School Timetabling using Satisfiability Solvers

– A method to construct timetables for Dutch secondary education –

Kenneth K. Chin–A–Fat

Delft, The Netherlands

September, 2004

Preface

In 1998 I started studying Applied Mathematics at the Delft University of Technology, The Netherlands. Three years later I started designing timetables at the Rotterdamsch Lyceum, a high school in Rotterdam, The Netherlands as a side-line.

When I had to pick a subject for my thesis, my first thought went out to timetable designing. I contacted dr. H. van Maaren and by sheer chance he guided a student, I. Smeets, on tackling timetables for Dutch secondary education. When she finished her research, I could continue it.

I want to thank the members of my graduating committee prof. dr. ir. C. Roos and dr. C. Witteveen. Furthermore I want to thank I. Smeets, for giving me useful guidelines to continue the research on timetabling. Special thanks go to dr. H. van Maaren, who supervised my research and helped me bringing this task to an end. I hope you will enjoy reading my thesis.

Kenneth Chin-A-Fat,

September 2004

Summary

In this thesis we present a method to construct timetables for Dutch secondary education. We model the school timetable problem as a propositional satisfiability problem. Once the school timetable problem is transformed, we use programs, so called SAT solvers, to solve the propositional satisfiability problems by finding a satisfying assignment.

We first show how several requirements of a real school, the Rotterdamsch Lyceum, Rotterdam, The Netherlands, can be stated as constraints. Next we use the SAT solvers to find a satisfying assignment and thereby a timetable. We look at the performance on constructing a timetable and on adjusting the timetable. The first kind of these adjustments considers improvements wanted by the timetable designer. The other kind considers adjustments wanted due to temporarily events: several teachers or classes maybe absent, for instance because of illness or a school–trip.

After this, we expand the Rotterdamsch Lyceum virtually to gain a larger school. First we double the size, and next we triple the size of the Rotterdamsch Lyceum. Once a timetable is constructed for the school of triple size of the Rotterdamsch Lyceum, we again check on the ability of SAT solving techniques to adapt for temporarily events.

Contents

Chapter 1

Introduction

Without a timetable designer constructing proper timetables most schools, if not all schools, would be a total chaos. It's is only when things go wrong, that the importance is noticed. Yet most people don't know what it takes to create a timetable. In most cases it takes the timetable designer a large part of the summer break to construct an acceptable timetable. Recently, the number of optional and compulsory subjects in the senior department in Dutch secondary education has increased, which has made timetabling by hand practically impossible. Nowadays, designing timetables is a task performed by specialists, using state-of-the-art software, to help them to construct a timetable, using different methods to tackle the timetable problem.

We describe a new way to tackle the school timetabling problem. For we will model the timetabling problem as a propositional satisfiability problem (SAT). Recently, there has been made considerable progress in solving SAT problems, and we use this development in solving SAT problems to construct timetables for Dutch secondary education. This thesis describes how the school timetable problem can be transformed into a SAT problem. After this, we look at the performance of the solvers in creating a new timetable and adjusting an existing timetable.

1.1 Propositional satisfiability

A propositional or Boolean satisfiability problem (SAT) is a problem of determining a satisfying assignment for a propositional formula Φ or proving that it contains a contradiction. A formula consists of Boolean variables x_i and the logical connectives $\bigwedge$ (and), $\bigvee$ (or) and $\neg$ (not). A literal is a proposition or a negation of a variable x_i. If there is a satisfying assignment for a formula, the formula is called satisfiable; otherwise it' s called unsatisfiable.

We will model the timetable problem as a propositional formula and use SAT solvers to solve the formula. Most SAT solvers require the formula to conform one of the DIMACS standard formats [DIM 93], and the format that is used the most is the Conjunctive Normal Form (CNF). We will also create our formula in CNF.

A formula in CNF consists of the conjunction of a number of clauses, where a clause is a disjunction of a number of literals. It can be shown as: $\Phi = \bigwedge C_i$, where $C_i = \bigvee l_j$, with each l_j a literal. This format is often used to find an assignment for a formula because the formula is satisfied if and only if every clause C_i is satisfied. It is known that every propositional formula can be converted into CNF.

It is well known that SAT is NP-complete, which makes it unlikely that there is an algorithm solving it in reasonable time in all cases. Nevertheless, there are solvers capable of solving many instances pretty fast and because of this capability hope rises that also many school timetable problems can be solved in reasonable time by these solvers. At most cases we examine in this thesis we won't be disappointed with regards to the runtimes. Some of the results, which can be found in chapter 7 and 8, are shown in table 1.1. To solve the formulas we first use a pre–processor to simplify the formula. The simplified formula will be solved by the solvers. The runtimes in table 1.1 are the total runtimes of the pre–processor and the solver together. The results show that large formulas can be solved in reasonable time. It should be noticed that it would take the solver less than two minutes to assign periods at the Rotterdamsch Lyceum and less than nine minutes to assign periods at the Rotterdamsch Lyceum of double size.

Table 1.1: Results for school timetabling for the Rotterdamsch Lyceum

Part of school timetabling	Variables	Clauses	Runtime
Period assignment senior department	1,152,000	3,892,401	55 seconds
Period assignment junior department	803,700	1,806,454	55 seconds
Period assignment senior department of RL 2x	2,050,560	5,349,659	180 seconds
Period assignment junior department of RL 2x	2,145,600	3,872,675	319 seconds

Nowadays mostly local search methods are implemented in timetabling software. Often local search methods will keep on searching for a solution, even if there is none. One of the advantages of satisfiability solvers is that they can also tell us whether our constraints are unsatisfiable.

1.2 Dutch secondary education

In this section we'll introduce the Dutch secondary education system and point out the important aspects to us. The Dutch secondary education exists of three different kinds of education: vmbo (voorbereidend middelbaar beroepsonderwijs), havo (hoger algemeen voortgezet onderwijs) and vwo (voorbereidend wetenschappelijk onderwijs). The vmbo lasts four years, the havo lasts five years and the vwo six years. The vwo can be divided in the atheneum and the gymnasium, where both school types offer the same subjects at the same level, but at the gymnasium it is compulsory to attend Latin or Greek. It is common use to divide the secondary education in the junior department (basisvorming) and the senior department.

The junior department of the vmbo usually lasts two years, whereas the junior departments of the other school types last three years. During the junior department each kind of education offers the same subjects, which only differ in difficulty. In this way all Dutch children receive the same general education at the right level of difficulty.

In the senior department of the vmbo, the students have to choose one of four kinds of education (leerwegen) to finish the vmbo, which are each split into four sectors. Their curriculum depends on the chosen sector. At the senior department of the havo and vwo (De Tweede Fase) the students also have four options, called profiles (profielen). Their curriculum

is split into a common part, which all students of the same year of the same school type have to attend, a compulsory part, which all students of the same profile of the same year have to attend and a free part, which exists of subjects chosen by the students. The way of education in the senior department of the havo and vwo differs from the junior department and the vmbo. It is called 'studiehuis' and aims at teaching the students to study independently.

It should be pointed out that the division of secondary education in junior and senior department is a meaningful division to a timetable designer. In the junior department almost all subjects are compulsory, except for Latin and Greek, whereas in the senior department a large part of the subjects is (partial) optional. Besides, of the junior department it is known which teacher has to teach which classes, whereas at the senior department it is only known which students have to follow which subjects and which teachers have to teach which subject. The relation between student and teacher is not always evident, because for instance a subject can be taught by several teachers or a teacher can teach the same subject to several classes. This means that a different approach is needed at the different departments.

In The Netherlands timetabling is nowadays performed by specialists. To gain insight in the effort of Dutch timetable designers, we've distributed a survey among several schools in The Netherlands. The survey shows that the timetable designers who returned the survey need several weeks of their summer break to construct a timetable, and the used software usually needs several hours to run to help them at constructing timetables. Results of the survey are shown in the Appendix.

1.3 Timetable designing

The aim of designing timetables is to combine students, teachers, subjects, periods and rooms to gain lessons. As in most methods to tackle designing timetables, we'll start off ignoring the room assignment. When the other elements are combined, we'll assign the rooms. This is because if we assign the rooms together with the other elements, the problem would increase tremendously in size. And, as we will see, the problem will become huge, so we'll try to keep it as small as possible. Problems could rise for specific rooms. Some subjects are only suited for specific rooms, like physical education. If we assign more lessons of physical education than available gyms to the same periods, we'll be in trouble at assigning rooms. To prevent this, we'll consider the room type during the period assignment, so when we arrive at room assignment, we don't have to worry about the room type anymore.

Not every combination of students, teachers, subjects and periods is possible. Teachers only teach a certain subject or some subjects. Students are obliged to follow the compulsory subjects and choose the optional subjects. In the junior departments the students are combined in classes, so they have to follow each lesson together. These connections between the elements are given by the school.

The method will be illustrated by a real life example, for we will design a timetable for the Rotterdamsch Lyceum, a school in Rotterdam, The Netherlands, which offers vwo, havo and one type of the vmbo, which was used to be called mavo (middelbaar algemeen voortgezet onderwijs) and is now called vmbo-tl (theoretische leerweg). It has little over 500 students and is a considered a small school, but even here, the timetabling is a major task.

1.4 The approach of timetable designing

Even though there are five elements to gain a lesson, i.e. students, teacher, subject, period and room, we don't have to determine every relation. The relation between teacher and subject is given, as is the relation between student and subject. The approach to this problem will be as follows: first we'll transform the students curriculum in the senior department to gain a usable relation between student and subject in the senior department, similar to the relation in the junior department. Then we we'll assign periods to the senior department, followed by the period assignment of the junior department. The only implications for the junior department will be the adjusted availability of some teachers and of some rooms. This division of the timetable construction follows naturally from the difference in relations between students, subjects and teachers, as we pointed out in section 1.2. Moreover, it is well known that the most difficult part is the timetable of the senior department. So we'll tackle the bottleneck first, otherwise we might not be able to tackle it at all. At the end we'll assign the rooms and then every element is added to gain lessons and we attain a timetable. Even though the senior department is handled first, we'll first explain the way to handle the junior department, because this will help to understand the extra difficulties in the senior department.

1.5 Period assignment junior department

In the junior department, the students are usually divided into classes. Of each class, it is known which subjects will be taught by which teacher. To get a SAT solver to assign periods we'll have to state the constraints. In [Wil 02] the constraints are divided into requirements and desires. The problem with SAT is that desires cannot be easily stated. They can be approached in a cumbersome way, but this will in most cases create a huge amount of constraints. So, instead of stating desires, we have to decide to state a desire as a requirement or to ignore the desire at all. We stated the next constraints to gain a period assignment:

Teacher constraints: every teacher can have at most one lesson on the same period.

Student constraints: every student can have at most one lesson on the same period.

Teacher availability constraints: a lesson can only be taught on a period the teacher is available.

Double period constraints: two lessons of particular subjects, like drawing or physical education, will be assigned to two consecutive periods.

Room compatibility constraints: only rooms of a suitable type are assigned to each lesson. Even though we don't assign rooms yet, we have to keep in mind that some lessons can only take place in particular rooms, like physical education. If there are for instance two gyms available, we cannot assign three physical education lessons to the same period.

Educational constraints: lessons of the same subject are distributed over the week as much as possible.

Students idle period constraints: students have no idle periods. These constraints will be very hard constraints to satisfy. We will therefore require these constraints to be

satisfied for a large portion of the day, hoping that we can handle the rest of the idle periods by hand.

The next constraints are necessary to tell a SAT solver what is expected of a timetable.

Completeness constraints: every existing lesson does actually occur.

Negation constraints: every non existing lesson does not occur.

Uniqueness constraints: every lesson is scheduled only once. If a particular lesson could be assigned to several different periods, we only allow it to be assigned to one single period.

The next constraints are examples of constraints which are not included, because they're not desirable enough to us:

Travel time constraints: there must be sufficient travel time for teachers and students if they have to travel between two different locations

Teachers idle period constraints: teachers have no idle periods. These constraints are not taken into account, because they will often conflict with the students idle period constraints. Because it's the Rotterdamsch Lyceum's policy that there are no idle periods for students in the junior department, these constraint were omitted.

Didactic constraints: some subjects may not be suited to be taught at the end of the day, because students need to concentrate during these lessons and this is pretty hard at the end of the day. So a preference could be stated for some subjects not to be taught at the end of the day. Because we couldn't find any research on this subject, we omitted these constraints.

1.6 Student sectioning and period assignment senior department

In the senior department the students are usually not yet divided into classes. It is known which students have chosen which subjects. Of these subjects it is known which teachers will teach them, and finally the maximum number of students a teacher can teach at once is given. To be able to assign a group of students, together with a subject and a teacher, to a period, we first have to know which students are included in this group of students. When more students have chosen a subject than a teacher can teach at once, the group of students has to be split in several lessongroups. We'll call this *student sectioning*. While doing this we'll have to take into account that the groupsize of the different lessongroups have to be almost equal. It will be shown that, after a few adjustments, student sectioning can be tackled the same way as period assignment. When the student sectioning is finished, we know which students are contained in a particular group which has to be assigned to a period along with a subject and a teacher and we can assign periods almost just like we showed how to do in the junior department. The only constraints we will ignore are the students idle period constraints, which ensure that students have no idle periods, because these constraints will be very hard to satisfy. Idle periods are accepted in the senior department, mostly because during the idle periods students can study independently. The 'studiehuis', the senior department of the vwo and havo, aims at studying independently, so few idle periods are accepted.

1.7 Room assignment

When all periods are assigned, we will assign the rooms. This will not be handled as a satisfiability problem, but instead we use a simple algorithm. The next constraints should be satisfied during assigning rooms.

Room constraints: every room can have at most one lesson at the same period.

Room compatibility constraints: lessons are only assigned to suitable rooms. We already saw we've took these constraints into account at period assignment, so satisfying these constraints will be no problem.

Class own room constraints: several classes of the Rotterdamsch Lyceum in the junior department are assigned to one single room.

Teacher own room constraints: several teachers have their own room, especially teachers in the senior department.

Room capacity constraints: the group of students fits in the assigned room.

Same room constraints: teachers are assigned to the same room as much as possible.

Completeness constraints: every existing lesson does actually take place in a room.

Uniqueness constraints: every lesson takes place in one single room.

1.8 Adjustments

When a timetable is constructed by the SAT solver, it will hardly be perfect to the timetable designer. He or she would like to change the period assignment of the timetable by hand or by using the solver again. We will show how this can be done. Changing the room assignment may also be wanted. We'll also illustrate how this can be done.

When teachers or classes are absent, adjustments are needed for a single day. These *daily adjustments* are often needed. We'll show how to handle daily adjustments by creating a new curriculum for that single day, which leads us to a special case of assigning periods for the junior department, because only a part of the original curriculum is considered, as is only one single day. Absence of teachers or students can last longer than a day, for instance because of a school–trip. In that case we need *multiple daily adjustments*. We'll show how daily adjustments can be extended to multiple daily adjustments.

1.9 Results

Once we know how timetabling can be handled by modelling it as a satisfiability problem, we want to learn about the performance. This will be shown in chapter 7. First timetables are constructed for the Rotterdamsch Lyceum using several solvers. Then the performance on several adjustments are shown. Finally, the Rotterdamsch Lyceum is virtually expanded to look at the performance on a larger school than the Rotterdamsch Lyceum in chapter 8. First we will double the size of the Rotterdamsch Lyceum. Secondly, we triple the size of the Rotterdamsch Lyceum and look at the performance on adjustments too.

This thesis is set up in the following way. The period assignment of the junior department will be handled first in chapter 2 to gain insight in the way we tackle most problems. The senior department will be handled next, by student sectioning in chapter 3 and the period assignment of the senior department in chapter 4. In chapter 5 we'll take a look at assigning rooms and in chapter 6 the adjustments are considered. In chapter 7 we show the results on the performance on constructing a timetable by solvers, along with several tests to gain insight on the performance on adjustments by the solver. In chapter 8 the Rotterdamsch Lyceum is virtually expanded to examine the performance on larger schools than the Rotterdamsch Lyceum. Finally, in chapter 9 we enumerate our conclusions on constructing school timetables as satisfiability and point out some cases for further research on this subject.

Chapter 2

Period assignment junior department

As said, we will first show how to assign periods in the junior department, even though during constructing timetables we will first assign periods in the senior department. We reverse the order, because period assignment in the junior department is more straightforward and will make the period assignment in the senior department and the student sectioning in the senior department easier to understand.

We start off with enumerating the data required to construct timetables in the junior department. After this we introduce a useful tool to understand the variables, namely the allocationmatrix. Then we define the variables, which will be used to construct our constraints. The main part of this chapter will be about the construction of the constraints. Finally, we'll show what the input to the solver looks like.

2.1 Required information

In the junior department, the students are divided into classes. The curriculum of every class is known, which states the number of lessons per week of each subject that a class has to follow. The teachers are assigned to subjects, usually more than one, and to classes. The occurrence of teachers teaching more than one subject is a bit of a difficulty to us. We will handle this by 'splitting' the teacher in several *subject–teachers*, and don't assign periods to teachers, but to subject–teachers. If teacher `chk` teaches the subjects `wi` and `na`, this teacher will be seen as two different subject–teachers, namely `chk-wi` and `chk-na`. Of course, the assigned periods of the subject–teachers of the same teacher have to be mutually disjoint. The allowed number of days per week and periods per day are known, as are the periods that the teachers are available. Of particular subjects, it is known that they, if possible, need to be assigned in double periods, i.e. two consecutive periods, mostly the same subjects which need rooms that suit these subjects. The next symbols are used:

k	:	Class in the junior department	with	$1 \leq k \leq K$
i	:	Subject–teacher in the junior department	with	$1 \leq i \leq D$
j	:	Number of a lesson of a subject	with	$1 \leq j \leq U$
d	:	Day	with	$1 \leq d \leq S$
t	:	Period	with	$1 \leq t \leq T$

2.2 The allocationmatrix

A useful tool to understand the construction of timetables by satisfiability is the allocationmatrix. If a subject is taught several times to a class per week, we often need to distinguish between the different lessons, for instance to require an optimal distribution of the lessons during the week. By appointing different lessons to different elements of the allocationmatrix, we can distinct between different lessons.

The allocationmatrix of a class k $A_k(i,j)$ is constructed as follows. It's rows represent all subject–teachers, teaching that class or not. Because the maximum number of subject–teachers equals D, the number of rows of $A_k(i,j)$ will also equal D. The number of columns equals U, the maximum number of lessons per subject. If a subject–teacher teaches n lessons to class k, the first n elements of the corresponding row of $A_k(i,j)$ will contain ones, followed by $U-n$ zeros. Each column represents the number of a lesson, so the first lesson of subject–teacher i' to class k is represented by $A_k(i',1)$ and the second lesson is represented by $A_k(i',2)$. This leads to the next description of the allocationmatrix of a class k $A_k(i,j)$:

$$A_k(i,j) = \begin{cases} 1, & \text{if subject–teacher } i \text{ is teaching a } j^{th} \text{ lesson to class } k \text{ ;} \\ 0, & \text{else.} \end{cases} \tag{2.1}$$

It should be pointed out that the numbering of the lessons in this way doesn't imply an order in time on the lessons. It can occur that the second lesson of a subject is taught previous to the first. The numbering is only introduced to be able to distinguish between the different lessons.

We'll give an illustration on how to construct allocationmatrices from a curriculum. A curriculum is a matrix of which the rows represent the classes and the columns represent the subject–teachers. Element (k,i) of the curriculum contains the number lessons taught to class k by subject–teacher i. A mini curriculum is shown in table 2.1. We can for instance see that subject–teacher i_1 teaches two lessons to class k_1 and non to class k_2.

Table 2.1: Mini curriculum

	i_1	i_2	i_3	i_4
k_1	2	0	4	3
k_2	0	2	3	5

If we want to construct an allocationmatrix for class k_1 we see that there are four subject–teachers, so we need four rows in $A_{k_1}(i,j)$. We also see that even though the maximum number of lessons of class k_1 equals four, we need five columns, because the maximum number of lessons of the 'whole school' equals five. Subject–teacher i_1 teaches two lessons to class k_1, so the first row contains two ones, followed by three zeros. The second subject–teacher teaches no lessons to class k_1, so the second row only contains zeros. The allocationmatrix is shown in table 2.2. The allocationmatrix of class k_2 is constructed in the similar way, and is shown in table 2.3.

