, the column generation phase. This can be done by minimizing the reduced cost over *all* feasible patterns. If this minimum is nonnegative, apparently there is no pattern that could improve the quality of the solution: we have found an optimal solution of the LP relaxation. On the other hand, if this minimum is negative, we can add the corresponding pattern to the model, and solve it to optimality (we return to step 2), hopefully improving the objective value. This new solution also gives a new vector p, and we can try again (step 3) to find a pattern that could be added. We repeat step 2 and 3 until we have found an optimal solution of the LP relaxation.

I haven't explained yet how to find among all feasible patterns the one minimizing its reduced cost. This is the so-called *pricing problem*. In this case, we have to find a feasible vector a such that $1 - p'a$ is minimal. We can write this problem as an ILP, where the components a_i are the variables. Of course, since a representa a cutting pattern, all its components must be nonnegative and integer. A pattern is feasible if its total width $\sum_i w_i a_i$ does not exceed W. This leads to the following formulation:

$$\begin{aligned} &\min 1 - p'a \\ \text{subject to } &\sum_i w_i a_i \leq W \\ &a \geq 0 \\ &a \in \mathbb{Z}^n \end{aligned}$$

This problem can be transformed to a relatively small *integer knapsack problem* that can be solved quickly using dynamic programming. If we have

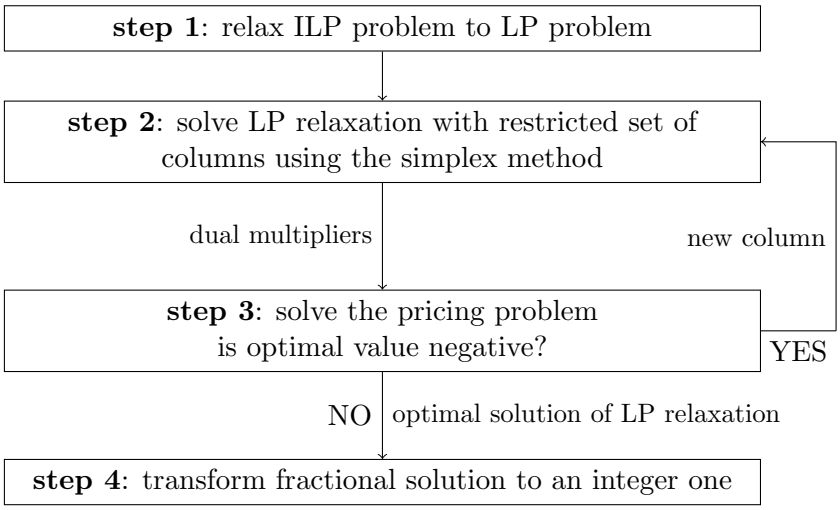


Figure B.1: The Column Generation Algorithm

solved this pricing problem, we must check whether it has a negative objective value. If it has, we add this column to the cutting stock LP relaxation, if not, we know that we have an optimal solution of this LP relaxation and we proceed to the final step 4.

step 4 The fourth step is to transform the optimal solution of the LP relaxation to an *integer* solution: the original cutting stock problem requires integer values x_j. If we are lucky, the solution is already integer, and we immediately have an optimal solution of the integer cutting stock problem. If not, we have to use some other method, for instance by rounding up. Another option is to solve the original ILP version of the cutting stock problem (for instance using branch-and-bound methods) with the restricted set of patterns that were generated. It is important to realize that an optimal solution of the LP relaxation generally gives no guarantee for the quality of the solution of the integer version. It may even happen that the integer problem is infeasible for the restricted set of columns. Nevertheless, in most practical situations the set of columns generated also allows a good integer solution.

B.2.2 Application to the Timetabling Problem

Figure B.1 shows the general column generation algorithm schematically. In the timetabling problem, columns correspond to timetable layouts for teachers and classes. The restricted set of columns that we start with are the n 'best' columns (with lowest cost) per teacher/class. If n is big enough this results in a feasible LP relaxation and we can start the column generation algorithm. The corresponding pricing problem can be modeled as a shortest

path problem (see Section B.3). We solve this problem for each pair (j, d) of teacher/class j and day d, and add the best column for every such pair (if it has negative reduced cost). This way, we usually add more than one column per iteration and make progress more quickly. Finally, the transformation of a fractional optimal solution to the LP relaxation to an integer solution is done by simply running an ILP solver on the integer programming problem, using the restricted set of generated columns (plus some extra columns, see Section 5.2.3).