Table 2.2: Allocationmatrix A_{k_1} of class k_1

k_1	j_1	j_2	j_3	j_4	j_5
i_1	1	1	0	0	0
i_2	0	0	0	0	0
i_3	1	1	1	1	0
i_4	1	1	1	0	0

Table 2.3: Allocationmatrix $A_{k_2}$of class k_2

k_2	j_1	j_2	j_3	j_4	j_5
i_1	0	0	0	0	0
i_2	1	1	0	0	0
i_3	1	1	1	0	0
i_4	1	1	1	1	1

2.3 Definition of the variables

We will now define the variables of which the values will be assigned by the solver to construct our timetable.

$x_{k,i,j,d,t}$: the lesson corresponding to $A_k(i,j)$ is assigned to period t of day d;

where k	:	Class in the junior department	with	$1 \leq k \leq K$
i	:	Subject–teacher in the junior department	with	$1 \leq i \leq D$
j	:	Number of a lesson of a subject	with	$1 \leq j \leq U$
d	:	Day	with	$1 \leq d \leq S$
t	:	Period	with	$1 \leq t \leq T$

In other words, the Boolean variable $x_{k,i,j,d,t}$ indicates that to class k subject–teacher i is teaching lesson j on day d on period t. For example, stating $x_{1,2,3,4,5}$ makes sure that to class 1 subject–teacher 2 is teaching lesson 3 on day 4 on period 5. Stating $\neg\, x_{5,4,3,2,1}$ forbids that to class 5 subject–teacher 4 is teaching lesson 3 on day 2 on period 1. Every index of the variable ranges from 1 to it's maximum, so this mean we will have K x D x U x S x T variables.

Reduction to one index
Solvers cannot handle variables like directly above. They require integers, except for zero, to represent variables, like stating 102 makes sure that variable 102 is assigned to true. Stating -201 makes sure that variable 201 is set to false. This means we'll have to transform our variable $x_{k,i,j,d,t}$ to a variable x_n, and if we want to state $x_{k,i,j,d,t}$ we state n, and if we want to state $\neg\, x_{k,i,j,d,t}$ we state $-n$. We will transform our indices in the next way:

$$n = t + T \text{ x } (d-1) + T \text{ x } S \text{ x } (j-1) + T \text{ x } S \text{ x } U \text{ x } (i-1) + T \text{ x } S \text{ x } U \text{ x } D \text{ x } (k-1) \quad (2.2)$$

By this construction, the variables of the same class are consecutive. Furthermore, of all classes the variables are sorted by subject–teacher in the same order. The variables of each subject–teacher are sorted by j, the number of the lesson. These variables are sorted by day and the variables of each day are sorted by period.

The solver will assign all variables to true or false. If a variable is assigned to true, then the corresponding lesson occurs. If a variable is set to false, the corresponding lesson does not occur. The constraints of this assignment will be explained in the next section.

2.4 Constraints of period assignment junior department

We first define several relations which often occur.

- SameTeacher(i,i')
  Let i and i' be two subject–teachers.

$$SameTeacher(i,i') = \begin{cases} 1, & \text{if subject–teachers } i \text{ and } i' \text{ represent the same teacher;} \\ 0, & \text{else.} \end{cases} \tag{2.3}$$

- TeacherAvailable(i,d,t)
  Let i be a subject–teacher, let d be a day and let t be a period.

$$TeacherAvailable(i,d,t) = \begin{cases} 1, & \text{if subject–teacher } i \text{ is available on day } d \text{ on period } t; \\ 0, & \text{else.} \end{cases} \tag{2.4}$$

- DoublePeriod(i)
  Let i be a subject–teacher.

$$DoublePeriod(i) = \begin{cases} 1, & \text{if subject–teacher } i \text{ requires double periods;} \\ 0, & \text{else.} \end{cases} \tag{2.5}$$

- SameRoom(i,i')
  Let i and i' be two subject–teachers.

$$SameRoom(i,i') = \begin{cases} 1, & \text{if subject–teachers } i \text{ and } i' \text{ need the same room;} \\ 0, & \text{else.} \end{cases} \tag{2.6}$$

These relations will turn out to be useful to express our requirements. If, for instance, we only want to compare two subject–teachers of the same teacher, we only choose those i and i' of which SameTeacher(i,i')=1. We will now express the constraints of section 1.5 in CNF. One type of clauses that will often occur is that two variables should not be both assigned to true. We transform this requirement into a usable constraint by the next law of De Morgan:

$$\neg\ (x_{k,i,j,d,t} \wedge x_{k',i',j',d',t'}) \equiv \neg\ x_{k,i,j,d,t} \vee \neg\ x_{k',i',j',d',t'} \tag{2.7}$$

Another type of clause we'll often need will be of the form A $\rightarrow$ B, meaning if A occurs then B also has to occur. We can transform this kind of clauses by the next equivalency:

$$x_{k,i,j,d,t} \rightarrow x_{k',i',j',d',t'} \equiv \neg\ x_{k,i,j,d,t} \vee x_{k',i',j',d',t'} \tag{2.8}$$

Teacher constraints: every teacher can have at most one lesson on the same period.

At these constraints all subject–teachers i' and i of the same teacher are compared. If $A_k(i,j) = 1$ and $A_{k'}(i',j') = 1$ where k and k' can either represent a couple of classes or the same class and j and j' can either represent different numbers of lessons or the same, meaning that subject–teacher i is teaching lessonnumber j to class k and subject–teacher i' is teaching lessonnumber j' to class k', only one of them can occur on periods the teacher is available, which will be stated as not both lessons should be assigned to the same period. The periods that the teacher is not available, will be blocked by other constraints. So we only have to look at the periods the teacher is available. We will need to keep the number of constraints as small as possible, because this can speed up the solver. To state these constraints, we'll have to state that for every couple of classes k and k', and for every couple of subject–teachers i and i' which refer to the same teacher, and for every couple of number of lessons taught by these subject–teachers, and for every period of the week i and i' are available, both lessons can not occur on the same period. Because SameTeacher(i,i) = 1, these constraints will also make sure that every individual subject–teacher can have at most one lesson on the same period. We should prevent that these clauses are stated when $k = k'$, $i = i'$ and $j = j'$, because in that case we would state clauses like this:

$$\neg\, x_{k,i,j,d,t} \vee \neg\, x_{k,i,j,d,t} \tag{2.9}$$

which leads to a situation where the lesson cannot be assigned to any period at all.

If we don't allow that $k = k'$, $i = i'$ and $j = j'$, we can state the teacher constraints in the next way:

$$\bigwedge_{k,k'} \bigwedge_{i,i'|SameTeacher(i,i')=1} \bigwedge_{\substack{j|A_k(i,j)=1 \\ j'|A_{k'}(i',j')=1}} \bigwedge_{d} \bigwedge_{t|TeacherAvailable(i,d,t)=1} \neg\, x_{k,i,j,d,t} \vee \neg\, x_{k',i',j',d,t} \tag{2.10}$$

Student constraints: every student can have at most one lesson on the same period.

For every class every two subject–teachers are compared. If they both teach this class, and don't refer to the same teacher, then for every couple of lessons by these subject–teachers to this class, only one can occur on the periods they're both available. On periods only one of the subject–teachers is available, it can't occur that they both teach that class.

$$\bigwedge_{k} \bigwedge_{i,i'|SameTeacher(i,i')=0} \bigwedge_{\substack{j|A_k(i,j)=1 \\ j'|A_k(i',j')=1}} \bigwedge_{d} \bigwedge_{t|\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i',d,t)=1}} \neg\, x_{k,i,j,d,t} \vee \neg\, x_{k,i',j',d,t} \tag{2.11}$$

Teacher availability constraints: a lesson can only be taught on a period the teacher is available.

For every period a subject–teacher is not available, all of the corresponding lessons are forbidden.

$$\bigwedge_{k} \bigwedge_{i} \bigwedge_{j|A_k(i,j)=1} \bigwedge_{d} \bigwedge_{t|TeacherAvailable(i,d,t)=0} \neg\, x_{k,i,j,d,t} \tag{2.12}$$

Double period constraints: two lessons of particular subjects, like drawing or physical education, will be assigned to two consecutive periods.

If i is a subject–teacher who requires double periods, the number of nonzero elements in the i^{th} row of A_k tells us how many double periods are needed. If there are four lessons of physical education needed, this will result in two double periods. If there are five lessons, this will also result in two double periods, and a single period.

Let's say there are five lessons of physical education needed for class k by subject–teacher i. How can we assure double periods? We tackle this problem by using the numbers of the lessons. In short, we stated the following for $1 \leq t < T$:

$$x_{k,i,1,d,t} \rightarrow x_{k,i,2,d,t+1} \quad \wedge \quad x_{k,i,3,d,t} \rightarrow x_{k,i,4,d,t+1} \tag{2.13}$$

which assures that the first lesson of physical education is followed by the second, and the third is followed by the fourth. We also need the first of the double periods not to be assigned to the last period of each day:

$$\neg\, x_{k,i,1,d,T} \quad \wedge \quad \neg\, x_{k,i,3,d,T} \tag{2.14}$$

The last thing we need to assure is that double periods don't become triple or quadruple periods. This can be done by assuring no lesson of physical education is taught directly before the double period. For the first double period this can be done in the next way for $1 < t < T$:

$$\begin{aligned} x_{k,i,1,d,t} &\rightarrow \neg x_{k,i,4,d,t-1} \\ x_{k,i,1,d,t} &\rightarrow \neg x_{k,i,5,d,t-1} \end{aligned} \tag{2.15}$$

We also have to state that no lesson of physical education is taught directly after a double period. We can do it for the second double period like this for $1 < t < T$:

$$\begin{aligned} x_{k,i,4,d,t} &\rightarrow \neg x_{k,i,1,d,t+1} \\ x_{k,i,4,d,t} &\rightarrow \neg x_{k,i,5,d,t+1} \end{aligned} \tag{2.16}$$

If we take a good look at 2.15 and 2.16, we notice that the first constraints of 2.15 and 2.16 are the same constraints. So we only need one of them. The second constraints of 2.15 and 2.16 make sure that the 'last' lesson of physical education remains a single period.

We needed several constraints to attain what was wanted, which will make a general description of the constraints as in previous cases difficult to give. That's why we'll split the possibilities and formulate a description for each of the possibilities:

- Let i be a subject–teacher with DoublePeriod$(i) = 1$ and for class k requiring an even number of lessons, so $\sum_j A_k(i,j)$ is even. A single lesson is not possible, so these constraints are omitted. We first make sure that the periods are coupled:

$$\bigwedge_k \bigwedge_i \bigwedge_{j\ odd|A_k(i,j)=1} \bigwedge_d \bigwedge_{t<T|\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i,d,t+1)=1}} \neg\, x_{k,i,j,d,t} \vee x_{k,i,j+1,d,t+1} \tag{2.17}$$

  and we assure that the first periods of the double periods are not assigned to the last period of each day:

$$\bigwedge_k \bigwedge_i \bigwedge_{j\ odd|A_k(i,j)=1} \bigwedge_d \bigwedge_{T|TeacherAvailable(i,d,T)=1} \neg\, x_{k,i,j,d,T} \tag{2.18}$$

To keep the double periods separated we state:

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{\substack{j \\ j'}}\bigwedge_{\substack{odd|A_k(i,j)=1 \\ even|A_k(i,j')=1}}\bigwedge_{d}\bigwedge_{1<t<T|}\bigwedge_{\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i,d,t-1)=1}} \neg x_{k,i,j,d,t} \vee \neg x_{k,i,j',d,t-1} \quad (2.19)$$

- Let i be a subject–teacher with DoublePeriod$(i) = 1$ and for class k requiring an odd number of lessons, so $\sum_j A_k(i,j)$ is odd. Let also the number of lessons needed be equal to J. We first make sure that the periods are coupled:

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j\neq J}\bigwedge_{odd|A_k(i,j)=1}\bigwedge_{d}\bigwedge_{t<T|}\bigwedge_{\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i,d,t+1)=1}} \neg x_{k,i,j,d,t} \vee x_{k,i,j+1,d,t+1} \quad (2.20)$$

and we assure that the first periods of the double periods are not assigned to the last period of each day:

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j\neq J}\bigwedge_{odd|A_k(i,j)=1}\bigwedge_{d}\bigwedge_{T|TeacherAvailable(i,d,T)=1} \neg x_{k,i,j,d,T} \quad (2.21)$$

To keep the double periods separated we state:

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{\substack{j\neq J \\ j'}}\bigwedge_{\substack{odd|A_k(i,j)=1 \\ even|A_k(i,j')=1}}\bigwedge_{d}\bigwedge_{1<t<T|}\bigwedge_{\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i,d,t-1)=1}} \neg x_{k,i,j,d,t} \vee \neg x_{k,i,j',d,t-1} \quad (2.22)$$

To keep the single lesson single we state:

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j}\bigwedge_{odd|A_k(i,j)=1}\bigwedge_{d}\bigwedge_{1<t<T|}\bigwedge_{\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i,d,t-1)=1}} \neg x_{k,i,j,d,t} \vee \neg x_{k,i,J,d,t-1} \quad (2.23)$$

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j}\bigwedge_{even|A_k(i,j)=1}\bigwedge_{d}\bigwedge_{1<t<T|}\bigwedge_{\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i,d,t+1)=1}} \neg x_{k,i,j,d,t} \vee \neg x_{k,i,J,d,t+1} \quad (2.24)$$

Room compatibility constraints: only rooms of a suitable type are assigned to each lesson. Even though we don't assign rooms yet, we have to keep in mind that some lessons can only take place in particular rooms, like physical education. If there are for instance two gyms, we cannot assign three lessons of physical education to the same period.

The subjects which require suitable rooms are physical education and drawing and crafts, the latter two share the same room. At the Rotterdamsch Lyceum there is only one gym and one craftsroom. We'll construct the constraints for physical education. The constraints of crafts and drawing are identical. Because there is only one suitable room, all timetables of subject–teachers of physical education have to be mutually disjoint concerning periods, as should be the subject–teachers of drawing and crafts. This can be stated as follows:

$$\bigwedge_{k,k'}\bigwedge_{i,i'|SameRoom(i,i')=1}\bigwedge_{\substack{j|A_k(i,j)=1 \\ j'|A_{k'}(i',j')=1}}\bigwedge_{d}\bigwedge_{t|}\bigwedge_{\substack{TeacherAvailable(i,d,t)=1 \\ TeacherAvailable(i',d,t)=1}} \neg x_{k,i,j,d,t} \vee \neg x_{k',i',j',d,t} \quad (2.25)$$

Just like at the teacher constraints, we should require that not all indices of $x_{k,i,j,d,t}$ and $x_{k',i',j',d,t}$ are equal, because this would lead to the situation where no lesson can be assigned.

We should remember that before we assign periods to the junior department, we first assigned periods to the senior department and physical education, drawing and crafts are also in the curriculum of the senior department. Now do drawing and crafts of the senior department have their own room at the Rotterdamsch Lyceum, but physical education doesn't. This means that on the periods that physical education is taught in the senior department, no physical education lesson can be taught in the junior department. Let $\mathbb{P}_d$ contain all periods on day d the gym is occupied by the senior department and let $\mathbb{I}$ contain all subject–teachers of physical education, we should block all periods in $\mathbb{P}_d$ for every day for the junior department for all subject–teachers in $\mathbb{I}$:

$$\bigwedge_{k}\bigwedge_{i\in\mathbb{I}}\bigwedge_{j|A_k(i,j)=1}\bigwedge_{d}\bigwedge_{t\in\mathbb{P}_d} \neg\, x_{k,i,j,d,t} \tag{2.26}$$

Educational constraints: lessons of the same subject are distributed over the week as much as possible.

Different views are possible, but it is common use that lessons of the same subject should be distributed over the week. Because the Rotterdamsch Lyceum also has this point of view, we'll try to create our timetable satisfying these constraints.

When the number of available days of a teacher is larger or equal to the number of lessons for a subject, the lessons can be distributed over the available days. The restriction will be that if a lesson occurs, no other lesson of the same subject is allowed to occur on the same day. If the subject–teacher requires double periods, we can of course not satisfy this constraints. That's why we exclude these subject–teachers from the constraints:

$$\bigwedge_{k}\bigwedge_{i|DoublePeriod(i)=0}\bigwedge_{\substack{j|A_k(i,j)=1\\ j'\neq j|A_k(i,j')=1}}\bigwedge_{d}\bigwedge_{\substack{t|TeacherAvailable(i,d,t)=1\\ t'\neq t|TeacherAvailable(i,d,t')=1}} \neg\, x_{k,i,j,d,t} \vee \neg\, x_{k,i,j',d,t'} \tag{2.27}$$

When we exclude the subjects which require double periods from the education constraints, several double periods can occur on the same day of the same subject. So we need an adjusted version of 2.27 for these subjects. If we think of it, every double period begins with a lesson having an odd number, so j is odd. If we restate 2.27, but requiring that j is odd, we state that all lessons with an odd number should occur on a different day and thereby that all double periods occur on a different day. As a bonus we also achieve that the remaining single lesson also occurs on a different day.

$$\bigwedge_{k}\bigwedge_{i|DoublePeriod(i)=1}\bigwedge_{\substack{jodd|A_k(i,j)=1\\ j'\neq jodd|A_k(i,j')=1}}\bigwedge_{d}\bigwedge_{\substack{t|TeacherAvailable(i,d,t)=1\\ t'\neq t|TeacherAvailable(i,d,t')=1}} \neg\, x_{k,i,j,d,t} \vee \neg\, x_{k,i,j',d,t'} \tag{2.28}$$

If the number of available days of a teacher is smaller than the number of lessons for a subject, we're forced to allow several lessons of the same subject on the same day, but we still want an optimal distribution of the lessons over the available days. We'll show how this can be achieved. If R is the ratio of the number of lessons and the number

of available days, then we want about R lessons every day. We can achieve this by requiring that at least $\lfloor R \rfloor$ lessons, which denotes the largest number of lessons less than R, and at most $\lceil R \rceil$ lessons, which denotes the smallest number of lessons larger than R, occur on each day.

We'll illustrate how this can be done. Let subject–teacher i be required to teach class k six lessons on four available days. If one or more of the available days contains less than $\lfloor R \rfloor$ available periods, we have to adjust our constraints. We won't give a solution to every possible situation, but with a little creativity it should be possible to cope with every situation. The ratio $R = \frac{6}{4} = 1.5$ tells us that we need at least $\lfloor 1.5 \rfloor = 1$ lesson and at most $\lceil 1.5 \rceil = 2$ lessons to occur on each day. To assure at least one lesson on a day we can state, informally:

$$\bigwedge_{d} \bigvee_{t} \bigvee_{j} x_{k,i,j,d,t} \tag{2.29}$$

These constraints make sure that for every day, at least on one of the periods at least one of the lessons occurs. To assure at most two lessons on a day, we state informally that it's not allowed that three lessons are given on each day:

$$\bigwedge_{j_1 \neq j_2 \neq j_3} \bigwedge_{d} \bigwedge_{t_1 \neq t_2 \neq t_3} \neg\, x_{k,i,j_1,d,t_1} \vee \neg\, x_{k,i,j_2,d,t_2} \vee \neg\, x_{k,i,j_3,d,t_3} \tag{2.30}$$

The constraints of 2.30 can easily be generalized. To assure at most $\lceil R \rceil$ lessons per day, we state that it's not allowed that $\lceil R \rceil + 1$ lessons occur each day. To generalize 2.29, we use the same trick as we did at the double period constraints. Let's say we want at least three lessons per day. We state that if the first lesson occurs, the second has to occur that same day on another period. Furthermore, we state that if the second lesson occurs, the third lesson has to occur that same day on another period. In this way, we assure at least three lessons on the same day. We get the next constraints:

$$\begin{aligned} x_{k,i,1,d,t} &\rightarrow x_{k,i,2,d,t'} \\ x_{k,i,2,d,t} &\rightarrow x_{k,i,3,d,t'} \end{aligned} \tag{2.31}$$

If we force the first three lessons to occur on the first available day, the second three lessons to occur on the second available day, etc, and we get the next constraints:

$$\bigwedge_{d|m_d>0} \bigwedge_{j=n+3(m_d-1)|n=1,2} \bigwedge_{t} x_{k,i,j,d,t} \rightarrow \bigvee_{t' \neq t} x_{k,i,j+1,d,t'} \tag{2.32}$$

where m_d tells us the number of the available day d. For instance, let a subject–teacher be available on Monday, Wednesday and Friday. Then $m_1 = m_{Monday} = 1, m_3 = m_{Wednesday} = 2$ and $m_5 = m_{Friday} = 3$. Also $m_2 = m_{Tuesday} = 0$ and $m_4 = m_{Thursday} = 0$. In general the constraints of 2.32 can be formulated as follows:

$$\bigwedge_{k} \bigwedge_{i|DoublePeriod(i)=0} \bigwedge_{d|m_d>0} \bigwedge_{j=n+\lfloor R \rfloor(m_d-1)|n=1,\ldots,\lfloor R \rfloor-1} \bigwedge_{t} \neg\, x_{k,i,j,d,t} \bigvee_{t' \neq t} x_{k,i,j+1,d,t'} \tag{2.33}$$

where $R = \frac{\sum_j A_k(i,j)}{number\ of\ available\ days\ of\ i}$

Of course, t and t' only refer to periods that the subject–teacher is available, but this took too much space. We also have to state the first lesson of each day to occur, just like in 2.29. This can be done in the next way:

$$\bigwedge_{k} \bigwedge_{i|DoublePeriod(i)=0} \bigwedge_{d|m_d>0} \bigvee_{t|TeacherAvailable(i,d,t)} x_{k,i,1+\lfloor R \rfloor (m_d-1),d,t} \tag{2.34}$$

In general the constraints of 2.30 can be formulated as follows:

$$\bigwedge_{k} \bigwedge_{i|DoublePeriod(i)=0} \bigwedge_{\substack{j_1|A_k(i,j_1)=1 \\ j_{\lceil R \rceil+1}|A_k(i,j_{\lceil R \rceil+1})=1}} \bigwedge_{d|m_d>0} \bigwedge_{t_1,\ldots,t_{\lceil R \rceil+1}} \neg x_{k,i,j_1,d,t_1} \vee \ldots \vee \neg\, x_{k,i,j_{\lceil R \rceil+1},d,t_{\lceil R \rceil+1}} \tag{2.35}$$

$$\text{where } R = \frac{\sum_j A_k(i,j)}{number\ of\ available\ days\ of\ i}$$

Within each clause of 2.35, $j_1, \ldots, j_{\lceil R \rceil+1}$ is a subset of $\lceil R \rceil + 1$ different elements of $\{1, 2, \ldots, U\}$ and $t_1, \ldots, t_{\lceil R \rceil+1}$ is a subset of $\lceil R \rceil + 1$ different elements of $\{1, 2, \ldots, T\}$, all only representing periods the subject–teacher is available.

Students idle period constraints: students have no idle periods. For most classes, there will be more periods in a week than necessary, which will lead to free periods for classes. If two occupied periods are separated by one or more free periods, the intermediate periods are called idle periods. It's obvious that the first period and the last period of a day can never be an idle period. The way schools handle idle periods can differ a lot. Some schools demand 'idle' periods for all classes at the same time. During this common 'idle' period, the students are free to work on any subject independently. Teachers are available to answer any questions. In this way, the schools try to teach the students to study independently. The Rotterdamsch Lyceum forbids idle periods in the junior department, which is a common way to deal with idle periods in the Netherlands. That's why we'll adopt these constraints.

Implemented constraints that forbid idle periods will lead to difficulties for the solver. Let's look at an example. If the fourth period of the first day of class k is a single idle period, this means that every variable with $d = 1$ and $t = 4$ is set to false, leading to the negation of D x U variables. It also means that a variable of the third period and a variable of the fifth period of the first day is set to true. To forbid that the fourth period of the first day is a single idle period, we'll state that if the third and the fifth periods are occupied, the fourth period is not allowed to be a free period. Forgetting about the teacher availability, this will lead to the next constraints for every possible combination of i_1, j_1, i_2 and j_2:

$$x_{k,i_1,j_1,1,3} \wedge x_{k,i_2,j_2,1,5} \rightarrow \neg \bigwedge_{i,j} \neg\, x_{k,i,j,1,4} \tag{2.36}$$

which can be transformed to

$$x_{k,i_1,j_1,1,3} \wedge x_{k,i_2,j_2,1,5} \rightarrow \bigvee_{i,j} x_{k,i,j,1,4} \tag{2.37}$$

stating that at least one lesson has to be given on the fourth period of the first day. These constraints can again be transformed into a disjunction of literals, required in CNF:

$$\neg\, x_{k,i_1,j_1,1,3} \vee \neg\, x_{k,i_2,j_2,1,5} \bigvee_{i,j} x_{k,i,j,1,4} \tag{2.38}$$

As the average number of lesson in the Rotterdamsch Lyceum is 30 these constraints will contain about 30 variables. The number of possible i_1 and j_1 will also be about 30, which leaves the number of possible i_2 and j_2 to about 29. This means that the number of clauses will be about 30 x 29 = 870 to assure that the fourth period of the first day is not an idle period for class k. But, with $T = 8$, there are 6 periods than can turn out to be an idle period. Furthermore, there are also 5 combinations of periods that can turn out to cause a double idle period and will need all clauses for one of the intermediate periods. The combinations of occupied periods which could lead to double idle periods are $\{1,4\},\{2,5\},\{3,6\},\{4,7\}$ and $\{5,8\}$. If one of the intermediate idle periods is occupied by the constraints which forbid two consecutive idle periods, the other intermediate idle period will be occupied, because of the constraints which forbid one idle period. Furthermore, there are 4 combinations of periods that can turn out to be a triple idle period etc. So we need $6 + 5 + 4 + 3 + 2 + 1 = 21$ times 870 clauses to assure no idle periods for one day of one class. To assure the absence of all sorts of idle periods for every day and for every class, with the quantities $K = 13$ and $D = 5$ for the Rotterdamsch Lyceum, we end up with 21 x 870 x 13 x 5 = 1,187,550 clauses. It can easily be imagined that this amount of clauses stating only these constraints will lead to some difficulties for the solver. But there is another point. To assure the absence of idle periods is actually an optimization constraint, which can easily end up unsatisfiable. We would like to state less clauses that will together approximate this requirement of no idle periods, but will also give the solver any space to solve the problem.

We saw that idle periods cannot occur at the beginning or at the end of a day. It will be clear that if a period in the middle of the day remains a free period, this can easily turn into an idle period. If we can avoid that the periods in the middle of the day will become free periods, we'll reduce the possibilities of idle periods. We already saw how it can be assured that the fourth period of the first day of class k is occupied:

$$\bigvee_{i,j} x_{k,i,j,1,4} \tag{2.39}$$

There are at most D x $(T-2)$ candidates for idle periods for class k, and the clauses should be stated for all K classes. For the Rotterdamsch Lyceum, this will lead to 5 x 6 x 13 = 390 clauses, which is only 0.03 % of 1,187,550! We now know, this will be a clever approach, we still have to decide of which periods we'll require to be occupied. We do this in a very simple way. For every class we wanted to occupy the same periods for every day. We took the minimum of the number of lessons of all classes in the junior department, which came to 28 lessons. As $D = 5$, we concluded that $\lfloor \frac{28}{5} \rfloor = 5$ periods per day could be occupied. Because the first period can never be an idle period, we stated that all periods between the second and the sixth are occupied:

$$\bigwedge_k \bigwedge_d \bigwedge_{t=2,\ldots,6 | TeacherAvailable(i,d,t)=1} \bigwedge_i \bigwedge_{j | A_k(i,j)=1} x_{k,i,j,d,t} \tag{2.40}$$

These constraints can of course differ per class or per day. If the timetable turns out to be unsatisfiable because of these constraints, we can of course try to require less periods to be occupied. Even though we secured that no idle periods occur during the middle of the day, there can still occur idle periods. We take these for granted and hope we can remove them by hand. Notice that as $T = 8$, we can only get idle periods at $t = 7$. We could also demand no idle periods for $t = 7$, which will not produce too much constraints and assures us the absence of idle periods.

Completeness constraints: every existing lesson does actually occur. Every combination of class k, subject–teacher i and lessonnumber j that corresponds to $A_k(i, j) = 1$ should occur on a period the subject–teacher is available.

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j|A_k(i,j)=1}\bigvee_{d}\bigvee_{t|TeacherAvailable(i,d,t)=1} x_{k,i,j,d,t} \tag{2.41}$$

Negation constraints: every non existing lesson does not occur.

These constraints are constructed in a similar way as the completeness constraints.

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j|A_k(i,j)=0}\bigwedge_{d}\bigwedge_{t|TeacherAvailable(i,d,t)=1} \neg\, x_{k,i,j,d,t} \tag{2.42}$$

By the construction of the variable $x_{k,i,j,d,t}$ all possible combinations of classes, subject–teachers, lessonnumbers, days and periods were introduced. All combinations that don't occur, so all lessons referring to zero elements of the allocationmatrix, should not be in the timetable. By these constraints, we forbid these combinations.

Uniqueness constraints: every lesson is scheduled only once. If a particular lesson could be assigned to several different periods, we only allow it to be assigned to one single period.

If a lesson $x_{k,i,j,d,t}$ occurs, then the same lesson cannot occur on another period.

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j|A_k(i,j)=1}\bigwedge_{d,d'}\bigwedge_{\substack{t|TeacherAvailable(i,d,t)=1\\ t'|TeacherAvailable(i,d,t')=1}} \neg\, x_{k,i,j,d,t} \vee \neg\, x_{k,i,j,d',t'} \tag{2.43}$$

As we've seen before, an extra constraint is needed. We also need to make sure that not if $d = d'$ then $t = t'$ too. This would lead to the same problems as at the teacher constraints. These constraints make sure that the same lesson cannot take place on several periods. By the completeness constraints, we make sure that every lesson occurs at least once, by these constraints, we make sure that every lesson occurs at most once.

2.5 Input to the solver

Once we've stated all constraints of the previous section, a timetable can be constructed by a solver by assigning variables. Besides the constraints, solvers require some extra information. They need to know the number of variables and the number of clauses in advance. Although we know the number of variables, K x D x U x S x T, we don't know the number of clauses in advance. Estimation will be very difficult, because the number of available periods of each

subject–teacher differs. The clauses can of course be counted as they are stated, but then we know the number of clauses at the end, whereas the solvers need that information in advance. If possible, the CNF-file can be opened and the number of clauses can be changed in the right number. But sometimes the CNF-file cannot be opened, because it's too large. In that case, we saw no other option than to state the constraints twice, the first time only to count the constraints, the second time in a new file to provide the required information in advance and to state the constraints.

We'll now give an example of what the output looks like. If the subject–teachers 1 and 2 refer to the same teacher, we need clauses from the teacher constraints. If the teacher teaches class 3 and is available on period 5 and on period 6 of day 4, we get the next clauses for these periods for the first lesson:

$$\begin{aligned} \neg\, x_{3,1,1,4,5} \quad &\vee \quad \neg\, x_{3,2,1,4,5} \\ \neg\, x_{3,1,1,4,6} \quad &\vee \quad \neg\, x_{3,2,1,4,6} \end{aligned}$$

By 2.2 and $K = 13, D = 60, U = 5, S = 5$ and $T = 8$ we get the single index of the variable:

$$\begin{aligned} x_{3,1,1,4,5} \quad &= \quad x_{24029} \\ x_{3,2,1,4,5} \quad &= \quad x_{24229} \\ x_{3,1,1,4,6} \quad &= \quad x_{24030} \\ x_{3,2,1,4,6} \quad &= \quad x_{24230} \end{aligned}$$

If these clauses are the first to be stated, we get the next start of the output:

```
p cnf 156000 number of clauses
-24029 -24229 0
-24030 -24230 0
...
```

Starting with the p-line, the line which contains information for the solver, it is stated that the formula is in CNF. The number of variables equals 156,000, whereas the number of clauses is yet unknown to us. According to [DIM 93] each clause is ended by a 0 and stated at a new line.

Chapter 3

Student sectioning

In this chapter we'll explain why student sectioning is needed and how it can be achieved using satisfiability by transforming the problem to a similar problem as period assignment. We'll use V4, the fourth grade of the vwo of the Rotterdamsch Lyceum to illustrate this transformation. The curriculum of V4 is illustrated in figure 3.1. The rows represent students and the columns represent subjects. If an element (i, j) of the curriculum contains a '1' student i has chosen subject j, otherwise the student has not. We'll end up with the right information to assign periods to the senior department, which will be dealt with in the next chapter.

	ne	la	gr	fa1	fa2	du1	du2	en	gs	ak	wa	wb	na	sk	bi	ec	ma	sp	lo	kcv	ck1	ck3	lob
F. el Arkoubi	1			1	1	1	1	1	1	1	1					1	1	1	1		1	1	1
N. Asut	1			1	1	1	1	1	1	1	1					1	1		1		1	1	1
W. Biharie	1	1		1		1		1	1		1	1	1	1	1	1	1		1	1			1
R. Bouzidi	1			1	1	1	1	1	1	1	1					1	1	1	1		1	1	1
F. Djorai	1			1	1	1	1	1	1	1	1				1	1	1		1	1	1	1	1
N. Gamhiouen	1	1	1	1	1	1	1	1	1	1	1				1	1	1		1	1		1	1
N. Girjasing	1	1	1	1		1		1	1		1	1	1	1	1	1	1		1	1		1	1
R. Groeneveld	1			1		1		1	1		1	1	1	1	1	1	1	1	1		1		1
D. Huisden	1	1	1	1	1	1	1	1	1	1	1				1	1	1		1	1		1	1
E. Kajmovic	1			1	1	1	1	1	1	1	1					1	1	1	1		1	1	1
H. Kelleci	1			1		1		1	1		1	1	1	1	1	1	1	1	1		1		1
A. Kewal	1			1	1	1	1	1	1	1	1				1	1	1		1		1	1	1
A. Mohan	1			1	1	1	1	1	1	1	1					1	1	1	1		1	1	1
W. Nejal	1			1		1		1	1		1	1	1	1	1	1	1		1		1		1
B. Petekci	1			1	1	1	1	1	1	1	1					1	1	1	1		1	1	1
R. Ramdat	1			1	1	1	1	1	1	1	1				1	1	1		1		1	1	1
S. Ramdat	1	1	1	1		1		1	1		1	1	1	1	1	1	1		1	1	1	1	1
C. Rouwers	1	1	1	1		1		1	1		1	1	1	1	1	1	1		1	1		1	1
G. Timmermans	1	1	1	1	1	1	1	1	1	1	1					1	1		1	1		1	1
V. de Veth	1			1	1	1	1	1	1	1	1					1	1	1	1		1	1	1

Figure 3.1: Curriculum of V4

3.1 Purpose of student sectioning

In the junior department the students were already divided into classes. This means that a class represents several students and all students within this class have mostly the same subjects, so we don't have to worry about the individual students. If we require that a class doesn't have two teachers at once, we'll be certain that the students within that class don't have two teachers at once.

In the senior department the students all have a compulsory part in their curricula, a common part and a free part. The students are divided into *years*, which are different grades of different schooltypes. For instance, the vwo contains six years of which the last three years form the senior department. The compulsory part is compulsory to all students in the same year and the common part is compulsory to all the students of the same year in the same profile. The free part of the curriculum contains subjects the students are freely to choose. The free part is the reason that often most students have different curricula, so therefore 'classes', as in the junior department, cannot be constructed. But to construct lessons we'll need a group of students. Because the term 'class' is not appropriate in this situation, we'll refer to a group of students taught at the same time by the same teacher as a *lessongroup*.

So this means the students have to be placed in lessongroups, which is called *student sectioning*. It's evident that students should only be placed in lessongroups of the subjects their curricula contain. Difficulties rise when more students choose a subject than a teacher can teach at the same time. The group of students has to be split into several lessongroups. These lessongroups of the same subjects should be almost equal in size. Furthermore, even though we don't have classes in the senior department, we still would like to have this structure in the senior department. Students tend to form a social group more quickly if they know each other. Forming almost similar lessongroups for several subjects could help this process. It's also an advantage to the teachers, because they can handle the same group of students in the same way.

There is another side to the lessongroup assignment. It is possible that lessongroups of different subjects are mutually disjoint concerning students and teachers. This would mean that the two lessongroups could be taught at the same time, assuming both teachers are available. Assigning two different lessongroups together with their teachers to the same period is called *clustering*. Let us say that lessongroup A and B are clusterable. If we don't cluster A and B, this would mean that if A is taking a class at a period, all students of B have a free period and if B is taking a class at a period, all students of A have a free period. Free periods could turn into idle periods, and we still want to have a small amount of idle periods. It is generally accepted that no idle periods for all students in the senior department is very hard to achieve, but we don't want to create idle periods if it's possible to avoid them. Moreover, if we don't cluster, A and B occupy two periods instead of one. This means that we'll need more periods if we don't cluster. Clustering is therefore an important part of the period assignment in the senior department. We saw that clustering is possible if lessongroups are mutually disjoint concerning students. This means that while constructing lessongroups, we want to try to create several combinations of lessongroups, which are mutually disjoint.

It is the most natural way to perform student sectioning per year. This means we need the curriculum of each year like figure 3.1, which combines all students curricula of the same year. We also need to know the maximum size of a lessongroup. Usually this is the same for every subject. If the maximum size of a lessongroup is M, we'll say that the lessongroups are split at $M + 1$. From now on we'll discuss student sectioning per year, so one should

remember that every action should be repeated for all the years in the senior department.

3.2 Transforming the curriculum

We start off with a curriculum for a year like figure 3.1, which shows of all students within that year which subjects they have chosen. The first thing we do is remove the compulsory subjects from the curriculum. This is not necessary, but if we don't the problem can become huge. So we decide to remove the compulsory subjects and construct the student sectioning of the compulsory subjects out of the student sectioning of the optional subjects, which are the subjects of the common and free part. If we take a look at the figure 3.1 we see that the subjects `ne, fa1, du1, en, gs, wa, ec, ma, lo` and `lob` should be removed from the curriculum.

We wanted to form lessongroups, which are similar for several subjects. So if there are subjects, which are chosen by exactly the same students, we surely want to form similar lessongroups. We achieve this by combining these subjects, which are chosen by exactly the same students, into so called *subjectgroups*. We'll now perform student sectioning for subjectsgroups instead of subjects, which implicates that for all subjects within a subjectgroup, the student sectioning is the same. A bonus is that the curriculum will decrease in size. At the curriculum of V4, we notice that the subjects `wb, na` and `sk` are chosen by exactly the same students. These subjects will therefore form one subjectgroup. The next subjectgroups were created from the curriculum of V4.

Subjectgroup $Subg_1$: `gr`
Subjectgroup $Subg_2$: `la`
Subjectgroup $Subg_3$: `wb, na` and `sk`
Subjectgroup $Subg_4$: `sp`
Subjectgroup $Subg_5$: `kcv`
Subjectgroup $Subg_6$: `bi`
Subjectgroup $Subg_7$: `fa2, du2` and `ak`
Subjectgroup $Subg_8$: `ck1`
Subjectgroup $Subg_9$: `ck3`

Just like the subjects, we can combine students. If students have the exact same curriculum, they're combined into *studentgroups*. All students in the same studentgroup will be assigned to the same lessongroups. This will also decrease the size of the curriculum. For instance, the first student of V4, F. el Arkoubi, has exactly the same curriculum as the fourth, R. Bouzidi. These students will be part of the same studentgroup.

If a studentgroup has chosen a certain set of subjects, and another student has chosen some of these subjects, but no other, he or she will also be added to this studentgroup. So from now on a studentgroup is a set of students who choose a set of subjects, or a part of this set of subjects. In this way students who almost have the same curriculum will attend the lessons together. We can see that student C. Rouwers has almost the same curriculum as S. Ramdat, only missing the subject `ck1`. This means that C. Rouwers will be added to the studentgroup of S. Ramdat. Of every studentgroup it is known how much students it contains. The next studentgroups were created from the curriculum of V4.