B.3 Shortest Path

Solving the pricing problem in the timetabling phase reduces to finding a shortest path from source r to sink s in a layered graph. In this case, a shortest path is the path that minimizes the sum of the costs of the nodes (rather than the arcs) that are in this path. Because such a graph is acyclic, this is a very easy task that can be done in time $O(m)$ (see [5], Theorem 8.16), where m is the number of arcs in the graph. Let $c(v)$ denote the cost of node v and $d(v)$ the length of the shortest path from r to v. For each node v in layer 1, the length of the shortest path from r to v, is simply $d(v) = c(r) + c(v)$. If we have determined the shortest paths from r to all nodes in layer i, we can determine the $d(v)$ for a node v in the next layer $i + 1$ as follows:

$$d(v) = c(v) + \min_u \{d(u) | u \text{ is a predecessor of } v\}$$

We say that u is a predecessor of v if there is an arc (u, v). Because the graph is layered, it has the property that if v is in layer $i+1$, all predecessors u must be in layer i. Using the relation above, we determine the shortest path lengths $d(v)$ layer by layer, and we ultimately compute $d(s)$ (sink s is the only node in the last layer). If we store for each node v, the predecessor u with minimal $d(u)$, we can 'track back' from sink to source to find the path itself (not only its length $d(s)$).

Figure B.2 shows a simple example graph with 11 nodes. The bottom part of each node represents its cost. Under each node v, the value of $d(v)$ is displayed, together with the predecessor of the shortest path to this node. By tracking back from sink s to source r, we see that the shortest path is $r - B - E - I - s$ with length 6.

B.3.1 k Shortest Paths

It is also possible to find the k shortest paths in these layered graphs. I use this algorithm to find the k best columns per teacher/class that together form the initial set of columns for the LP relaxation. Similar to the simple shortest path algorithm, we determine for each node v, layer by layer,

the length of the shortest path from r to v. We now store not only the predecessors, but the complete paths (since one predecessor can be part of more than one of the shortest paths). Define $d_i(v)$ as the length of the ith shortest path from r to v. Of course, for each node v in layer 1, there is only 1 shortest path from r to v with length $d_1(v) = c(r) + c(v)$. For each node v in layer $i + 1$ we inspect the stored shortest paths of its predecessors u with their lengths $d_i(u)$ and choose the k paths with minimal $d_i(u)$. Simply adding $c(v)$ to the $d_i(u)$ and node v to the k paths then gives the k shortest paths from r to v.

Figure B.3 shows the same example graph as Figure B.2, but now illustrates the k shortest paths algorithm for $k = 3$: below each node v, the lengths $d_1(v)$, $d_2(v)$ and $d_3(v)$ of the 3 shortest paths from r to this node, together with the paths itself, are displayed. Because the graph is quite small, there are less than 3 feasible paths from r to all except nodes G and s. We can immediately read that $r - B - E - I - s$, $r - A - C - G - s$ and $r - B - E - G - s$ are the three shortest paths from source to sink with lengths 6,7 and 7, respectively.