Studentgroup $Studg_1$	:	F. el Arkoubi, R. Bouzidi, E. Kajmovic, A. Mohan, B. Petekci and V. de Veth
Studentgroup $Studg_2$	:	R. Groeneveld and H. Kelleci
Studentgroup $Studg_3$	:	F. Djorai, N. Asut, A. Kewal and R. Ramdat
Studentgroup $Studg_4$	:	N. Gamhiouen, D. Huisden and G. Timmermans
Studentgroup $Studg_5$	:	S. Ramdat, W. Nejal, W. Biharie, N. Girasing and C. Rouwers

Once we've performed the actions mentioned in this section, we end up with table 3.1, which shows of every studentgroup how many students have chosen each subjectgroup.

Table 3.1: Acquired matrix after transformations

	$Subg_1$	$Subg_2$	$Subg_3$	$Subg_4$	$Subg_5$	$Subg_6$	$Subg_7$	$Subg_8$	$Subg_9$
$Studg_1$	0	0	0	6	0	0	6	6	6
$Studg_2$	0	0	2	2	0	2	0	2	0
$Studg_3$	0	0	0	0	1	3	4	4	4
$Studg_4$	3	3	0	0	3	2	3	0	3
$Studg_5$	3	4	5	0	4	5	0	2	3

We'll now show how student sectioning, i.e. assigning students to lessongroups of subjectgroups chosen by the students, is similar to period assignment in the junior department, i.e. assigning periods to the combination of classes and subject-teachers. At student sectioning we have to assign students to lessongroups of subjectgroups they've chosen. If we assign students to lessongroups, we also assign lessongroups to students, so student sectioning can interpreted as assigning lessongroups of subjectgroups to students, who have chosen the subjectgroups. Students are combined in studentgroups, and we've seen that we will make sure that every student in the same studentgroup will be assigned to the same lessongroup. When we put this all together, we see that student sectioning is assigning lessongroups of subjectgroups to studentgroups. By table 3.1 we've got the relation between subjectgroup and studentgroup, so in a way, student sectioning is assigning lessongroups of subjectgroups to a combination of studentgroups and subjectgroups. We now see the similarity. At the junior department we assigned periods to a combination of classes and subject-teachers, at student sectioning we assign lessongroups of subjectgroups to a combination of studentgroups and subjectgroups. Studentgroups can be interpreted as small 'classes' within each year, whereas subjectgroups can be interpreted as subjects.

We've seen that if we want to use this similarity, we should make sure that every student in the same studentgroup will be assigned to the same lessongroups, so we will need extra constraints. On the other hand we can skip several constraints we've used at period assignment, needing only the constraints which told the solver hat was expected of a timetable. We'll only use the completeness constraints, the negation constraints and the uniqueness constraints. The difference between student sectioning and period assignment is that at student sectioning we assign all students of the same studentgroup to the same lessongroup, whereas at period assignment, we assign all lessons of the same class to different periods. Because we assign several studentgroups to the same lessongroup, the lessongroup might become larger

than acceptable. So we need other constraints to make sure that the size of lessongroups doesn't exceed the maximum. When these two extra types of constraints are satisfied, we can tackle student sectioning in the same way as we tackled period assignment in the junior department. The achieved student sectioning only considers optional subjects, so to be clear, after that we'll also need student sectioning for compulsory subjects, which will be constructed out of the student sectioning for optional subjects.

3.3 Student sectioning optional subjects

Once we attained the matrix of table 3.1, we have to assign lessongroups of subjectsgroups to studentgroups. We have seen that we can use a similar approach as at assigning periods in the junior department. We therefore have to define the variables in the same way as in the junior department, only adjusted to the current problem. We first construct the allocationmatrix for student sectioning. Then we'll define the variables and finally we'll construct the constraints needed to achieve a student sectioning for optional subjects by the solver. As at period assignment, we'll first define quantities needed to create the allocationmatrix and the variable:

k	:	Studentgroup	with	$1 \leq k \leq K$
i	:	Subjectgroup	with	$1 \leq i \leq D$
j	:	Number of student within studentgroup k	with	$1 \leq j \leq U$
g	:	Lessongroup	with	$1 \leq g \leq G$

3.3.1 The allocationmatrix of student sectioning

At period assignment we had to distinct between the lessons of a subject, so we defined the index j as the number of the lessons of a subject. At student sectioning, we have to know the number of students within a studentgroup who have chosen a subjectgroup, because we have to make sure we don't exceed the maximum size of a lessongroup. We don't have to distinct between the lessons of subjectgroups, because that's part of period assignment. So now the columns of the allocationmatrix represent the students who have chosen some subjectgroups. The number of ones in row i is equal to the number of students who have chosen all subjects in subjectgroup i. We want to make clear that a column doesn't represent a particular student, because we don't order the students within a studentgroup. Analogous to the allocationmatrix in section 2.2 we define the allocationmatrix $B_k(i,j)$ of studentgroup k:

$$B_k(i,j) = \begin{cases} 1, & \text{if subjectgroup } i \text{ is chosen by a } j^{th} \text{ student of studentgroup } k; \\ 0, & \text{else.} \end{cases} \tag{3.1}$$

We interpret the elements of the allocationmatrix as follows: $B_k(i_1,3)$ represents the third student of studentgroup k who has chosen all subjects in subjectgroup i_1 and $B_k(i_2,3)$ represents the third student of studentgroup k who has chosen all subjects in subjectgroup i_2. It should be pointed out that this doesn't refer to a particular student, because columns of the allocationmatrix don't refer to particular students. Just like the allocationmatrix of period assignment, we have the next implications:

$$\bigwedge_k \bigwedge_i \bigwedge_{j>1} B_k(i,j) = 1 \rightarrow B_k(i,j-1) = 1 \tag{3.2}$$

$$\bigwedge_k \bigwedge_i \bigwedge_{j<U} B_k(i,j) = 0 \rightarrow B_k(i,j+1) = 0 \tag{3.3}$$

We didn't point them out at period assignment of the junior department, because at that time these implications were very clear: it's impossible to have a second lesson if a first lesson doesn't exist. In fact we mean the same at the implications above: there can't be a j^{th} student, if there's no $j-1^{th}$ student for $j > 1$, and if there's no j^{th} student, there's also no $j+1^{th}$ student for $j < U$. But for now, we state these trivial implications to avoid misconceptions.

3.3.2 The variables of student sectioning

We define the variable $y_{k,i,j,g}$ analogous to the variable $x_{k,i,j,d,t}$ at period assignment of the junior department:

$y_{k,i,j,g}$: the student corresponding to $B_k(i,j)$ is assigned to lessongroup g.

where k	:	Studentgroup	with	$1 \leq k \leq K$
where i	:	Subjectgroup	with	$1 \leq i \leq D$
where j	:	Number of student within studentgroup k	with	$1 \leq j \leq U$
where g	:	Lessongroup	with	$1 \leq g \leq G$

In other words, $y_{k,i,j,g}$ indicates that of studentgroup k at all subjects in subjectgroup i the j^{th} student is assigned to lessongroup g. For instance the propagation of variable $y_{1,2,3,4}$ makes sure that of studentgroup 1 at the subjects in subjectgroup 2 student number 3 is assigned to lessongroup 4, and $\neg\ y_{4,3,2,1}$ forbids that of studentgroup 4 at the subjects in subjectgroup 3 student number 2 is assigned to lessongroup 1. The total number of created variables is equal to K x D x U x G.

At period assignment of the junior department we had to transform the variable $x_{k,i,j,d,t}$ to x_n, because the solver couldn't cope with several indices. We'll do the same with our current variables. We transform the variable $y_{k,i,j,g}$ to y_n by the next formula:

$$n = g + G \text{ x } (j-1) + G \text{ x } U \text{ x } (i-1) + G \text{ x } U \text{ x } D \text{ x } (k-1) \tag{3.4}$$

3.3.3 Constraints of student sectioning

When we've created our variable $y_{k,i,j,g}$ we can state our constraints. We used the next five constraints, of which the first three are similar to the constraints of period assignment in the junior department. We'll state these first three constraints with only a short explanation.

Completeness constraints: every student should be assigned to a lessongroup of all subjectgroups he or she has chosen. If there are only $\mathcal{G} <$ G lessongroups needed, or in other words not every available lessongroup is necessary, we only assign the students to the used lessongroups, which are the lessongroups 1,2,...,$\mathcal{G}$.

$$\bigwedge_k \bigwedge_i \bigwedge_{j|B_k(i,j)=1} \bigvee_{g \leq \mathcal{G}} y_{k,i,j,g} \tag{3.5}$$

Negation constraints: a student should not be assigned to lessongroups of subjectgroups which he or she hasn't chosen.

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j|B_k(i,j)=0}\bigwedge_{g} \neg\, y_{k,i,j,g} \tag{3.6}$$

Uniqueness constraints: every student is assigned to only one lessongroup of a subjectgroup.

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j|B_k(i,j)=1}\bigwedge_{g\neq g'} \neg\, y_{k,i,j,g} \vee \neg\, y_{k,i,j,g'} \tag{3.7}$$

The completeness constraints don't prevent a student to be assigned to lessongroups that are not needed, i.e. to a lessongroup g, where $g > \mathcal{G}$. By stating the uniqueness constraints for all lessongroups, we prevent this from happening, because a student has to be assigned to one of the used lessongroups by the completeness constraints and is only assigned to one lessongroup by the uniqueness constraints.

Studentgroup binding constraints: all students of the same studentgroup who have chosen the same subjectgroups should be assigned to the same lessongroups. If a student is assigned to a lessongroup, than all other students of the same studentgroup should be assigned to that lessongroup. We can accomplish this by forcing the next sequence: if the first student is assigned to a lessongroup, so will the second student. If the second student is assigned to a lessongroup, so will the third, etc. Informally stated, we get the next implications, where J is the number of students:

$$\bigwedge_{j<J} y_{k,i,j,g} \rightarrow y_{k,i,j+1,g} \tag{3.8}$$

These implications are sufficient to meet our demands, because every student has to be assigned to a lessongroup. So if the second student is assigned before the first student, this won't have an immediate implication for the first student. But if the first student is assigned to another lessongroup, this will lead to a change of lessongroups for the other students, if needed and possible. We get the next constraints for the needed $\mathcal{G}$ lessongroups, where J_i is the number of students in studentgroup i:

$$\bigwedge_{k}\bigwedge_{i}\bigwedge_{j<J_i|B_k(i,j)=1}\bigwedge_{g\leq\ \mathcal{G}} \neg\, y_{k,i,j,g} \vee y_{k,i,j+1,g} \tag{3.9}$$

Lessongroup-size constraints: each lessongroup cannot contain more than M students. If a subjectgroup is chosen by less than M students, we'll don't have to state any constraints. So we restrict ourselves to the case where more students have chosen a subjectgroup than a teacher can teach at the same time. We'll illustrate our approach with an example. Let's say that 60 students have chosen a subjectgroup and $M = 32$. In this case there should be $\lceil \frac{60}{32} \rceil = 2$ lessongroups. We should assign about equally amount of students to each lessongroup. The optimal groupsize of each lessongroup will be $\frac{60}{2} = 30$, but if this is impossible, we'll allow a difference of 1 with respect to the optimal groupsize. We'll say we allow a margin of 1. Note that this could mean a difference of two students between two lessongroups. So now we know the maximum groupsize of this particular lessongroup, which equals $30 + 1 = 31$. Let's say the 60

students are spread over four studentgroups, containing 18, 12, 19 resp. 11 students. We now take all combinations of two studentgroups. If the sum of all students is larger than the maximum lessongroup-size, we forbid this combination. We need $\binom{4}{2} = 6$ calculations, as is shown in table 3.2. We see that we should forbid studentgroup 1 and

Table 3.2: Decision for each combination of two studentgroups

studentgroups	sum of students	conclusion
1 and 2	18 + 12 = 30	allow combination
1 and 3	18 + 19 = 37	forbid combination
1 and 4	18 + 11 = 29	allow combination
2 and 3	12 + 19 = 31	allow combination
2 and 4	12 + 11 = 23	allow combination
3 and 4	19 + 11 = 30	allow combination

studentgroup 3 to be combined. But we've only checked the groupsize for combinations of two studentgroups. We should do the same for three studentgroups. This could lead to $\sum_{i=2}^{3} \binom{4}{i} = 10$ combinations. In general, we could get for n studentgroups $\sum_{i=2}^{n-1} \binom{n}{i} = 2^n - n - 2$ combinations at most. We hopefully won't get to this amount of combinations, because we can skip some combinations. If the combination of studentgroup 1 and 3 are forbidden, this implies that every combination of three studentgroups containing these two studentgroups is also forbidden. So we can skip all these combinations. A bonus is that we don't state unnecessary clauses. A difficulty here is that we have to keep up all forbidden combinations and have to compare all candidates for new combinations with the already forbidden combinations. We see that especially these constraints need a lot of comparisons. That's the reason why we excluded the compulsory subjects from student sectioning for now: the compulsory subjects will require the most if not too much comparisons.

We'll now show how to state our constraints. As said, these constraints are only required for subjectgroups of which the number of students is larger than M. If we don't want several studentgroups together at a lessongroup, we only have to forbid that the first students of each studentgroup are not combined in the same lessongroup, because all students in the same studentgroup are assigned to the same lessongroup. Let's say that $\mathbb{K}$ contains all sets $\mathcal{K}_1, \ldots, \mathcal{K}_{|\mathbb{K}|}$, which contain minimal sets of studentgroups, which are not allowed to be together in the same lessongroups. For instance, if $\mathcal{K}_1 = \{1, 2, 3\}$, then we don't want studentgroup 1, studentgroup 2 and studentgroup 3 to be assigned to the same lessongroup. The studentgroups contained within $\mathcal{K}_k \in \mathbb{K}$ are called $\mathcal{K}_k^1, \ldots, \mathcal{K}_k^{|\mathcal{K}_k|}$. Let's say that $\mathbb{I}_{\mathcal{K}_k}$ contains all subjectgroups of which $\mathcal{K}_k$ consist of a forbidden combination of studentgroups. We can state the constraints which make sure that the maximum groupsize is respected for all subjectgroups in the following way for the needed $\mathcal{G}$ lessongroups:

$$\bigwedge_{\mathcal{K}_k \in \mathbb{K}} \bigwedge_{i \in \mathbb{I}_{\mathcal{K}_k}} \bigwedge_{g \leq \mathcal{G}} \neg\, y_{\mathcal{K}_k^1, i, 1, g} \vee \ldots \vee \neg\, y_{\mathcal{K}_k^{|\mathcal{K}_k|}, i, 1, g} \tag{3.10}$$

These are the only constraints we use to satisfy the lessongroup-size constraints. Several questions can be asked. First of all, don't we need a minimum lessongroup-size, we only stated clauses for the maximum lessongroup-size? It is of course possible when we have a lot lessongroups and several lessongroups have maximum size, the difference in size between several lessongroups can be pretty larger. Yet we think this would take a lot lessongroups and won't be likely to happen.

Secondly, what if the studentgroups have such an awkward size that they cannot be combined at all? Let's say we have 60 students, which should be placed in 2 lessongroups with at most 31 students each, and we have 3 studentgroups, each containing 20 students. In this case we can never satisfy the lessongroup-size constraints. The answer is clear: we need different studentgroups. So adjustments in student sectioning should be made. Many problems can rise at this point of timetabling, but we feel that with a little creativity a solution can be found for every problem.

Now we've stated all the constraints, we can turn to the solver to perform student sectioning for optional subjects. Just like at period assignment, we know the number of variables, but we can't in advance tell the number of constraints, so we'll have to change this number afterwards or state the constraints twice.

3.4 Student sectioning compulsory subjects

If the solver has performed student sectioning for the optional subjects, we can try to perform student sectioning for the compulsory subjects. There are of course many ways, but we follow the next steps, keeping in mind that students of the same studentgroup should be assigned to the same lessongroups and that it's desirable to achieve as many mutually disjoint lessongroups as possible, but we also have to assure that the difference in size is small. We start off with the student sectioning of the subjectgroup, which was chosen by the most students. If we copy the student sectioning of the optional subjects to the compulsory subjects, we have to assign as little as possible ourselves.

1. Complete studentgroups:
   Because all students within the same studentgroup should be assigned to the same lessongroup, we add the not yet assigned students to the lessongroups, to which the other students of their studentgroups are assigned. We end up with total studentgroups that are not yet assigned.

2. Construct new lessongroups if needed:
   It is possible that more lessongroups are needed at the compulsory subjects than at the optional subject. At this stage we create these extra lessongroups.

3. Add not yet assigned studentgroups to lessongroups:
   Each time we pick the largest studentgroup and perform the same actions. If the difference in size of the lessongroups is larger than the size of the studentgroup, we assign the studentgroup to the smallest lessongroup. In this way the lessongroups will remain almost of equal size. If not, then we count how often the current studentgroup is combined with the already assigned studentgroups. We assign the studentgroup to the lessongroup, which contains the most studentgroups the not yet assigned studentgroup is combined with.

If there are more than one compulsory subjects, as is the case in The Netherlands, we can of course cluster the lessongroups of the compulsory subjects. But if we try to assign student-groups in this way, we might be able to create more possibilities to cluster lessongroups. And the more possibilities the better.

3.5 A lessongroup-based curriculum

Now we've performed the student sectioning for all subjectgroups, we have to transform the acquired information into a format we can use to assign periods. But to assign periods we need a usable curriculum. We will construct a curriculum which assigns a number of lessons of a lessongroup of a subject–teacher to studentgroups. With this curriculum, we can assign periods as if it were the junior department, along with the constraints that all studentgroups in the same lessongroup should be assigned to the same period for that subject. With these constraints, we will state enough to assign periods to the senior department. Notice that we don't force the solver to cluster lessongroups. We feel the lack of periods will force the solver to do so. This means that we don't get an 'optimal clustered' timetable, but a 'sufficient clustered' timetable.

We start off with a curriculum like table 3.1. The first thing we do is to split subjectsgroups to attain subjects and we split each subject into U lessongroups. Notice that not all of these U lessongroups have to be used, but we need to construct them because of the way we constructed the variable $y_{k,i,j,g}$. We know the curriculum with respect to the subjects of each year, i.e. the total number of lessons to be taught at every subject, but it's not yet split to studentgroups and lessongroups. For instance, when there are two subjectgroups needed at the subject `wb` and the curriculum tells us there have to be six lessons of `wb`, three lessons will be assigned to each lessongroup of the subject `wb`. In general, if a subject requires a lessons to be taught and the subject requires n lessongroups, each lessongroup will be $\frac{1}{n}a$ lessons assigned. We know which subjectgroups each studentgroup has chosen, so we also know which subjects each studentgroup has chosen. To each studentgroup which has chosen a certain subject, we assign $\frac{1}{n}a$ lessons to the lessongroups the studentgroup is assigned to, where n is the number of used lessongroups for that subject and a is the number of lessons assigned to the year for that subject.

We'll illustrate the above with an example. We've got the curriculum in table 3.3 for a certain year:

Table 3.3: Mini curriculum

subject	`ne`	`en`	`fa`	`du`
number of lessons	9	4	4	3

The subjects couldn't be combined into subject–groups. We can also say that each subject–group consist of one subject. We know that the subject `ne` requires three lessongroups, so also three lessongroups are assigned to the other subjects, and the year exists of three studentgroups, which are assigned to the lessongroups of the subjects in table 3.4. As we follow the directions above, we get our lessongroup-based curriculum, shown in table 3.5.

This lessongroup-based curriculum looks a lot like the curriculum we worked with in the junior department. The only extra required constraints will be about making sure that

Table 3.4: Distribution of studentgroups over lessongroups of subjects

subject	`ne`			`en`			`fa`			`du`		
lessongroup	1	2	3	1	2	3	1	2	3	1	2	3
studentgroup 1			x				x					
studentgroup 2	x			x			x					
studentgroup 1		x			x					x		

Table 3.5: Lessongroup-based curriculum

subject	`ne`			`en`			`fa`			`du`		
lessongroup	1	2	3	1	2	3	1	2	3	1	2	3
studentgroup 1			3				4					
studentgroup 2	3			2			4					
studentgroup 1		3			2					3		

studentgroup 1 and studentgroup 2 are assigned to the same period at all of the four lessons of the subject `fa`.