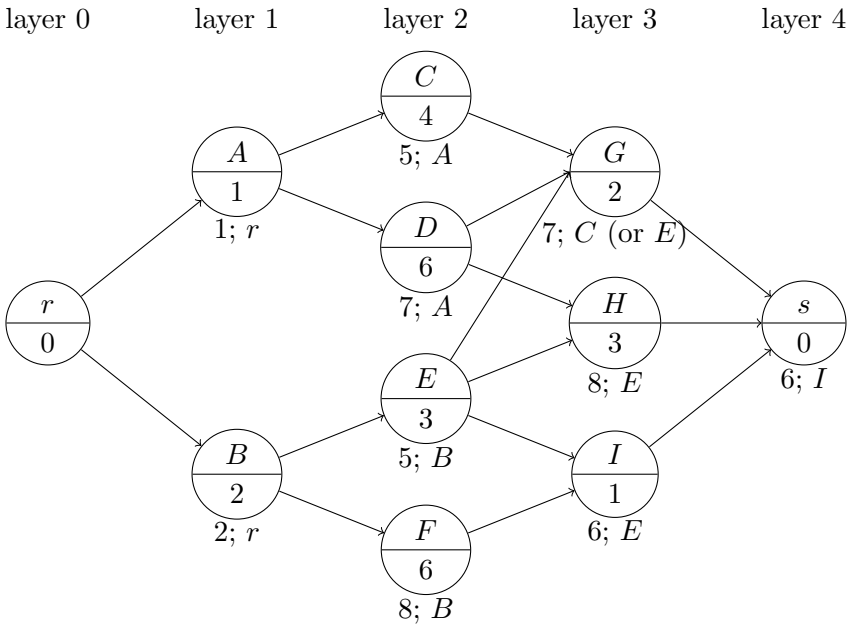


Figure B.2: Example Shortest Path Algorithm

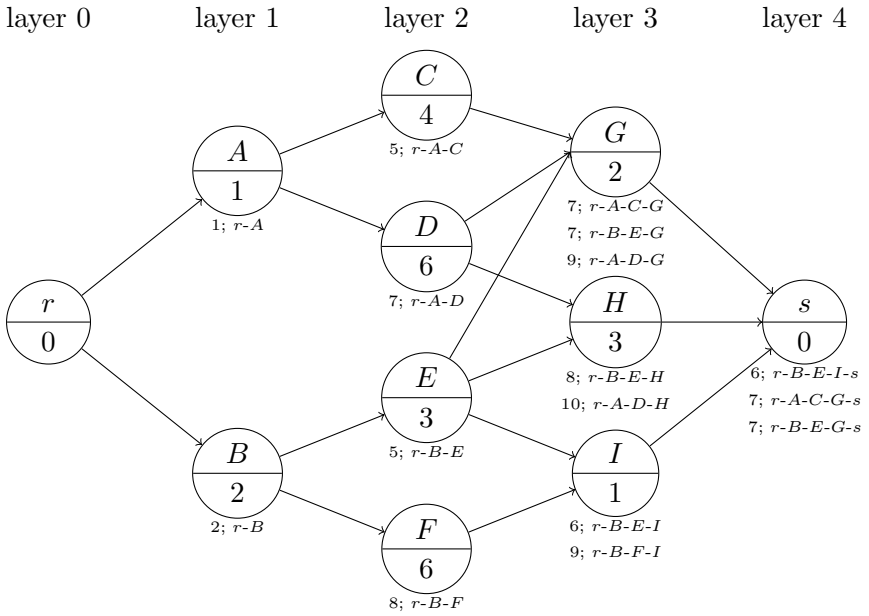


Figure B.3: Example k Shortest Paths Algorithm for $k = 3$

Appendix C

Reference

Introduction This appendix can be used as a reference while reading this thesis. It briefly describes the several modeling concepts (with some examples), variables, constraints and objectives for both the timetabling and the clustering problem described in part I and part II. The last section C.3 contains the full ILP models of the two problems.

C.1 Timetabling

C.1.1 Modeling Concepts

Classes/Teachers Teachers and classes are very similar from a modeling point of view: they both have *timetable layout preferences*. These preferences concern the number of lessons, intermediate hours and trips per day/week. Also, a teacher (or class) may have some *blocked time slots* (in particular, this applies to part-time teachers): no lesson may be scheduled at those time slots. All those preferences are known and determine the cost functions.

Here are some examples of timetable layout preferences:

- All upper school VMBO classes are preferred to have the first our off, each day.
- Teacher Abc is unavailable on Wednesday (that is, he has blocked all time slots on Wednesday).
- Teacher Def wants less than 3 trips per week.
- Teacher Ghi wants at most 1 intermediate hour on Monday, and at least 2 on Tuesday.