Chapter 4

Period assignment senior department

When student sectioning has been completed, periods can be assigned in the senior department. We will first look at the consequences student sectioning has had. Then we will define the allocationmatrix and the variables to assign periods in the senior department equivalently to the junior department. The constraints are also similar. But we forget about the students idle period constraints and we add some other required constraints. Then we will show what transformations are needed to be able to assign periods in the junior department after assigning periods in the senior department.

4.1 Impact of student sectioning

We've seen that student sectioning ended with a lessongroup-based curriculum per year. The lessongroup-based curricula of each year are stacked to create one large lessongroup-based curriculum for the senior department. The rows of this curriculum represent all studentgroups in the senior department. We've seen that we can consider a studentgroup to be a small 'class' within a year. At the Rotterdamsch Lyceum, we had 13 classes in the junior department, but after student sectioning we constructed 50 studentgroups out of the six years in the senior department. This shows why the senior department is the bottleneck of timetabling. Another important reason of this fact is that studentgroups within a year are related to each other, opposite to the classes in the junior department. The columns of the lessongroup-based curriculum represent all lessongroups of all subject–teachers. Even though some subject–teachers don't need all the lessongroups, still the maximum number of lessongroups G has been assigned to every subject–teacher. This means that if we started with D_1 subject–teachers in the senior department, we now have $D = G$ x D_1 columns in the curriculum. At the Rotterdamsch Lyceum, we had 60 subject–teachers in the junior department. In the senior department we had 64 subject–teachers, so we ended up with 128 columns in the curriculum, as $G = 2$. This also shows the size of the senior department exceeds the junior department by far. To keep as close to the period assignment of the junior department as possible, we will again refer to the columns as subject–teachers, which they are, but we won't need of them to be known which lessongroup they're assigned to.

4.2 Allocationmatrix and variables

The allocationmatrix and the variables of period assignment in the senior department are similar constructed as at period assignment in the junior department. We'll only show them for the sake of completeness.
The allocationmatrix of studentgroup k is defined as follows:

$$A_k(i,j) = \begin{cases} 1, & \text{if subject–teacher } i \text{ is teaching a } j^{th} \text{ lesson to studentgroup } k \text{ ;} \\ 0, & \text{else.} \end{cases} \tag{4.1}$$

The variables of period assignment in the senior department are defined as follows:

$x_{k,i,j,d,t}$: the lesson corresponding to $A_k(i,j)$ is assigned to period t of day d;

where k	:	Studentgroup	with	$1 \leq k \leq K$
where i	:	Subject–teacher in the senior department	with	$1 \leq i \leq D$
where j	:	Number of a lesson of a subject	with	$1 \leq j \leq U$
where d	:	Day	with	$1 \leq d \leq S$
where t	:	Period	with	$1 \leq t \leq T$

4.3 Constraints of period assignment junior department

Most of the constraints needed to assign periods to the senior department are similar to the constraints of the junior department. We'll state only their descriptions. For further information we refer to section 2.4. It should be noticed that the constraints of section 2.4 refer to classes.

Teacher constraints: every teacher can have at most one lesson on the same period. We know that teachers in the senior department teach several studentgroups of the same year at the same time, so we have to make some adjustments. We handled this problem in the next way. At stating these constraints, for every lessongroup which contained more than one studentgroup, we erased all studentgroups of the same year from the lessongroups in the lessongroup-based curriculum except for the first. The constraints are now still stated for different lessongroups, which is required, but not within a lessongroup. Besides, the fix lessongroups constraints, which will follow, make sure all studentgroups of the same lessongroup are assigned to the same periods. If for every lessongroup we assure no different lessons of the same teacher to be taught at the same period to the first of the studentgroups, we implicitly assure no different lessons of the same teacher to be taught at the same period. This trick can also be used in other constraints.

Student constraints: every student can have at most one lesson on the same period.

Teacher availability constraints: a lesson can only be taught on a period the teacher is available.

Double period constraints: two lessons of particular subjects, like drawing or physical education, will be assigned to two consecutive periods.

Room compatibility constraints: only rooms of a suitable type are assigned to each lesson. If several subject–teachers need the same room, we have to keep this in mind at assigning periods. At these constraints we also need to adjust the lessongroup-based curriculum by erasing all studentgroups from the lessongroups except for the first, like at the teacher constraints.

Educational constraints: lessons of the same subject are distributed over the week as much as possible.

Completeness constraints: every existing lesson does actually occur.

Negation constraints: every non existing lesson does not occur.

Uniqueness constraints: every lesson is scheduled only once.

The only constraints missing are the students idle period constraints. These constraints can be seen as optimization constraints and will often not be able to be satisfied. Therefore we don't state these constraints. And idle periods are accepted in the 'studiehuis', the senior department of the havo and vwo, because during these idle periods students can study independently, which is one of the aims of the studiehuis. If we think of it we can imagine why it's so hard to satisfy these constraints. The students idle period constraints demanded all periods in the middle of the day to be occupied. But for the senior department this means that within each year all periods in the middle of the day should be occupied by lessons which are mutually disjoint concerning studentgroups. And these clusters should also contain all studentgroups. One can imagine this is very hard to satisfy. Besides, if we even managed to excluded idle periods for each studentgroup, we still don't know if each student has no idle periods, because we construct timetables based on studentgroups and not on students. Studentgroups contain students who don't follow every subject the other students of their studentgroup have chosen, so they would probably have idle periods anyway.

There are some constraints which only occur in the senior department. The next constraints were also stated to assign periods in the senior department.

Fix lessongroups constraints: lessons of studentgroups of the same lessongroups are assigned to the same periods. This can be interpreted as if a lesson is assigned to one of the studentgroups, the same lesson of the other studentgroups also have to be assigned to the same period. Notice that if k and k' are studentgroups of the same year, they're in the same lessongroup if they're having the same amount of lessons in the same column in the lessongroup-based curriculum. This is equivalent to having the same amount of one's in the corresponding row of the allocationmatrix. If two studentgroups are in the same lessongroups can be checked by comparing corresponding rows of each of their allocationmatrices. Let $\mathbb{Y}$ be the set of sets of all studentgroups of all subject–teachers and $\mathcal{Y}$ a set of all studentgroups of a subject–teacher, so $\mathbb{Y} = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_{|\mathbb{Y}|}\}$, then we get the next constraints.

$$\bigwedge_{\mathcal{Y}\in\mathbb{Y}} \bigwedge_{k\neq k'\in\mathcal{Y}} \bigwedge_{i} \bigwedge_{j|\substack{A_k(i,j)=1\\A'_k(i,j)=1}} \bigwedge_{d} \bigwedge_{t|TeacherAvailable(i,d,t)=1} \neg\, x_{k,i,j,d,t} \vee x_{k',i,j,d,t} \tag{4.2}$$

Block periods constraints: We'll use the Rotterdamsch Lyceum to illustrate these optimization constraints. Even though we needed 8 periods per day in the junior department, we need 9 periods per day in the senior department. Studentgroups of some

years have many lessons, and together with the idle periods, it was inevitable to allow 9 periods a day. But not every studentgroup needed 9 periods a day. Some studentgroups had less than 26 lessons per week. If we leave all options to the solver, the solver might spread these 26 lessons over the 45 periods, which could create an awful lot of idle periods. So for studentgroups with less than 26 lessons per week, we could for instance block the eighth and ninth period of each day. As the number of days per week S equals 5, this leaves 7 x $S = 35$ periods, which hopefully is enough for the solver to assign 26 or less lessons. Also if studentgroups had 26 or more but less than 31 lessons, we blocked the ninth period. Let's say $\mathcal{E}$ contains all studentgroups of which the eighth and the ninth periods of each days have to be blocked, and let's say $\mathcal{N}$ contains all studentgroups of which only the ninth period of each day has to be blocked. We get the next two constraints:

$$\bigwedge_{k\in\mathcal{E}} \bigwedge_{i} \bigwedge_{j|A_k(i,j)=1} \bigwedge_{d} \bigwedge_{t=8,9|TeacherAvailable(i,d,t)=1} \neg\, x_{k,i,j,d,t} \tag{4.3}$$

$$\bigwedge_{k\in\mathcal{N}} \bigwedge_{i} \bigwedge_{j|A_k(i,j)=1} \bigwedge_{d} \bigwedge_{t=9|TeacherAvailable(i,d,t)=1} \neg\, x_{k,i,j,d,t} \tag{4.4}$$

These constraints are of course meant to optimize the timetable. They can be adjusted to every situation and every desire, but should leave the solver enough space to find a satisfiable assignment.

4.4 From senior department to junior department

When all constraints are stated and the solver has found a satisfiable assignment, we got ourselves a period assignment of the senior department. Then we still have to assign periods to the junior department. We will show two ways how this can be achieved. At both ways we'll see we'll have to transform the assigned variables, because the quantities K and D will change, and U might change.

We start off with the lessongroup-based curriculums for the senior department, which contained all lessongroup-based curricula of every year. We know that if we got a curriculum of the whole school, we can construct an allocationmatrix and variables. Then we hopefully can assign periods to the junior department, taking the period assignment of the senior department into account. So we need to add the curriculum of the junior department to the curriculum of the senior department. The rows of the curriculum of the junior department represent the classes and the columns represent the subject–teachers of the junior department. We considered the way showed in table 4.1 to combine the curricula to be useful.

Table 4.1: Curriculum of whole school

	subject–teachers of senior department	subject–teachers of junior department
studentgroups	lessongroup-based curriculum	0
classes	0	curriculum of junior department

If for instance a subject–teacher teaches in the senior department and in the junior department, we could also add the number of lessons in the junior department to a particular

class in the corresponding row in the column of the subject–teacher in the senior department. Now we create an extra column in the junior department, which expands the curriculum, but which is more clear.

If we take a look at the curriculum, we see that it's number of rows is the sum of the number of rows of the separated curricula and so is the number of columns. At the Rotterdamsch Lyceum, the number of studentgroups equals 50, and the number of classes equals 13. The number of subject–teachers in the senior department is 64, which leads to 128 columns, because to every student–teacher $G = 2$ lessongroups are assigned. The number of subject–teachers in the junior department is 60. This leads to a matrix of size 63 by 188. If we take a look at the number of variables, we also need U, S and T. Although U equals 4 in the senior department, U equals 5 in the junior department, so we need $U = 5$. Together with $S = 5$ and $T = 9$ we end up with K x D x U x S x T = 2,664,900 variables. Of course, all variables in the senior department are assigned and a lot other variables should be set to false, because of the 'zero-matrices' needed to add the curriculum of the junior department to the curriculum of the senior department. But still, the amount of variables is huge.

It should be noted that because K, D and U change, the numbering of variables has changed. This means that every variable in the senior department should be transformed before it can be propagated or negated at assigning periods in the junior department.

After assigning periods to the junior department, it's possible that we want to adjust the assignment of the senior department. We then also need the constraints of the senior department to be stated along with the constraints of the junior department. This doesn't bother the solver, because we also state a satisfying assignment of the senior department, namely the one we just found in the period assignment of the senior department. It will be clear that the amount of clauses will be huge. For instance, only the assigning of variables of the senior department will lead to 50 x 188 x 5 x 5 x 9 = 2,115,000 clauses for the Rotterdamsch Lyceum. When one recalls that the Rotterdamsch Lyceum is considered a 'small' school, we can imagine this just might work for the Rotterdamsch Lyceum, but for other larger schools these amounts will be too much to handle for solvers.

But it can be questioned if we want to adjust the period assignment of the senior department after assigning periods to the junior department. We can also adjust the assignment of the senior department before the assignment of the junior department, and consider the assignment of the senior department as definite. From experience I can tell that adjusting the period assignment of the senior department is practically impossible, especially because of the clusters. Small changes are possible, but as said, these can also be made before the period assignment of the junior department.

If we indeed decide not to state the constraints of the senior department, we can go one step further. Do we actually need the assignment of each studentgroup separately ? If we indeed consider the period assignment in the senior department as definite, we only need to know the period assignment of the teachers. It's not important which studentgroups the teacher is teaching, it's only important that a teacher is occupied for several periods and that some particular rooms are occupied. This means we can get rid of the studentgroups and go back to the curricula per year. Now every year only takes one row of the curriculum of the senior department, and each element equals the maximum in the corresponding column of the lessongroup-based curriculum. The Rotterdamsch Lyceum has 6 years in the senior department. If we want to compare this with the other option, we have 19 x 188 x 5 x 5 x 9 = 803,700 variables in total to assign periods to the junior department, which is 30% of 2,664,900. To state the assignment of the senior department, we need $\frac{6}{50}$ = 12% of the

2,115,000 clauses we needed at the other option. It should be pointed out that because no constraints are stated for the senior department, the true and also the false assigned variables referring to the senior department have to be stated.

We showed two ways how the period assignment of the senior department can be integrated in the period assignment of the junior department. If the constraints and the assignment to the studentgroups are also stated, the amount of clauses and variables will become huge, but adjustments after assigning periods to the junior department will be possible. On the other hand, if we don't require this possibility, the amounts are relatively small, and will hopefully lead to a solution found quickly by the solver. This option is the one which we've used to continue our search for a timetable.

Chapter 5

Room assignment

Now we've assigned periods to all lessons, we only have to assign rooms and we got ourselves a timetable. We will show why we don't use SAT to assign rooms. Instead we'll use a simple algorithm to assign rooms. Then we will describe our demands and the way we handle room assignment. Our starting point will be that there are enough rooms available at all periods. If not, we should have stated extra constraints at the period assignment, which could cause a lot of clauses.

5.1 Why don't we use satisfiability?

Rooms can of course be assigned in the same way we assigned periods or lessongroups. This could work, although we would get a huge amount of variables. At the Rotterdamsch Lyceum the number of rooms $|R|$ equals 29, so we would get K x D x U x S x T x $|R|$ = 803,700 x 29 = 23,307,300 variables. But we only have to assign rooms to lessons which actually occur, so we could transform the period assigned variables in a way we would only assign rooms to these variables, which will decrease the amount of variables tremendously.

But we don't use satisfiability, because we have two desires at assigning rooms. We already saw that desires can be difficult to satisfiability, because they have to be stated as requirements, which can easily end up with an unsatisfiable formula. The first desire won't be a problem. We want every room to be assigned to a group of students, that actually fit in that room. This can easily be stated in CNF, because the groupsize of each class or lessongroup is known. But we also want that every teacher is assigned to the same room as much as possible. Some teachers demand their own room, so for these teachers this desire will be satisfied. But for all other teachers, this will be a bit of a problem. We already saw that stating 'as much as possible' in CNF won't be possible, but how should we state these constraints ? We would get constraints, stating at least x rooms will be the same, or if two lessons are consecutive, the assigned rooms should be the same. This can easily end up in unsatisfiable, also because we have to take the groupsize in account. If two lessons are consecutive, but the groupsize differs, so different rooms might be needed, this will cause difficulties.

So the reason we don't use satisfiability to assign rooms is that it's not flexible enough. But an algorithm to assign rooms is easily imagined, so we will tackle this problem in another way.

5.2 Constraints of room assignment

Even though we don't use satisfiability to assign rooms, we still have to satisfy some requirements and desires. We already stated the desires in section 5.1:

Room capacity constraints: the group of students have to fit in the assigned room.

Same room constraints: teachers are assigned to the same room as much as possible

There are also some requirements which have to be met. Some of these will actually make the room assignment easier.

Room constraints: every room can have at most one lesson on the same period.

Room compatibility constraints: lessons are only assigned to suitable rooms. We already saw we've took these constraints into account at period assignment, so satisfying these constraints will be no problem. For instance, because we've already made sure we didn't assign too many lessons of physical education to the same periods, we will have no difficulties at assigning rooms to the lessons of physical education. The same can be said about drawing and crafts. Often these rooms are also only suitable for certain subjects. We should for instance not assign a lesson mathematics to the gym.

Class own room constraints: several classes of the Rotterdamsch Lyceum in the junior department are assigned to their own room. So every lesson takes place in that particular rooms, except for physical education, drawing and crafts. These constraints are of course easily satisfied, and will actually decrease the number of lessons to which rooms have to be assigned, so will make the problem easier.

Teacher own room constraints: also several teachers have their own room, especially teachers in the senior department. These rooms will also be easy to assign, only the room capacity constraints can cause difficulties. But usually teachers claim rooms that fit all of their lessons.

Completeness constraints: every existing lesson does actually take place in a room.

Uniqueness constraints: every lesson takes place in one single room.

5.3 Assigning rooms

We first create the matrix Assign. It's rows represent all lessons, that don't have a room yet. It's columns represent the indices k, i, d and t of the lessons. Each time we assign a room to a lesson, we remove the indices of this lesson from the matrix Assign. The number of rooms will be referred to as $|R|$. When Assign is empty, we've assigned rooms to all lessons.

5.3.1 Room allocationmatrix

We create the room allocationmatrix $R(p, r)$. The rows of $R(p, r)$ represent a period of the week, so there will be S x T rows. The columns represent the different rooms, so there will be $|R|$ columns. We start off with an all-zero allocationmatrix. Each time a room is assigned

to a lesson on a certain period, we change the corresponding element of $R(p, r)$ into the subject–teacher i of the lesson. Formally,

$$R(p,r) = \begin{cases} i, & \text{if room r has been assigned to subject–teacher } i \text{ on period p;} \\ 0, & \text{if room r has not been assigned.} \end{cases} \tag{5.1}$$

Of each lesson in the matrix Assign, we know the day d and the period t of day d it occurs. If we want to assign a room, we have to know which row of $R(p, r)$ we have to consider. The corresponding row p of $R(p, r)$, when the lessons has indices d and t, can be found in the next way: $p = (d - 1)$ x $T + t$.

We'll give a little example. Let's say a school has only two days per week, each containing two periods, and five available rooms. We construct the next initial allocationmatrix shown in table 5.1.

Table 5.1: Initial room allocationmatrix $R(p, r)$

	r_1	r_2	r_3	r_4	r_5
p_1	0	0	0	0	0
p_2	0	0	0	0	0
p_3	0	0	0	0	0
p_4	0	0	0	0	0

If we assign subject–teacher i_1 to room r_3 on the first period of the first day and to room r_4 on the second period of the second day, and if we assign subject–teacher i_2 to room r_1 on every period of the week, we get the allocationmatrix shown in table 5.2.

Table 5.2: Adjusted room allocationmatrix $R(p, r)$

	r_1	r_2	r_3	r_4	r_5
p_1	i_2	0	i_1	0	0
p_2	i_2	0	0	0	0
p_3	i_2	0	0	0	0
p_4	i_2	0	0	i_1	0

5.3.2 Assigning subject–related rooms

We saw we can immediately assign the rooms which are only suited for certain lessons, like physical education, drawing and craft. Because we know which subject–teachers need the rooms, we can easily find the corresponding lessons in the matrix Assign. When we've found a lesson, we also know at what period this lesson takes place and we can assign the room by changing the corresponding element of the room allocationmatrix into the number of the subject–teacher. Finally, we remove the lesson from the matrix Assign.

5.3.3 Assigning own rooms

Just as easy as we assigned the rooms which are only suited for certain lessons, we can assign the own rooms of classes. For every row of the matrix Assign we can determine by the first index k if it refers to a class with an own room. If it does, we assign the required room and remove the row from the matrix Assign.

We can also easily assign own rooms to teachers. If a teacher with an own room has to teach a class with an own room, one has to look at the situation which room will be assigned. This problem won't be difficult to solve.

5.3.4 Assigning rooms to remaining lessons

It will be a bit harder to assign rooms to the remaining lessons. Our procedure will be simple: search for every teacher for the room which has the most free periods at his or her assigned periods and assign this room. If there are still lessons without a room, find the room which has the most free periods at the remaining lessons and repeat the procedure until all lessons of the teacher have a room. We'll have to find an order of the teachers by which the room capacity constraints are satisfied and the same room constraints are met in a satisfying way.

Often we can distinguish between large groups and normal groups and large rooms and normal rooms. We will first assign the large rooms to the large groups, because if we don't this might give some difficulties at the end. Then we take a look at the remaining lessons. We sort the teachers in descending order considering the amount of lessons and start to assign rooms to the teacher with the most lessons. For every teacher we try to find the rooms which suits his or her lessons best. If two rooms are equally suited, we take the smallest one. In this way, it's impossible we assign teachers with a lot lessons, but with relatively small groups to a relatively large room, which might be needed for other teachers with less lessons.

Chapter 6

Adjustments

There are four kinds of adjustments to the timetable we will consider. The first two adjustments consider the period assignment. The third considers the room assignment. The fourth kind considers daily adjustments, which are adjustments needed because of the absence of classes or teachers, and which will only last for a day. Adjustments like daily adjustments can be needed for more than one day. We'll also consider multiple daily adjustment. In specific, we'll look at weekly adjustments.

6.1 Introduction

It's an illusion to think that the timetable created by the solver will be perfect in the eyes of the timetable designer. We have the list of variables, all assigned to true or false, and if we want we can change the assignment of variables to change the timetable to our desires. Because we didn't exclude all possible idle periods for the junior department and we didn't state some optimality constraints for the timetables of the teachers, the timetable will contain several undesirable features. For instance, teachers may have three or four consecutive idle periods. The requirements we didn't think of as important to state as constraints can now be violated in a way we cannot allow. Therefore, it should be possible to adjust the timetable. Besides, by using satisfiability we construct a timetable which satisfies all constraints, but not an optimal timetable, whatever that may be. So maybe, the timetable designer wants to adjust the timetable, not because it's impermissible, but only to improve it.

Optimality is a difficult concept in timetabling. It will be clear that two idle periods for a teacher is better than three idle periods, but what if we can solve an idle period of a class by creating one for a teacher? It all depends on the situation, and that's why it's so hard to give a formal description of optimality at timetabling. Other methods use 'penalty points' to punish idle periods or other unwanted features of a timetable. Several timetables are constructed and the one with the least amount of penalty points is considered to be the optimal timetable. This of course depends on the number of penalty points given to an unwanted feature, and this amount is given by the timetable designer. This is a artificial way to achieve an optimal timetable, and won't always create a timetable which a timetable designer would consider to be optimal. We won't try to achieve the optimal timetable, because we won't be able to state what constraints should be satisfied to achieve an optimal timetable, we will only try to achieve a satisfiable timetable. By the option of adjustments we intend to give the timetable designer a useful tool to adjust the timetable to satisfy his of her desires.

There are four kinds of adjustments which we will consider. The first two consider the period assignment. The third considers the room assignment. The fourth kind considers daily adjustments, which are adjustments needed because of the absence of classes or teachers, and which will only last for a day. The first considers the adjustments of period assignment by hand. These will give the timetable designer the possibility to assign a lesson to another period. This mainly comes down to setting the variable, referring to the lesson at the old period to false and the variable, referring to the lesson at the new period to true. It is possible that by these adjustments conflicts occur, so we might need some way to verify the adjustments. On the other hand, we're able to make adjustments which conflict with our constraints, but which are admissible to us. For instance, let's say we can solve three idle periods for a teacher, just by assigning the lesson to another day. But at that day, a lesson of that teacher already occurs. We now have to make the decision if we want two lessons of the teacher on the same day, and conflict with the educational constraints, or three idle periods for the teacher.

The second adjustments are the adjustments of period assignment by the solver. If the timetable still contains features we don't appreciate, but we cannot solve them by hand, we should be able to use the solver to try and find another timetable. If we add some clauses, we might get the adjustments we want. If, for instance, we don't want a lesson to be assigned to a particular period, we set this variable to false. When the solver creates a new timetable, this lesson will not be assigned to the period and we will hopefully achieve a better timetable. It is be possible that we want to fix a part of the timetable we do like, so it should also be possible to fix lessons.

When these adjustments are made, one of the possible problems is that rooms might conflict. We therefore suppose that any adjustment to the period assignment is made before the rooms are assigned. In this way, we won't have to worry about rooms assigned to several lessons at the same time.

The third kind of adjustments are about changing the room assignment. We can imagine this also might need some changes. Changing the room assignment comes down to changing the room allocationmatrix, which can easily be achieved.

Even if we've constructed a timetable that pleases us, the fourth kind of adjustments will be needed. When some teachers are ill or classes are on a trip, adjustments to the timetable of a single day might be desirable, only to be used a single day. We'll refer to these adjustments as *daily adjustments*. We'll now look at the timetable differently. The timetable we've constructed, by the adjustments previously mentioned to a timetable constructed by the solver, is our basic timetable. Because of the absence of teachers or classes, daily adjustments will have to be made to achieve an acceptable timetable for that day. During the year I've been designing timetables, only a few times no daily adjustments were needed. This shows how important this side of timetabling is to a timetable designer.

6.2 Adjustments to period assignment by hand

When a timetable designer looks at a timetable constructed by a solver, he will probably see some trivial, easy improvements. This will mostly come down to assigning a lesson to another period or switching periods of two lessons, which is actually assigning a lesson to another period twice. Let's say we want to change the period assignment of the j^{th} lesson of class k by subject–teacher i from the current period t_1 on day d_1 to period t_2 on day d_2. To

achieve this, we have to change the assignment of these variables in the next way to adjust the timetable by hand:

$$\begin{array}{llll} \text{change} & x_{k,i,j,d_1,t_1} & \text{into} & \neg\, x_{k,i,j,d_1,t_1} \\ \text{change} & \neg\, x_{k,i,j,d_2,t_2} & \text{into} & x_{k,i,j,d_2,t_2} \end{array}$$

By changing the assignment of the variables, we can obviously make mistakes. We could for instance assign a lesson of a teacher to a period that's already occupied. We want some way to verify the changes we intend to make.

On the other hand, we might want to make changes which on purpose conflict with the constraints we've stated. For instance, if we can move a lesson, by which idle periods for the teacher are solved, but which will end up with two lessons at the same day, we might consider this option, even though it might conflict with the educational constraints. If we decide we want to make these changes, this should be possible. If the changes are verified, we want these changes to pass the verification. On the other hand, there are also constraints which cannot be violated, like the teacher constraints. We now see that the verification of the changes made by hand should only verify the constraints that can not be violated, and not the constraints that should not be violated. We only verify on the real requirements, not on the desires stated as requirement.

So how can we verify a change only to meet the requirements? It's straightforward to state the constraints that refer to requirements together with all the assigned variables, of which some are adjusted by hand. If the solver returns satisfiable, we didn't create any conflicts. It's not wise to verify several changes at once, because, if unsatisfiable, the solver does not return which variables create a problem, so we won't be able to solve the conflicts easily. Another way is to use a program by Fahiem Bacchus, called `modelcheck`[1]. This program determines whether or not the adjusted assignment satisfies the formula.

6.3 Adjustments to period assignment by the solver

If we're still not satisfied with the timetable, but we don't see any simple changes to solve unwanted features by hand, we can try to ask the solver to construct another timetable with some extra clauses. If a teacher for instance has a lesson on the first period on Monday followed by three idle periods, we can block the period the lesson is assigned to for that lesson. The lesson will be assigned to another period, which will, hopefully, solve some of the idle periods. It's of course possible that to assign the lesson to another period, another lesson will to be assigned to the first period of Monday. To prevent this, we can also block the period for all lessons, so the first period of Monday will be a free period to the teacher. To ensure no idle periods on Monday morning, we can also block all periods on Monday morning, so the teacher is granted a morning off. These are possibilities to force the solver to find a better timetable than has been found. The different ways to block periods can be executed by stating the next negation of variables, together with the constraints. If we let the solver take another look at the problem, it will produce another assignment of the variables and thereby another timetable. Let's say that we don't want the lesson j_1 of class k_1 by subject–teacher i_1 to be assigned to day d_1 on period t_1.

[1]This program can be downloaded from http://www.cs.toronto.edu/~fbacchus/2clseq.html

Block the period for this single lesson:
add $\neg\, x_{k_1,i_1,j_1,d_1,t_1}$ to the constraints.
If subject–teacher teaches more than one lesson to the same class, it is possible that after stating these clauses, the j_2^{th} lesson is assigned to period t_1 on day d_1. To prevent all lessons of the subject–teacher to be assigned to period t_1 on day d_1, we'll have to state the next clauses:

$$\bigwedge_{j|A_{k_1}(i_1,j)=1} \neg\, x_{k_1,i_1,j,d_1,t_1}$$

Block the period for all lessons of this teacher:
add all next literal constraints to the constraints:

$$\bigwedge_{k} \bigwedge_{i|SameTeacher(i_1,i)=1} \bigwedge_{j|A_k(i,j)=1} \neg\, x_{k,i,j,d_1,t_1}$$

Block all periods between t_2 and t_3 for all lessons of this teacher:
add all next literal constraints to the constraints:

$$\bigwedge_{k} \bigwedge_{i|SameTeacher(i_1,i)=1} \bigwedge_{j|A_k(i,j)=1} \bigwedge_{t_2\leq t\leq t_3|TeacherAvailable(i_1,d_1,t)=1} \neg\, x_{k,i,j,d_1,t}$$

One can imagine the same situation for a class. The same types of constraints have to be stated to get a similar effect on the timetables of classes.

If we block several periods, the solver produces a total new timetable if satisfiable. But it's very likely that a part of the original timetable did please us, so we want to keep it that way. Besides, the changes we did by hand could also all disappear. So we want to fix several lessons, that will not be changed by the solver. This is very straightforward, if we want to fix the j_1^{th} lesson for class k_1 by subject–teacher i_1 on period t_1 of day d_1, we only have to state the propagation of the variable referring to it:

Fix this single lesson:
add x_{k_1,i_1,j_1,d_1,t_1} to the constraints

It might be useful to create functions which fix whole days for classes and teachers. When we fix a whole day for a teacher, we should of course fix all single lessons of that day. But we should also state that all free periods have to remain free, otherwise lessons could be added to that day. This means we have to block the free periods for all lessons.

6.4 Adjustments to room assignment

Adjustments to the assignment of rooms can be done by adjusting the room allocationmatrix, which was defined in section 5.3.1 by

$$R(p,r) = \begin{cases} i, & \text{if room r has been assigned to subject–teacher } i \text{ on period p;} \\ 0, & \text{if room r has not been assigned.} \end{cases}$$

Most changes will be regarding the change of a room because a teacher needs or wants another room, for instance because a television is needed. By the way the room allocationmatrix is constructed, these kind of changes can easily be made. The period on which the room needs

to be changed is known, so we know which row of $R(p, r)$ we have to consider. To take the lesson out of room r_1 we set the corresponding element of $R(p, r_1)$ to zero and to assign the lesson to room r_2 we set the corresponding element of $R(p, r_2)$ to the number of the subject–teacher. If we first check if the room was empty, just by checking if the element equaled zero, we're sure not to rise any conflicts. The only problem might be that the room is to small for the number of students, but this could hardly be a problem, because the teacher himself asks for the room. If we're not sure the teacher is thinking of the room capacity constraints while asking for another room, we can also verify this very easily.

6.5 Daily adjustments

We'll now look at daily adjustments, i.e. adjustments which have to be made because of the absence of classes or teachers and which will only last for a day. Let's say we have to make daily adjustments for day d_1. The most straightforward way to do this is to fix all days except for day d_1 and try to achieve an acceptable timetable for day d_1. But we cannot just prevent the assignment of the lessons of the absent classes and teachers because of the completeness constraints, which make sure that every lesson in the curriculum actually occurs. To avoid unsatisfiability we could assign the lessons to other days, if possible, or adjust the curriculum. But this will not be the way we handle daily adjustments, we'll take another road.

When we think of what is needed, it is a timetable for day d_1, which is a single day. We know which lessons have to occur, because we know which lessons are assigned to day d_1. We can now create a new curriculum for day d_1. If we remove the lessons of the absent classes and teachers, we'll end up with assigning lessons to a single day, which is a problem we know how to tackle.

Similar to section 5.3, we first create the matrix Assign, which contains rows with the indices k, i, d and t of all assigned lessons. Because of every assigned lesson all needed indices are known, we can easily remove all lessons assigned to the other days, just by comparing the third index, all lessons of the absent classes and of the absent teachers, by comparing the first and the second index.

At the Rotterdamsch Lyceum, the timetables of the senior department are not adjusted. The argument is that if a lesson doesn't take place, the students should study independently during that period. This is also what the 'studiehuis' is about. Therefore, we also won't adjust the timetables of the senior department. By the way, if we want to do this properly, we should construct a timetable for the senior department as well, which is not easily constructed. So the period assignment of the senior department remains the same, just like at period assignment of the junior department.

The construction of the curriculum out of the remaining lessons in Assign is straightforward. We initiate a curriculum as an all-zero matrix of size K by D. For every lesson of Assign, we add one lesson to the element (k,i) of the curriculum. When we construct an allocationmatrix just like in section 2.2 and we state the next constraints, which are explained in section 2.4 the solver can construct a timetable:

Teacher constraints: every teacher can have at most one lesson at the same time.

Student constraints: every student can have at most one lesson at the same time.

Teacher availability constraints: a lesson can only be taught at a time the teacher is available.

Double period constraints: two lessons of particular subjects, like drawing or physical education, will be assigned to two consecutive periods.

Room compatibility constraints: only rooms of a suitable type are assigned to each lesson.

Students idle period constraints: students have no idle periods.

Completeness constraints: every existing lesson does actually occur.

Negation constraints: every non existing lesson does not occur.

Uniqueness constraints: every lesson is scheduled only once.

The only constraints missing are the educational constraints, which make sure that lessons of the same subject are distributed over the week as much as possible. Obviously, these are not useful because we only consider a single day. Because less lessons have to be assigned than in the original situation, we can try to state better 'optimizing' constraints. We adjusted the students idle period constraints in a way, that no idle periods occur.

Handling daily adjustments in this way might give rise to some disadvantages. We'll only discuss them, but won't give a solution to them. The solution is mostly straightforward and we feel that subjective preferences of a timetable designer should be handled in an individual way.

It is well possible that every week a totally new timetable for the same day is constructed. This can be confusing to both students and teachers. If daily adjustments are needed, one would like a timetable which is almost like the basic timetable, only adjusted to remove the idle periods for students, not a totally new timetable. This could be tackled by fixing some of the lessons. One can think of fixing all lessons of classes which aren't taught by the absent teachers or fixing a large part of the the remaining lessons.

Another problem is that students and teachers get used to the basic timetable. When the first three periods of Monday are free to a class, problems will rise if a lesson has been assigned to the first period by daily adjustments. It might not be that smart to adjust the timetable in this way, so some features of the basic timetable should be taken into account. Daily adjustments can also contain adjustments to the room assignment. These can be handled in the way mentioned in section 6.4.

Several extensions of the daily adjustments can be imagined. At the Rotterdamsch Lyceum, for a whole week the first classes go on a school outing to get to know each other, supervised by some of their teachers. The needed adjustments can be seen as *multiple daily adjustments* or in this case, weekly adjustments. These adjustments can be tackled in the same way as daily adjustments, only for multiple days.

Because daily adjustments or multiple daily adjustments are an important aspect of school timetabling, we've tested several cases for the Rotterdamsch Lyceum to get an impression of the performance on daily adjustments. Results will be shown in chapter 7, together with the performance on constructing a timetable for the Rotterdamsch Lyceum and results on adjustments by the solver, as in section 6.3.

Chapter 7

Results

Now we know how to construct school timetables and handle adjustments by satisfiability, we take a look at the performance of solvers. The first part of this chapter will cover the construction of the school timetable for the Rotterdamsch Lyceum. We recall that the construction of the school timetable consists of four parts, i.e. student sectioning, period assignment of the senior department, period assignment of the junior department and room assignment. Of the first three parts the performance of the solvers will be shown. We didn't use solvers at room assignment, so no results will be shown for that part. The second part of this chapter will consider adjustments to the timetable by the solver, when we're not satisfied with the initial timetable provided by the solver. The last part of this chapter shows results of various types of daily and weekly adjustments. To state the constraints we've used the program MATLAB 6.5. All runtimes are in seconds and rounded, and the runtimes refer to the time the solver needs to solve the instance. It takes the solver longer to actually give us a result, because it needs to read the input-file and write the output-file too.

7.1 Construction timetable

To test the performance on constructing timetables for the Rotterdamsch Lyceum Linux Knoppix version 3.1 operating system was used, and three solvers with default settings were tested: `Jerusat 1.2.b for Linux`, `March` and `zChaff version 2003.12.04`. We start off using a computer with 512 MB of memory, x86 Family 6 Model 8 Stepping 1 Genuine Intel $\sim$ 601 Mhz processor.

The first part of constructing timetables consists of student sectioning. For every year of the senior department the constraints were stated, which took a few minutes for all six years together. In table 7.1 the number of variables and stated clauses per year are shown as are the runtimes of the solvers to solve the instances. It will be clear that the student sectioning part is easily solved by each solver.

Table 7.1: Student sectioning

Year	vwo 4	vwo 5	vwo 6	havo 4	havo 5	mavo 4
Variables	300	450	896	3888	2860	2160
Clauses	365	529	992	4178	3196	2875
`Jerusat`	0	0	0	0	0	0
`March`	0	0	0	0	0	0
`zChaff`	0	0	0	0	0	0

After student sectioning we proceed with period assignment of the senior department. By the previous runs each solver produced a different type of student sectioning. Now each solver has to use each type of student sectioning to assign periods to the senior department. So each solver has to run three times. Stating the clauses for the period assignment of the senior department took about 10 minutes. The next quantities appeared in the senior department:

K= 50 : Total number of studentgroups in the senior department, which was divided in the next way :
- vwo 4 : 5 studentgroups
- vwo 5 : 6 studentgroups
- vwo 6 : 7 studentgroups
- havo 4 : 9 studentgroups
- havo 5 : 11 studentgroups
- mavo 4 : 12 studentgroups

D= 128 : Total number of subject–teachers in the senior department, who represent 33 real teachers

U= 4 : Maximum number of lessons of a subject

S = 5 : Number of days per week

T = 8 : Number of periods per day

This comes down to 50 x 128 x 4 x 5 x 8 = 1,024,000 variables.

At this point it turned out that more than 512 Mb of memory is required for solving these formulas. `March` was quickly aborted and the message "killed" was returned. `Jerusat` and `zChaff` couldn't decide on satisfiability or unsatisfiability within 8 hours. After 8 hours we decided to cancel the solvers. But we didn't want to give up so easily, so we turned to a pre-processor called `hypre` [Bac 03]. Pre-processors simplify the formula, which can be quite a remarkable simplification, which will hopefully lead to a formula which can be handled with 512 Mb memory in the first place, and will speed up the solving too. So the three formulas were pre-processed by `hypre`. Unfortunately, `hypre` too needed more memory than 512 Mb. We decided on using `hypre` at Linux Fedora Core 1 on a different computer, namely Intel Pentium 4 CPU 3,06 GHz 1,00 GB of RAM. More often `hypre` will be used, and each time this computer is used to run the pre-processor. All the other results, by the solvers and by MATLAB, are performed on the computer we started off with, with 512 MB of memory. In table 7.2 the results are shown as are the runtimes of `hypre`.

Table 7.2: Pre-processing period assignment senior department for $T = 8$ by `hypre`

Student sectioning by	`Jerusat`	`March`	`zChaff`
Initial variables	1024000	1024000	1024000
Initial clauses	3289225	3294201	3295049
Resulting variables	8966	8966	8966
Resulting clauses	396582	398459	395694
`hypre`	10	10	10

We can see for instance that at the formula of the student sectioning by `Jerusat` the number of variables has been reduced from 1,024,000 to 8,966 and the number of clauses has been reduced from 3,289,225 to 396,582. This took only 10 seconds. Now the formulas are simplified, we again turn to the solvers. The results are shown in table 7.3.