Lessons Another thing teachers and classes have in common, is that they both have lessons. Which teachers and classes are involved in a lesson (that

is, the *teachers and classes set*) is predetermined. Another known property of a lesson is its *multiplicity*: the number of time slots per week the lesson has to be scheduled. Additionally, we know for each lesson whether it has to be scheduled as one or more *n-double hours* (determining the hour group vector), in which *location* (building) it has to take place and in which kind of *classroom(s)*. A lesson can also be *fixed* to a time slot.

Most lessons in lower school only have one teacher and one class, for instance the following Mathematics lesson:

Example Lesson: a1aMath

classes set	a1a
teachers set	Abc
multiplicity	3
hour group vector	2-1 (one 2-double hour, one single hour)
location	lower school building
classroom type	lower school Mathematics/Economics room
fix	not applicable

For meetings, the classes set is empty, and the teachers set contains more than one teacher:

Example Lesson: mathMeeting

classes set	
teachers set	Abc Def Ghi
multiplicity	4
hour group vector	4 (four consecutive hours)
location	upper school building
classroom type	meeting room 1
fix	Friday, time slots 6-9

Finally, in upper school we can have cluster patterns, that involve several classes and teachers:

Example Lesson: v6ClusterPattern1

classes set	v6a v6b v6c
teachers set	Abc (Mathematics) Jkl (Biology)
multiplicity	1
hour group vector	1
location	upper school building
classroom types	upper school Mathematics/Economics room upper school Biology room
fix	not applicable

Classrooms/Locations Each lesson has to be given in an appropriate classroom. I distinguish different classroom groups for different types of lessons. For each classroom group, the *number of rooms* is known, as well as the *set of lessons* that have to be scheduled in a classroom group. For example, at the HLC there is a classroom group consisting of 3 upper school Mathematics/Economics rooms, in which all upper school Mathematics and Economics lessons must be given. Example lesson v6ClusterPattern1 is thus included in its lesson set.

At the HLC, there are two buildings: one for lower school lessons and one for upper school lessons. This concept of locations is important because teachers that teach both lower and upper school must have time to travel from one building to the other (the so-called trips).

Time A year is divided into several *periods*. Timetabling is done for each period separately, since each period a different set of lessons must be taught. I make a schedule for one *week*: every week in some period, the same schedule is used. At the HLC, a week has 5 *days*, and 9 *time slots* each day (at the HLC) with *breaks* after the third and fifth time slot (this is important for trips and double hours).

C.1.2 Variables, Constraints and Objective

There are two kinds of basic variables that I use in the timetabling model: x_c and y_{li}. They are defined as follows:

$$x_c = \begin{cases} 1 & \text{if column } c \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{li} = \begin{cases} 1 & \text{if lesson } l \text{ is scheduled at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

A column, in this case represents a timetable layout for some teacher or class for one single day. A column c can be seen as a 0-1-vector with an element for each time slot on the corresponding day and location b:

$$c_{ib} = \begin{cases} 1 & \text{if this teacher/class has a lesson in location } b \text{ at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

Constraints (1) make sure that the values of variables x_c and y_{li} correspond to each other. For instance, if lesson l is scheduled at time slot i and must be given in location b, the chosen columns of the teachers and classes involved in this lesson must have $c_{ib} = 1$. Here C_{jd} denotes the set of columns for teacher/class j and day d, L_{jb} denotes the set of lessons for teacher/class j that have to be in location b and I_d is the set of time slots on day d.

$$\sum_{c \in C_{jd}} x_c c_{ib} - \sum_{l \in L_{jb}} y_{li} = 0 \quad \text{for all } j, d, i \in I_d, b \quad (1)$$

Constraints (2) make sure that exactly one column is chosen per teacher or class per day, and by constraint (3), I force that the multiplicity m_l of the lessons is respected. Finally, if a lesson l is fixed to time slot i, constraint (4) makes sure of this.

$$\sum_{c \in C_{jd}} x_c = 1 \qquad \text{for all } j, d \qquad (2)$$

$$\sum_{i} y_{li} = m_l \qquad \text{for all } l \qquad (3)$$

$$y_{li} = 1 \qquad \text{for all } l \text{ fixed to } i \qquad (4)$$

If a lesson has double hours (meaning it has not an all-one hour group vector, this set of lessons is denoted by L^{DH}), I introduce variables $y^n_{l^k i}$ for each sublesson l^k of l (a sublesson corresponds to a component of the hour group vector):

$$y^n_{l^k i} = \begin{cases} 1 & \text{if the } n\text{th consecutive hour of sublesson } l^k \\ & \text{is scheduled at time slot } i \\ 0 & \text{otherwise} \end{cases}$$