Table 7.3: Period assignment senior department for $T = 8$

Student sectioning by	`Jerusat`	`March`	`zChaff`
Variables	8966	8966	8966
Clauses	396582	398459	395694
`Jerusat`	> 8 hours	> 8 hours	> 8 hours
`March`	> 8 hours	> 8 hours	> 8 hours
`zChaff`	> 8 hours	> 8 hours	> 8 hours

Even though `March` wasn't aborted this time, the solvers couldn't decide on satisfiability or unsatisfiability within 8 hours again. It would be disappointing if we couldn't construct a timetable, so we had to find another way to get the solvers to construct a timetable. We decided to allow more periods per day, which is actually the case at the Rotterdamsch Lyceum. This leads to 9 periods available per day, so $T = 9$.

We stated the clauses for period assignment in the senior department again, but now for $T = 9$. This took MATLAB about 11 minutes. First, we use `hypre` to simplify the formulas. The results of this simplification are shown in table 7.4. Again, a severe simplification can be noticed.

Table 7.4: Pre-processing period assignment senior department for $T = 9$ by `hypre`

Student sectioning by	`Jerusat`	`March`	`zChaff`
Initial variables	1152000	1152000	1152000
Initial clauses	3892401	3897999	3898953
Resulting variables	10254	10254	10254
Resulting clauses	494800	497322	493764
`hypre`	13	12	12

When the files are pre-processed, we turn to the solvers and hope they find a satisfying assignment within reasonable time or decide on unsatisfiability of the formula. In table 7.5 the results are shown. Now `Jerusat` could each time find an satisfying assignment within minutes, `March` could only find an satisfying assignment once within 8 hours and `zChaff` performs very divergent.

Table 7.5: Period assignment senior department for $T = 9$

Student sectioning by	`Jerusat`	`March`	`zChaff`
Variables	10254	10254	10254
Clauses	494800	497322	493764
`Jerusat`	42	80	56
`March`	671	>8 hours	>8 hours
`zChaff`	4	>8 hours	7040

After period assignment of the senior department we proceed to the last part, the period assignment of the junior department. Because six times a period assignment for the senior department was found, we could now test the solvers performing on the period assignment of the junior department on each of the six assignments of the senior department. Stating the clauses for the period assignment of the junior department took about 5 minutes. The next quantities appeared at assigning periods to the junior department.

K= 19	:	Total number of years in the senior department (=6) and classes in the junior department (=13)
D= 188	:	Total number of subject–teachers, who represent 46 real teachers
U= 5	:	Maximum number of lessons of a subject
S = 5	:	Number of days per week
T = 9	:	Number of periods per day

This comes down to 19 x 188 x 5 x 5 x 9 = 803,700 variables

First, `hypre` was used to simplify the formula. Results are shown in table 7.6.

Table 7.6: Pre-processing period assignment junior department by hypre

Student sectioning by	`Jerusat`	`March`	`zChaff`	`Jerusat`	`Jerusat`	`zChaff`
Senior department by	`Jerusat`	`Jerusat`	`Jerusat`	`March`	`zChaff`	`zChaff`
Initial variables	803700	803700	803700	803700	803700	803700
Initial clauses	1806454	1806454	1806454	1806454	1806454	1806454
Resulting variables	10566	10561	10543	10590	10605	10544
Resulting clauses	449403	450595	448502	451674	453496	449142
hypre	8	8	8	8	8	8

Now the formulas to assign periods to the junior department are simplified, we turn to the solvers to find an satisfying assignment. The results are shown in table 7.7.

Table 7.7: Period assignment junior department

Student sectioning by	Jerusat	March	zChaff	Jerusat	Jerusat	zChaff
Senior department by	Jerusat	Jerusat	Jerusat	March	zChaff	zChaff
Variables	10566	10561	10543	10590	10605	10544
Clauses	449403	450595	448502	451674	453496	449142
Jerusat	47	31	51	40	45	53
March	>8 hours	>8 hours	>8 hours	>8 hours	>8 hours	>8 hours
zChaff	>8 hours	>8 hours	97	27	>8 hours	>8 hours

We see that each time Jerusat found a satisfying assignment within a minute, whereas it took March each time over 8 hours to find it. Twice it took zChaff only several minutes to find a solution, but the other times it took over 8 hours.

So what can be concluded from these results? The first thing is that student sectioning is tackled easily by each solver, probably because after dividing the problem into several years, the size is relatively small. The period assignment of the senior department has only been tackled easily by Jerusat, the only tested solver that found an satisfying assignment each time within minutes. The other solvers couldn't handle this part as well as Jerusat. March succeeded once, needing about 11 minutes, whereas zChaff once needed only seconds, once needed almost two hours and once needed more than 8 hours. More instances need to be tested, but it seems like Jerusat performs the best and the most consequent of the three tested solvers. The period assignment of the junior department was again only handled easily by Jerusat, needing less than a minute each time. March needed more than 8 hours each time and zChaff needed more than 8 hours at 4 instances. The other instances were solved within minutes. On pre–processing, we can point out that each time hypre only needed seconds to simplify a formula and its performance was very stable.

Finally, we can say that it is possible to construct a timetable using satisfiability. For the Rotterdamsch Lyceum, if we use Jerusat and hypre, the total runtime can be less than 2 minutes. March is not suitable for solving these instances and the performance of zChaff differs a lot, but also doesn't seem to suit these problems.

7.1.1 Influence of pre-processing

Let's take a look at the influence of pre-processing. In table 7.4 is shown that the pre-processor hypre reduced the number of variables of the formula, representing the student sectioning by Jerusat, to $\frac{10254}{1152000} = 1\%$ of the original number of variables, whereas the number of clauses was reduced to $\frac{494800}{3892401} = 13\%$ of the original number of clauses. Of the other formulas the reduction is equal. Even though this reduction is quite remarkable, it is of no use if the runtime of the solvers isn't reduced too and, in our case, the needed memory isn't reduced. We've seen that indeed the needed memory has reduced, which gave the solvers the opportunity to actually find a satisfying assignment, without running out of memory.

But what is the influence on the runtime? To get insight into this aspect, we've also used the not yet pre-processed formulas to be solved by the solvers to assign periods to the senior department. So, we went immediately from student sectioning to period assignment of the

senior department without pre-processing. The result is shown in table 7.8.

Table 7.8: Period assignment senior department for $T = 9$ without pre-processing

Student sectioning by	`Jerusat`	`March`	`zChaff`
Number of variables	1152000	1152000	1152000
Number of clauses	3892401	3897999	3898953
`Jerusat`	killed	killed	killed
`March`	killed	killed	killed
`zChaff`	91	247	89

If we compare these results with table 7.5, we see a total different picture. Indeed, `Jerusat` and `March` are aborted because of too little memory, but the runtimes of `zChaff` are not showing an improvement by pre-processing. Whereas the runtime to solve the formula, representing the student sectioning by `Jerusat`, has been reduced by pre-processing, the other runtimes are by far quicker without pre-processing. Pre-processors create a formula which is equivalent to the original formula, but the structure will most likely be different. This won't always have a positive effect on the runtimes, as this example shows.

7.2 Adjustments by solver

Once we've constructed a timetable, we might want to adjust it using the solver, as we've seen in section 6.3. To test the adjustments by the solver only `Jerusat 1.2.b for Windows` at Microsoft Windows 2000 Professional operating system was used. The used computer was the computer with 512 MB of memory, x86 Family 6 Model 8 Stepping 1 Genuine Intel $\sim$ 601 Mhz processor. We've tested this for several instances to see how the solver would react if we, for instance, forced some lessons to be fixed at certain periods along with blocking some periods or requiring rearrangement of some lessons.

Change one day of a class and fix no lessons
We first took a look at the situation when we're not satisfied with a single day of a single class. The adjustments we examined only consider the junior department, so we randomly chose a class of the junior department and we randomly chose a day, which the solver had to change. For now we didn't fix days yet. This could imply a rearrangement of lessons at other days too, besides the one we've chosen. The results are shown in table 7.9. All instances were solved within reasonable time.

Change one day and fix three days of a class
It might become more interesting when we add the desire to fix several days of the same class, of which a day has to be rearranged. We chose three days to be fixed at random. This didn't seem to affect the runtimes a lot, as is shown in table 7.10.

Block one period of a class and fix no lessons
Let us now only try to block one period of a class, so no forced rearrangement or fixing days, and take a look at the performance of the solver. It should be pointed out that we can't

Table 7.9: Change one day of a class and fix no lessons

	Satisfiable	Unsatisfiable
Percentage	100 %	0 %
Average	144	
Standard deviation	66	
Total runs	20	

Table 7.10: Change one day and fix three days of a class

	Satisfiable	Unsatisfiable
Percentage	100 %	0 %
Average	146	
Standard deviation	52	
Total runs	20	

just pick any period to block, because the students idle constraints require the second period until the sixth period to be occupied and the ninth period in the junior department is blocked anyway, so we can only pick the first, the seventh or the eighth period. Whatever combination of class and period we picked, the solver coped with it easily, as is shown in table 7.11.

Table 7.11: Block one period of a class and fix no lessons

	Satisfiable	Unsatisfiable
Percentage	100 %	0 %
Average	175	
Standard deviation	57	
Total runs	20	

Block two periods and fix three days of a class
We now fix three days of a class, and block two periods of the remaining days for the same class. If we fix three days, we only leave two days to be changed. Of course we blocked periods that were occupied the first time, otherwise the same timetable could be returned. If we block two periods of the remaining days, only 14 available periods remain. Because at least the second until the sixth period were occupied, we have at least 10 lessons within those 14 periods. Together with the two lessons at the blocked periods, the solver needs to assign at least 12 lessons to 14 periods. This shows the impact of our restrictions. Table 7.12 shows that a part turned out to be unsatisfiable, but at both satisfiable and unsatisfiable cases, a result is given within minutes.

Concluding, we can say that at several instances with additional requirements the solvers returned a result pretty fast, even when it turned out to be unsatisfiable.

Table 7.12: Block two periods and fix three days of a class

	Satisfiable	Unsatisfiable
Percentage	70 %	30 %
Average	149	9
Standard deviation	69	7
Total runs	20	

7.3 Daily and weekly adjustments

To test the daily and weekly adjustments again only `Jerusat 1.2.b for Windows` was used on the computer with 512 MB of memory, x86 Family 6 Model 8 Stepping 1 Genuine Intel $\sim$ 601 Mhz processor. The first part of this section covers the daily adjustments, the second part considers the weekly adjustments. At both cases the following situations were simulated: the absence of only one teacher, the absence of five teachers and the absence of five teachers and five classes. Because these kind of adjustments are only made for the junior department, we've tested only the interesting cases where the absent teacher taught the junior department and the absent classes were classes of the junior department.

7.3.1 Daily adjustments

We chose a teacher at random to be absent and a day for the teacher to be absent at. Of course only combinations we allowed, that the teacher is indeed teaching classes, otherwise no adjustments need to be made.

One teacher and no classes absent
If only one teacher is absent, it is easier to construct a timetable than when the teacher is present. Therefore we can try to construct a better timetable than we had. We try to construct a timetable where no idle periods are allowed at all from the 2^{nd} period on. In table 7.13 the results are shown.

Table 7.13: One teacher and no classes absent without idle periods

	Satisfiable	Unsatisfiable
Percentage	69 %	31 %
Average	5	1
Standard deviation	1	1
Total runs	32	

We can see that the results are known within seconds. If a case turned out to be unsatisfiable, we have to decide what to do. If a teacher teaches each class only one lesson per day, the absence of this teacher can only lead to one idle period. Because we cannot achieve no idle periods for all classes, we can take the idle periods for granted and don't try to improve the timetable of that day. A second option is to try to achieve no idle periods for a part of the classes of the junior department, by forcing no idle periods for specific classes. We tried

to achieve this in a different way. We blocked the last periods of the day of each class, only to leave enough periods to create one idle period. So now we can get at most one idle period for at most all classes. The results are shown in table 7.14.

Table 7.14: One teacher and no classes absent with one idle period allowed

	Satisfiable	Unsatisfiable
Percentage	88 %	12 %
Average	5	1
Standard deviation	1	1
Total runs	32	

A better result is achieved with regards to the ratio of satisfiable and unsatisfiable results. Unfortunately, a small part remains unsatisfiable. We feel that this will always be the case, so we'll leave the case of one absent teacher and proceed to the case of five absent teachers. At any case, the answer is clear within seconds.

Five teachers and no classes absent
When five teachers are absent, many idle periods can occur. If we require no idle periods, it can turn out to be unsatisfiable easily, also because the remaining teachers can be occupied several periods to teach the senior department. The results in table 7.15 show that this indeed occurred. A large part turned out to be unsatisfiable within seconds, if we required no idle periods from the 2^{nd} period on.

Table 7.15: Five teachers and no classes absent without idle periods

	Satisfiable	Unsatisfiable
Percentage	19 %	81 %
Average	2	1
Standard deviation	0	1
Total runs	32	

So we tried the same trick as at the previous case, we allowed one idle period for each class, by blocking the last periods of the day. This didn't turn out to be an improvement, as table 7.16 shows.

Table 7.16: Five teachers and no classes absent with one idle period allowed

	Satisfiable	Unsatisfiable
Percentage	19 %	81 %
Average	2	0
Standard deviation	0	0
Total runs	32	

The next step is to allow at most two idle periods, by blocking as many periods at the end of the day to leave enough space to create at most two idle periods. Although now the main part turned out to be satisfiable, the ratio of satisfiable and unsatisfiable instances is still unsatisfying to us, as is shown in table 7.17. But we feel allowing three idle periods is just too much. If we cannot provide a solution by the solver, other options can be considered. Lessons can be discarded or a substitute teacher can take care of a single lesson. These options can also be considered, if it is too hard to achieve a timetable. After all, it is of course possible that we ask to much of the solvers. The important part to us is that the solver gives us an answer within seconds.

Table 7.17: Five teachers and no classes absent with two idle periods allowed

	Satisfiable	Unsatisfiable
Percentage	69 %	31 %
Average	2	0
Standard deviation	0	0
Total runs	32	

Five teachers and five classes absent
Finally, we look at the situation where five teachers and five classes are absent, which can be imagined to occur on a school–trip. At these school–trips, mostly 'similar' classes are absent, like all classes of the first grade or all classes of the junior department of the vwo. At choosing the absent classes we also took this into account. So, for instance, several times all classes of the first grade are absent. Most of the times these classes are supervised on their school–trip by their own teachers, so we also took teachers who teach the absent classes. It turned out the situations can be handled quite easily, as is shown in table 7.18.

Table 7.18: Five teachers and five classes absent without idle periods

	Satisfiable	Unsatisfiable
Percentage	84 %	16 %
Average	2	0
Standard deviation	0	0
Total runs	32	

7.3.2 Weekly adjustments

At the weekly adjustments, the teachers and the classes are not absent for a day, but for a week. Difficulties rise at the same situation as at the daily adjustments, namely at the absence of five teachers and no classes. But because constructing a timetable for a week can take a while, sometimes no result was given within an hour. We decided to stop the solver if this was the case.

One teacher and no classes absent
If one teacher is absent, we want to create a timetable with no idle periods, just like at daily adjustments. But now we cannot force this, because we have to consider several days. So we first try to create a timetable just like we used to create a timetable. The results in table 7.19 show that in most cases this can be achieved quickly and table 7.20 shows that if we allow the 6^{th} period to be a free period, and possibly an idle period, all cases can be solved within minutes.

Table 7.19: One teacher and no classes absent without idle periods from 2^{nd} until 6^{th} period

	Satisfiable	Unsatisfiable	Runtime > 1 hour
Percentage	85 %		15 %
Average	160		
Standard deviation	53		
Total runs	26		

Table 7.20: One teacher and no classes absent without idle periods from 2^{nd} until 5^{th} period

	Satisfiable	Unsatisfiable
Percentage	100 %	0 %
Average	150	
Standard deviation	57	
Total runs	26	

Five teachers and no classes absent
As we've already seen at daily adjustments, this will be the hardest case. We start off with a quite naive approach, namely to require no idle periods for all periods between the 2^{nd} until 5^{th} period. When five teachers are absent, it is possible that a class is left with less than 20 lessons per week, so this could easily turn out to be unsatisfiable. Even so, 41 % of the cases turned out to be satisfiable. The other 59 % remained unsolved. as is shown in table 7.21.

Table 7.21: Five teachers and no classes absent without idle periods from 2^{nd} until 5^{th} period

	Satisfiable	Unsatisfiable	Runtime > 1 hour
Percentage	41 %	0 %	59 %
Average	78		
Standard deviation	32		
Total runs	32		

We tried to achieve a better result by allowing one idle period per day. This is turned out to be worse than the previous case in one way and better in another. The worse part is that less instances turned out to be satisfiable. The better part is that a larger part actually was

determined to be satisfiable or unsatisfiable, but the runtimes raised tremendously. Especially the average of the unsatisfiable cases is high, i.e. over half an hour. Because of these high runtimes, we decided to let the solver run for at most two hours, just to give the solver enough time to decide on satisfiability or unsatisfiability. Table 7.22 shows the results.

Table 7.22: Five teachers and no classes absent with one idle period per day allowed

	Satisfiable	Unsatisfiable	Runtime > 2 hours
Percentage	22 %	34 %	44 %
Average	269	2136	
Standard deviation	166	1165	
Total runs	32		

If we allow two idle periods per day, we get a slightly better result. It still takes 50 % over an hour, but the rest is satisfiable, and the average runtime of the satisfiable cases has decreased quite a lot. Table 7.23 shows the results.

Table 7.23: Five teachers and no classes absent with two idle periods per day allowed

	Satisfiable	Unsatisfiable	Runtime > 1 hour
Percentage	50 %	0 %	50 %
Average	121		
Standard deviation	74		
Total runs	32		

Still a large part remained unsolved. We're trying to create a very tight timetable and this leads to problems for the solver. It might just be too hard to be solved anyway. We conclude that half of the cases were determined to be satisfiable within reasonable time, taking into account that we're constructing a whole timetable.

Five teachers and five classes absent

The last case considers the absence of five teachers and five classes. No difficulties rise at solving these instances, as table 7.24 shows.

Table 7.24: Five teachers and five classes absent without idle periods from 2^{nd} until 5^{th} period

	Satisfiable	Unsatisfiable
Percentage	100 %	0 %
Average	18	
Standard deviation	13	
Total runs	32	

At daily and weekly adjustments most instances were determined to be satisfiable or unsatisfiable within reasonable time. Only at weekly adjustments where five teachers were absent only 50 % could be solved. Of course other ways can be thought of to tackle those instances. We had rather strict requirements and could just ask to much to construct a timetable. It can even be questioned if that situation will ever occur. Anyway, it is interesting to see how the solvers react at the "border" of satisfiability and unsatisfiability on instances of this size. We see that with a little creativity the larger part can be handled within reasonable time.

Chapter 8

Expansion

In the previous chapter we saw that constructing a timetable for the Rotterdamsch Lyceum is possible within reasonable time, as is adjusting the timetable to several situations. We now want to look at larger schools. We didn't have any information of larger schools so we expanded the Rotterdamsch Lyceum ourselves to get a larger school. After we take a look at doubling the Rotterdamsch Lyceum, we end up tripling the Rotterdamsch Lyceum. We'll arrive at a school of about 1,500 students, which is a sizable school in the Netherlands.

8.1 Doubling the size of the school

We will now take a look at how we doubled the Rotterdamsch Lyceum. We start at the senior department. If we want to double the senior department, we need to add students so we need to know the students curricula of the students to be added. This is kind of a problem, because choosing students curricula at random could turn out to be lucky or disastrous. The period assignment of the senior department depends on the number of studentgroups per year. If we for instance give the added students the same curricula of the current students, the number of studentgroups will not increase, because the added students will be added to the existing studentgroups. Besides the increasing size of the studentgroups, this won't be of any real influence, because the constructed studentgroups were relatively small of size. So this could turn out to be not quite more difficult than the Rotterdamsch Lyceum. On the other hand, we could also assign curricula to the new students that all imply an extra studentgroup. This would be quite disastrous, because in section 3.3.3 we showed that the number of comparisons to state the lessongroup-size constraints could rise to $2^n - n - 2$, with n the number of studentgroups. So choosing realistic curricula for the new students would hopefully lead to neither of these two cases.