Constraints (5a) make sure the lessons within an hour group are indeed scheduled consecutively, constraints (5b) restrict the choices for the first time slot of an hour group (avoiding an hour group to cross a break or a day break) and constraints (5c) link the new variables to the y_{li}. Here n_{h^l} denotes the number of components of the hour group vector h^l for lesson l.

$$y^n_{l^k i} - y^{n+1}_{l^k (i+1)} = 0 \qquad \text{for all } l \in L^{\mathrm{DH}}, i,\ 1 \le k \le n_{h^l},\ 1 \le n < h^l_k \quad (5a)$$

$$y^1_{l^k i} = 0 \qquad \text{for all } l \in L^{\mathrm{DH}}, \text{ infeasible } i,\ 1 \le k \le n_{h^l} \quad (5b)$$

$$\sum_{k=1}^{n_{h^l}} \sum_{n=1}^{h^l_k} y^n_{l^k i} = y_{li} \qquad \text{for all } l \in L^{\mathrm{DH}}, i \quad (5c)$$

We want a lesson to be scheduled for at most one hour (group) per day. This is a soft constraint; variables z_{ld} measure deviation from this constraint. If l has double hours, constraints (6a) apply, if it has an all-one hour group vector, constraints (6b) apply.

$$z_{ld} = \begin{cases} n & \text{if } n+1 \text{ hour groups of lesson } l \text{ are scheduled on day } d \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{i \in I_d} \sum_{k=1}^{n_{h^l}} y^1_{l^k i} - z_{ld} \le 1 \qquad \text{for all } l \in L^{\mathrm{DH}}, d \qquad (6a)$$

$$\sum_{i \in I_d} y_{li} - z_{ld} \le 1 \qquad \text{for all } l \notin L^{\mathrm{DH}}, d \qquad (6b)$$

To make sure the lessons fit in their appropriate rooms, I included constraints (7). Here L^r denotes the set of lessons that must be scheduled in classroom group r, and n^r is its number of rooms.

$$\sum_{l \in L^r} y_{li} \le n^r \quad \text{for each } r, i \quad (7)$$

Constraints (8a), (8b) and (8c) measure deviation from the week layout preferences (preferred minimum and maximum number of intermediate hours and preferred maximum number of trips). The values of the nonnegative variables $\hat{z}^{\mathrm{int}}_j$, $\check{z}^{\mathrm{int}}_j$ and $\hat{z}^{\mathrm{trips}}_j$ represent these deviations:

$$\sum_{c \in C_j} x_c \mathrm{int}(c) - \hat{z}^{\mathrm{int}}_j \le \hat{n}^{\mathrm{int}}_j \qquad \text{for all } j \qquad (8a)$$

$$\sum_{c \in C_j} x_c \mathrm{int}(c) + \check{z}^{\mathrm{int}}_j \ge \check{n}^{\mathrm{int}}_j \qquad \text{for all } j \qquad (8b)$$

$$\sum_{c \in C_j} x_c \mathrm{trips}(c) - \hat{z}^{\mathrm{trips}}_j \le \hat{n}^{\mathrm{trips}}_j \qquad \text{for all } j \qquad (8c)$$

Naturally int(c) and trips(c) denote the number of intermediate hours and trips in column c, respectively; $\hat{n}_j^{\text{int}}$, $\check{n}_j^{\text{int}}$ and $\hat{n}_j^{\text{trips}}$ the corresponding bounds.

The objective is to minimize the costs of the columns plus spreading and week layout deviations:

$$\min \sum_j \sum_d \sum_{c \in C_{jd}} f_{jd}(c) x_c + c_z \sum_l \sum_d z_{ld} + \sum_j (\hat{c}^{\text{int}} \hat{z}_j^{\text{int}} + \check{c}^{\text{int}} \check{z}_j^{\text{int}} + \hat{c}^{\text{trips}} \hat{z}_j^{\text{trips}}) \quad \text{(obj)}$$

Here c_z, $\hat{c}^{\text{int}}$, $\check{c}^{\text{int}}$ and $\hat{c}^{\text{trips}}$ are the weighing constants, and $f_{jd}(c)$ is the cost of column c for teacher/class j and day d.

C.2 Clustering

C.2.1 Modeling Concepts

Class Types/Students Clustering is done for upper school students belonging to the same class type (same year and level). For each student its *subscriptions* to optional subjects are known.

At the HLC there are thee 6VWO classes (thus belonging to the same class type): v6a, v6b and v6c. In total, they have 77 students. As an example, the subscriptions of five of these students are shown:

student	subscriptions
Mark	Biology, Mathematics, Chemistry
Steven	History, Mathematics
Charlotte	History, Mathematics, Physics
Kevin	Biology, Physics
Suzan	Economics, History, Mathematics

Subjects For each subject, we know its *multiplicity*. If there are many subscriptions to the same subject, the set of subscribed students must be partitioned into several *groups*. The number of groups per subject is known, the partitioning of the students is not (this is part of the clustering problem). Of course, the multiplicity of each of the groups is equal to the multiplicity of the corresponding subject.

For the subjects from the example student subscriptions, these could be the corresponding multiplicities for period 1 and groups:

subject	multiplicity	#subscriptions	group(s)
Biology	2	50	B1, B2
Chemistry	3	10	C1
History	3	35	H1, H2
Mathematics	2	61	M1, M2, M3
Physics	3	22	P1

Groups For each group of some subject, I thus know its *multiplicity*, but also the *teacher* that teaches this group. Additionally, I have known *bounds on the number of students* that may be assigned to this group and an *ideal group size*. Finally, I could have *group assignment restrictions*: sometimes the set of students that may be assigned to a group is restricted (and therefore a *subset* of the set of subscribed students to the corresponding subject).

We have seen that in our example, Mathematics is very popular in 6VWO, and needs three groups: M1, M2 and M3. These could be their size bounds, ideal sizes and sets of students (from above five students) that could be assigned to the groups:

group	teacher	min. #students	max. #students	ideal #students
M1	Abc	12	18	15
M2	Def	20	26	23
M3	Def	20	26	23

group	possible participants
M1	Mark, Charlotte, Suzan
M2	Mark, Steven, Charlotte, Suzan
M3	Mark, Steven, Charlotte, Suzan

As we can see, group M1 is preferred to be a bit smaller than M2 and M3: this is because teacher Abc is inexperienced. We can also see that student Steven may not be assigned to this group M1, since Steven is difficult to handle (especially by an inexperienced teacher): this is an example of a group assignment restriction.

Cluster Patterns A *set of groups* whose sets of assigned students do not overlap and that all have a different teacher could be grouped in a cluster pattern. It is the objective of the clustering problem to form as few cluster patterns as possible, while each group appears the correct number of times (according to the multiplicities of the groups). These cluster patterns are input of the timetabling problem (patterns can be seen as big lessons involving several classes and teachers). To avoid infeasibility issues in the timetabling phase, *teacher availabilities* and *classroom information* (these things are known) could also be included in the clustering phase.

The following set of 10 patterns could be a solution to the example (for period 1) in this section. Note that all groups appear the correct number of times (according to their multiplicities) in the chosen patterns:

M3-B1-B2	M3-B1-B2	
M1-M2	M1-M2	
C1-H1	C1-H1	C1-H1
P1-H2	P1-H2	P1-H2

Of course, the group assignments are also part of the solution, but they are not displayed here.

C.2.2 Variables, Constraints and Objective

There are two kinds of basic variables that I use in the clustering model: x_c and y_{ig}. They are defined as follows:

$$x_c = \begin{cases} n & \text{if column } c \text{ is chosen } n \text{ times} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ig} = \begin{cases} 1 & \text{if student } i \text{ is assigned to group } g \\ 0 & \text{otherwise} \end{cases}$$

A column in this case represents a pattern (set of groups). A column c can be seen as a 0-1-vector with an element for each group g:

$$c_g = \begin{cases} 1 & \text{if group } g \text{ is in this cluster pattern} \\ 0 & \text{otherwise} \end{cases}$$

Constraints (1) make sure each group g appears the correct number of times (according to its multiplicity m_g^p for period p) in the set of chosen patterns for each period (C^p denotes the set of columns for period p). Constraints (2) force a student i to be assigned to exactly one group per subject it is subscribed to, and constraints (3) impose an upper bound $\hat{n}^p$ on the number of patterns we may use per period p. The set of subjects student i is subscribed to is denoted by S_i, and $G_{i,s}$ denotes the set of groups of subject s, student i may be assigned to.

$$\sum_{c \in C^p} x_c c_g = m_g^p \qquad \text{for all } g, p \qquad (1)$$

$$\sum_{g \in G_{i,s}} y_{ig} = 1 \qquad \text{for all } i, s \in S_i \qquad (2)$$

$$\sum_{c \in C^p} x_c \leq \hat{n}^p \qquad \text{for all } p \qquad (3)$$

For each group g, we have a lower and upper bound ($\check{n}_g$ and $\hat{n}_g$, respectively) on the number of students it may have. Constraints (4a) and (4b) force these bounds to be respected. Constraints (4c) measure deviation from the ideal size n_g^{ideal} of each group g. The values of the nonnegative variables z_g^+ and z_g^- represent this deviation. Here I_g denotes the set of students that could be assigned to group g.

$$\sum_{i \in I_g} y_{ig} \leq \hat{n}_g \qquad \text{for all } g \qquad (4a)$$

$$\sum_{i \in I_g} y_{ig} \geq \check{n}_g \qquad \text{for all } g \qquad (4b)$$

$$\sum_{i \in I_g} y_{ig} - z_g^+ + z_g^- = n_g^{\text{ideal}} \qquad \text{for all } g \qquad (4c)$$

We also have to make sure that for every chosen pattern, there is no student that is assigned to more than one group in this pattern. This is forced by constraint (5):

$$M_{(g_1,g_2)}(y_{ig_1} + y_{ig_2}) + \sum_{c \in C_{g_1} \cap C_{g_2}} x_c \leq 2M_{(g_1,g_2)}$$
$$\text{for all } i,\ g_1,g_2 \in G_i, g_1 \neq g_2 \quad (5)$$

Here C_g denotes the set of columns in which group g appears and G_i is the set of groups (of all subjects) student i may be assigned to. Finally, $M_{(g_1,g_2)}$ is a constant, that must be at least the maximum value $\sum_{c \in C_{g_1} \cap C_{g_2}} x_c$ can take.

The objective is to minimize the total number of patterns chosen. There is also a secondary objective (with lower priority, implying a low weighing constant c_z) to minimize the deviation of the ideal group sizes.

$$\min \sum_c x_c + c_z \sum_g (z_g^+ + z_g^-) \quad \text{(obj)}$$

C.3 Full ILP Models

Timetabling ILP Model

$\min \sum_j \sum_d \sum_{c \in C_{jd}} f_{jd}(c) x_c + c_z \sum_l \sum_d z_{ld} + \sum_j (\hat{c}^{\text{int}} \hat{z}_j^{\text{int}} + \check{c}^{\text{int}} \check{z}_j^{\text{int}} + \hat{c}^{\text{trips}} \hat{z}_j^{\text{trips}})$			(obj)	Objective
Subject to:				
$\sum_{c \in C_{jd}} x_c c_{ib} - \sum_{l \in L_{jb}} y_{li}$	$= 0$	for all $j, d, i \in I_d, b$	(1)	Variable correspondence
$\sum_{c \in C_{jd}} x_c$	$= 1$	for all j, d	(2)	Choose one column per day
$\sum_i y_{li}$	$= m_l$	for all l	(3)	Multiplicity of lessons
y_{li}	$= 1$	for all l fixed to i	(4)	Fixed lessons
$y^n_{l^k i} - y^{n+1}_{l^k (i+1)}$	$= 0$	for all $l \in L^{\text{DH}}, i, 1 \le k \le n_{h^l}, 1 \le n < h^l_k$	(5a)	Double hours
$y^1_{l^k i}$	$= 0$	for all $l \in L^{\text{DH}}$, infeasible i, $1 \le k \le n_{h^l}$	(5b)	
$\sum_{k=1}^{n_{h^l}} \sum_{n=1}^{h^l_k} y^n_{l^k i}$	$= y_{li}$	for all $l \in L^{\text{DH}}, i$	(5c)	
$\sum_{i \in I_d} \sum_{k=1}^{n_{h^l}} y^1_{l^k i} - z_{ld}$	≤ 1	for all $l \in L^{\text{DH}}, d$	(6a)	Spreading
$\sum_{i \in I_d} y_{li} - z_{ld}$	≤ 1	for all $l \notin L^{\text{DH}}, d$	(6b)	
$\sum_{l \in L^r} y_{li}$	$\le n^r$	for each r, i	(7)	Classrooms
$\sum_{c \in C_j} x_c \text{int}(c) - \hat{z}_j^{\text{int}}$	$\le \hat{n}_j^{\text{int}}$	for all j	(8a)	Week layout preferences
$\sum_{c \in C_j} x_c \text{int}(c) + \check{z}_j^{\text{int}}$	$\ge \check{n}_j^{\text{int}}$	for all j	(8b)	
$\sum_{c \in C_j} x_c \text{trips}(c) - \hat{z}_j^{\text{trips}}$	$\le \hat{n}_j^{\text{trips}}$	for all j	(8c)	
x_c	$\in \{0, 1\}$	for all c	(9a)	Variable bounds and integrality
y_{li}	$\in \{0, 1\}$	for all l, i	(9b)	
$y^n_{l^k i}$	$\in \{0, 1\}$	for all $l \in L^{\text{DH}}, i, 1 \le k \le n_{h^l}, 1 \le n < h^l_k$	(9c)	
z_{ld}	≥ 0	for all l, d	(9d)	
$\check{z}_j^{\text{int}}, \hat{z}_j^{\text{int}}, \hat{z}_j^{\text{trips}}$	≥ 0	for all j	(9e)	

Clustering ILP Model

$\min \sum_c x_c + c_z \sum_g (z_g^+ + z_g^-)$		(obj)	Objective
Subject to:			
$\sum_{c \in C^p} x_c c_g = m_g^p$	for all g, p	(1)	Lessons table
$\sum_{g \in G_{i,s}} y_{ig} = 1$	for all $i, s \in S_i$	(2)	One group per subject
$\sum_{c \in C^p} x_c \leq \hat{n}^p$	for all p	(3)	Number of cluster patterns per period
$\sum_{i \in I_g} y_{ig} \leq \hat{n}_g$	for all g	(4a)	Group sizes
$\sum_{i \in I_g} y_{ig} \geq \check{n}_g$	for all g	(4b)	
$\sum_{i \in I_g} y_{ig} - z_g^+ + z_g^- = n_g^{\text{ideal}}$	for all g	(4c)	
$M_{(g_1,g_2)}(y_{ig_1} + y_{ig_2}) + \sum_{c \in C_{g_1} \cap C_{g_2}} x_c \leq 2M_{(g_1,g_2)}$	for all i, $g_1, g_2 \in G_i, g_1 \neq g_2$	(5)	No overlapping students
$x_c \in \mathbb{Z}_{\geq 0}$	for all c	(6a)	Variable bounds and integrality
$y_{ig} \in \{0, 1\}$	for all $i, g \in G_i$	(6b)	
$z_g^+, z_g^- \geq 0$	for all g	(6c)	

Bibliography

[1] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization (Athena Scientific Series in Optimization and Neural Computation, 6)*. Athena Scientific, February 1997.

[2] Guido Diepen. *Column Generation Algorithms for Machine Scheduling and Integrated Airport Planning*. PhD thesis, Utrecht University, 2008.

[3] Olivier Du Merle, Daniel Villeneuve, and Jacques Desrosiers. Stabilized column generation. *Discrete Math*, 194:229–237, 1999.

[4] ILOG. ILOG CPLEX v11. `http://www.ilog.com`.

[5] Alexander Schrijver. *Combinatorial Optimization*. Springer, February 2003.