Fortunately, I could lay my hands on the students curricula of previous years. So in this way we can add students with realistic curricula. By doubling the number of students we didn't need to double the number of teachers. This was possible because of three reasons. The first reason is that the number of added students was smaller than number of students in the current years. So for instance, vwo 5 had 20 students and 13 students were added. This lead to at most $20 + 13 = 33$ students at the same lessongroup, which would just fit in the same lessongroup. At years at which lessongroups could exceed the maximum size most of the lessongroups, they didn't. If 14 students followed the subject `ak` and 13 are added, this won't lead to extra lessongroups. If no extra lessongroups were constructed no more lessons

have to be taught. The third reason is the next. When more lessons had to be taught, we can choose between adding teachers or assigning more lessons to the current teachers. The last option was possible so for now we choose this option. At the junior department we choose the other option.

So now the senior department is doubled, we tried to construct a timetable in the same way we did in the previous chapter. We start off with student sectioning. The next number of studentgroups appeared in the senior department:

vwo 4	:	6 studentgroups
vwo 5	:	6 studentgroups
vwo 6	:	11 studentgroups
havo 4	:	15 studentgroups
havo 5	:	12 studentgroups
mavo 4	:	14 studentgroups

The number of studentgroups per year has only slightly increased. This can be explained by two reasons. The first is that a smaller number of students is added, and the second is that a large part of the new students obviously could be added to the existing studentgroups. Results for student sectioning can be seen in table 8.1. Stating all these files takes about 18 minutes.

Table 8.1: Student sectioning after doubling

Year	vwo 4	vwo 5	vwo 6	havo 4	havo 5	mavo 4
Number of variables	1728	1122	1936	11340	8568	3080
Number of clauses	1969	1273	2106	13016	10785	4653
Jerusat	0	0	0	0	0	0
March	0	0	0	1	1	0
zChaff	0	0	0	0	0	0

Doubling the school in this way didn't seem to affect the runtimes to solve student sectioning. We proceed to period assignment of the senior department. We didn't need more teachers, but we did need more lessongroups for some subjects, so the maximum number of lessongroups G per teacher increased. This means that our total number of subject–teachers in the senior department D also increased. The next quantities appeared.

K= 64	:	Total number of studentgroups in the senior department, which was divided in the next way :
		vwo 4 : 6 studentgroups
		vwo 5 : 6 studentgroups
		vwo 6 : 11 studentgroups
		havo 4 : 15 studentgroups
		havo 5 : 12 studentgroups
		mavo 4 : 14 studentgroups
D= 178	:	Total number of subject–teachers in the senior department, who represent 33 real teachers

$U = 4$: Maximum number of lessons of a subject
$S = 5$: Number of days per week
$T = 9$: Number of periods per day
This comes down to 64 x 178 x 4 x 5 x 9 = 2,050,560 variables

We now immediately turn to $T = 9$, and first pre-process the files by `hypre`, of which the results are shown in table 8.2. Stating these files took about 15 minutes.

Table 8.2: Pre-processing period assignment senior department for $T = 9$ after doubling by `hypre`

Student sectioning by	`Jerusat`	`March`	`zChaff`
Initial variables	2050560	2050560	2050560
Initial clauses	5336974	5335856	5349659
Resulting variables	11503	11399	11575
Resulting clauses	574385	556844	581344
`hypre`	19	18	18

Now the files have been pre-processed, we can turn to the period assignment of the senior department.

Table 8.3: Period assignment senior department for $T = 9$ after doubling

Student sectioning by	`Jerusat`	`March`	`zChaff`
Variables	11503	11399	11575
Clauses	574385	556844	581344
`Jerusat`	237	$>$ 8 hours	162
`March`	$>$ 8 hours	$>$ 8 hours	$>$ 8 hours
`zChaff`	$>$ 8 hours	$>$ 8 hours	$>$ 8 hours

We see that now only `Jerusat` could find a satisfying assignment. After assigning periods to the senior department, we assign periods to the junior department. At this department we indeed created new teachers to teach the new classes, so the number of teacher increased too. These new teachers only teach the new classes and the new classes are only taught by the new teachers. Stating the clauses takes about 15 minutes. The next quantities appeared at assigning periods to the junior department.

$K = 32$: Total number of years in the senior department (=6)
and classes in the junior department (=26)
$D = 298$: Total number of subject–teachers,
who represent 80 real teachers
$U = 5$: Maximum number of lessons of a subject
$S = 5$: Number of days per week
$T = 9$: Number of periods per day
This comes down to 32 x 298 x 5 x 5 x 9 = 2,145,600 variables

Table 8.4: Pre-processing period assignment junior department after doubling by `hypre`

Student sectioning by	`Jerusat`	`zChaff`
Senior department by	`Jerusat`	`Jerusat`
Initial variables	2145600	2145600
Initial clauses	3872675	3872675
Resulting variables	24274	24274
Resulting clauses	972046	972046
`hypre`	18	18

But first we pre-process the files by `hypre`. The results are shown in table 8.4. After pre-processing we can assign periods to the junior department. Table 8.5 shows that it took `Jerusat` about 5 minutes, whereas the others could not find a satisfying assignment within 8 hours. All together we see that no extra problems arise at doubling the Rotterdamsch Lyceum. Together with `hypre` `Jerusat` performs pretty solid, whereas `March` couldn't cope with it too, just like at the Rotterdamsch Lyceum at normal size. At the normal size `zChaff` performed divergent, but now `zChaff` can't cope with the size. Because this size doesn't give rise to any serious problems, we proceed to tripling the size of the Rotterdamsch Lyceum in the next section.

Table 8.5: Period assignment junior department after doubling

Student sectioning by	`Jerusat`	`zChaff`
Senior department by	`Jerusat`	`Jerusat`
Variables	24021	24028
Clauses	957037	957382
`Jerusat`	302	301
`March`	$>$ 8 hours	$>$ 8 hours
`zChaff`	$>$ 8 hours	$>$ 8 hours

8.2 Tripling the size of the school

In the previous section we increased the size of the Rotterdamsch Lyceum to almost it's double size and it turned out that a timetable could still be constructed in reasonable time by `Jerusat`. Yet we're not arrived at a sizable school. So now we try and triple the size of the Rotterdamsch Lyceum. We'll end up with a school of about 1,500 students, which is a sizable school in the Netherlands. In the school-year 2002/'03 the average schoolsize in the Netherlands was 1,320 [CBS 04]. At this count a 'school' can consist of several establishments, and each establishment could have it's own timetable. But anyway, this shows that a school of 1,500 students is indeed sizable in the Netherlands.

We already saw that `March` is not suited for these kind of instances. Therefore we decide to leave `March` and proceed with the two other solvers. Just at the same time as we started to try to construct timetables for a school with triple size of the Rotterdamsch Lyceum, new

versions were released of `Jerusat` and `zChaff`. The new versions were used, hoping to be better suited for our type of instances than the old versions. The new version of `Jerusat` is called `Jerusat1.3` and will be abbreviated as `Jerusat3` and the new version of `zChaff` is `zChaff 2004.5.13` and will be abbreviated as `zChaff4`. Both solvers and pre-processor `hypre` were used at Linux Fedora Core 1 on an Intell Pentium 4 CPU 3.06 GHz 1,00 GB of RAM computer.

In the previous section we sometimes took the easy way out, like not hiring extra teachers for the senior department and adding a smaller number of students in the senior department than the current. We also added teachers to teach only in the junior department. If a school increases in size, this is not plausible. So now we truly triple the school. We make sure that in every senior year the number of students is indeed tripled and that new occurring lessons are to be taught by new teachers. This still doesn't mean we need three times the number of current teachers. For instance, in vwo 4 there are 20 students. If we triple them, we get 60 students, which leads to one extra lessongroup at some subjects, instead of two, because we split at 35 students. So for every subject we now need to count how many extra lessongroups will be needed.

Because no more old students curricula were available, we need to make up curricula for the new students. Once we've done this, we can proceed to the student sectioning. The first problems will arrive at this stage.

8.2.1 Forced student sectioning

When we truly triple the students per year in the senior department, the number of student-groups per year rises, as the next table shows:

vwo 4	:	12 studentgroups
vwo 5	:	14 studentgroups
vwo 6	:	16 studentgroups
havo 4	:	22 studentgroups
havo 5	:	13 studentgroups
mavo 4	:	33 studentgroups

The number of comparisons to state the lessongroup-size constraints for mavo 4 is so huge, that they could not be stated anymore by MATLAB. So we need a way to force a student sectioning ourselves, without needing to state clauses and therefore without the solvers. We'll follow the way we performed student sectioning on compulsory subjects in section 3.4. For the other years we could hold on to our method using the solvers, but it took over 1:20 hour to state all clauses of the remaining five years. We now explain how forced student sectioning can be achieved.

We'll illustrated the method by vwo 4 of the Rotterdamsch Lyceum, as we've seen in chapter 3. The start is equal to the student sectioning we've already seen. We construct studentgroups and subjectgroups in the similar way. We end up with the next distribution of subjects over the subjectgroup and the distribution of students over the studentgroups. Table 8.6 is equal to table 3.1 and indicates how many students of each studentgroup have chosen a particular subjectgroup.

Subjectgroup $Subg_1$: `gr`
Subjectgroup $Subg_2$: `la`
Subjectgroup $Subg_3$: `wb, na` and `sk`
Subjectgroup $Subg_4$: `sp`
Subjectgroup $Subg_5$: `kcv`
Subjectgroup $Subg_6$: `bi`
Subjectgroup $Subg_7$: `fa2, du2` and `ak`
Subjectgroup $Subg_8$: `ck1`
Subjectgroup $Subg_9$: `ck3`

Studentgroup $Studg_1$: F. el Arkoubi, R. Bouzidi, E. Kajmovic, A. Mohan, B. Petekci and V. de Veth
Studentgroup $Studg_2$: R. Groeneveld and H. Kelleci
Studentgroup $Studg_3$: F. Djorai, N. Asut, A. Kewal and R. Ramdat
Studentgroup $Studg_4$: N. Gamhiouen, D. Huisden and G. Timmermans
Studentgroup $Studg_5$: S. Ramdat, W. Nejal, W. Biharie, N. Girasing and C. Rouwers

Table 8.6: Acquired matrix after transformations

	$Subg_1$	$Subg_2$	$Subg_3$	$Subg_4$	$Subg_5$	$Subg_6$	$Subg_7$	$Subg_8$	$Subg_9$
$Studg_1$	0	0	0	6	0	0	6	6	6
$Studg_2$	0	0	2	2	0	2	0	2	0
$Studg_3$	0	0	0	0	1	3	4	4	4
$Studg_4$	3	3	0	0	3	2	3	0	3
$Studg_5$	3	4	5	0	4	5	0	2	3

Let's say the maximum groupsize equals 14. According to figure 3.1, this means we need two lessongroups for the compulsory subjects, which are `ne, fa1, du1, en, gs, wa, ec, ma, lo` and `lob`, and `ck3`, which is chosen by 16 students. The other subjects need only one lessongroup. The aim is a lessongroup-based curriculum, as is explained in 3.5. Table 8.7 gives an impression of what this should look like for vwo 4.

Now we have to choose an order to assign the studentgroups to the lessongroups of all chosen subjects. We choose the studentgroups in descending order with respect to size, starting with the one containing the most students. We assign each studentgroup to a lessongroup of the chosen subjects which has the least students. Let's take a look how this works out.

We first take the largest studentgroup, which is $Studg_1$ containing 6 students. No students have all ready chosen any subjects, so it doesn't matter to which lessongroups $Studg_1$ is assigned to. We just take the first of all, assign the right number of lessons and end up with table 8.8.

Next we choose the second large studentgroups, which is $Studg_5$. Of the two available lessongroups of the subjects `ne` and `fa1`, lessongroup 2 contains the least students, so at those

Table 8.7: Forced clustering: start

subject	ne		la	gr	fa1	
lessongroup	1	2	1	1	1	2
number of students	0	0	0	0	0	0
studentgroup x						
studentgroup y						

Table 8.8: Forced clustering: step 1

subject	ne		la	gr	fa1	
lessongroup	1	2	1	1	1	2
number of students	6	0	0	0	6	0
$Studg_1$	3				2	

subjects they're assigned to the second lessongroup. At the other subjects there is only one lessongroup, so no problems at picking one. We end up with table 8.9.

Table 8.9: Forced clustering: step 2

subject	ne		la	gr	fa1	
lessongroup	1	2	1	1	1	2
number of students	6	5	4	3	6	5
$Studg_1$	3				2	
$Studg_5$		3	4	2		2

We repeat these simple steps until we've assigned all studentgroups to lessongroups. In this way we still assign all the students of the same studentgroup to the same lessongroup and the difference in size between lessongroups remains small.

8.2.2 Timetabling continued

Now we know how to perform student sectioning when the number of studentgroups is too large, we apply this method to mavo 4. Student sectioning of the other years was still performed by the solver. We first stated the clauses for the other years, which took over 1:20 hour. Results on this are shown in table 8.10.

Table 8.10: Student sectioning after tripling

Year	vwo 4	vwo 5	vwo 6	havo 4	havo 5
Variables	5712	10080	6144	86240	39208
Clauses	6191	10568	7160	95950	43482
`Jerusat3`	0	0	0	0	0
`zChaff4`	0	0	0	0	0

Once student sectioning is performed, we turn to period assignment of the senior department. The next quantities appeared in the senior department.

K= 110 : Total number of studentgroups in the senior department, which was divided in the next way :
- vwo 4 : 12 studentgroups
- vwo 5 : 14 studentgroups
- vwo 6 : 16 studentgroups
- havo 4 : 22 studentgroups
- havo 5 : 13 studentgroups
- mavo 4 : 33 studentgroups

D= 404 : Total number of subject–teachers in the senior department, who represent 60 real teachers

U= 4 : Maximum number of lessons of a subject

S= 5 : Number of day per week

T= 9 : Number of periods per day

This comes down to 110 x 404 x 4 x 5 x 9 = 7,999,200 variables

At this stage serious problems have risen. We've made several attempts to achieve a period assignment for the senior department, but no attempt was successful. We'll illustrate our attempts.

Regular way Of course we first tried to achieve a timetable at the way we also used at our previous sizes. Over 14,6 million clauses were stated, but at pre-processing `hypre` was aborted after 50 minutes. We also tried to solve the not-pre-processed instance by the solvers, but `Jerusat3` was killed within 8 minutes and `zChaff4` was killed within 2 minutes.

Removing zero columns The first adjustments we tried was to remove the zero columns from the lessongroup-based curriculum. We've already seen that by our way of student sectioning for all subject–teachers the same amount of lessongroups are constructed, several needed, but also several unnecessary. These extra lessongroups result in zero columns in the lessongroup-based curriculum. This increases D, which is the number of columns of the lessongroup-based curriculum. We've removed these zero columns, which resulted in removing 181 columns, so D = 404 – 181 = 223. Yet `hypre` was aborted within 45 minutes, even though the number of variables was reduced to 4,415,400 and the number of clauses was about 11 million.

Partial timetabling So what to do next? We tried to construct a timetable for the vwo only, and hold on to this timetable at constructing a timetable for the rest of the senior department. A timetable for the vwo was constructed pretty easily. Initially we had 442,260 variables and almost 3 million clauses, but after pre–processing by `hypre` we had an instance with only 6,701 variables and almost 300 thousand clauses, which was solved by `Jerusat3` in about 43 seconds and by `zChaff4` within a second. Once we attained a timetable for the vwo, we tried several attempts to attain a timetable for the whole senior department.

State only true variables When all variables referring to the vwo are known, we can choose either to state all variables or state only the true variables together with the constraints to achieve a timetable for the rest of the senior department. When the true variables are stated, the clauses make sure that all other variables are set to false. We first tried this option, but the instance with 4,415,400 variables and over 11 million clauses could not be pre–processed by `hypre`, for it was aborted within 27 minutes.

State all variables The next option is to state all variables referring to the vwo, but the result was the same: `hypre` was aborted after an hour trying to cope with an instance with 4,415,400 variables and almost 13 million clauses.

Erase superfluous clauses When the timetable of the vwo is known and all these variables are stated together with the constraints to achieve a timetable for the rest of the senior department, all clauses which refer to the vwo only aren't needed anymore. When these clauses are removed, we ended up with 4,415,400 variables and almost 9 million clauses, but again, `hypre` was aborted after 5 minutes.

Erase studentgroups of vwo We tried the same trick as we've used when we went from senior department to junior department. The only implications of the timetable of the vwo for the rest of the senior department is the occupation of the teachers. So we erased the studentgroups of the vwo and modelled each year of the vwo as a single 'class'. We ended up with an instance with 2,849,940 variables and over 7 million clauses, which was pre–processed by `hypre` within 40 seconds. The simplified instances contained 268,543 variables and over 500 thousand clauses, but couldn't be solve by either `Jerusat3` or `zChaff` within 24 hours.

Partial timetabling continued: havo Maybe the rest of the senior department was too much to be scheduled at once. So we now tried to construct a timetable for the havo only, keeping the timetable of the vwo as it is. We ended up with 943,920 variables and over 3 million clauses, which was pre–processed within three minutes. We ended up with an instance 9,211 variables and almost 400 thousand clauses, but again couldn't be solved within 24 hours.

Partial timetabling continued: mavo Now we tried to schedule the mavo instead of the havo. After pre–processing an instance with 758,160 variables and over 3 million clauses, which resulted in an instance which 4,101 variables and over 422 thousand clauses, both `Jerusat3` and `zChaff4` couldn't find an satisfying assignment within 24 hours.

After all our efforts we still haven't construct a timetable. The next option is to allow more periods per days, so $T = 10$. This seemed to work. We first again constructed a timetable for the vwo only. The next quantities appeared.

K= 42	:	Total number of studentgroups in the vwo, which was divided in the next way :
		vwo 4 : 12 studentgroups
		vwo 5 : 14 studentgroups
		vwo 6 : 16 studentgroups
D= 78	:	Total number of subject–teachers in the vwo, who represent 52 real teachers
U= 3	:	Maximum number of lessons of a subject
S= 5	:	Number of day per week
T= 10	:	Number of periods per day

This comes down to 42 x 78 x 3 x 5 x 10 = 491,400 variables

We first simplified the instances, which took about three minutes to be constructed, with `hypre` and the construction of a timetable for the vwo only seems to be a piece of cake. It is remarkable that even though both solvers seem to find an satisfying assignment very fast, `zChaff4` is indeed very quick.

Table 8.11: Pre-processing partial period assignment vwo first senior department for $T = 10$ after tripling by `hypre`

Student sectioning by	`Jerusat3`	`zChaff4`
Initial variables	491400	491400
Initial clauses	3242709	3231909
Resulting variables	9130	7530
Resulting clauses	446534	362988
`hypre`	6	6

Table 8.12: Partial period assignment vwo first for $T = 10$ after tripling

Student sectioning by	`Jerusat3`	`zChaff4`
Variables	9130	7530
Clauses	446534	362988
`Jerusat3`	12	18
`zChaff4`	0	0

After the vwo we proceed to the rest of the senior department. The next quantities appeared in the senior department. We see that all years of the vwo are represented by one single ‘studentgroups’, which reduces K.

K= 71	:	Total number of studentgroups in the senior department, which was divided in the next way :
		vwo 4 : 1 studentgroups
		vwo 5 : 1 studentgroups
		vwo 6 : 1 studentgroups
		havo 4 : 22 studentgroups
		havo 5 : 13 studentgroups
		mavo 4 : 33 studentgroups
D= 223	:	Total number of subject–teachers in the senior department, who represent 60 real teachers
U= 4	:	Maximum number of lessons of a subject
S= 5	:	Number of day per week
T= 10	:	Number of periods per day

This comes down to 71 x 223 x 4 x 5 x 10 = 3,166,600 variables

We first simplify the instances and then solve the simplified instances. It took about 500 seconds to state the instances. It can be seen in table 8.13 that one instance turned out to be unsatisfiable. It is of course possible that the timetable of the vwo has such unfortunate consequences that a timetable for the rest of the senior department can't be constructed. Fortunately, the other instances were not unsatisfiable. Again, the solvers are both very fast, but the increased number of variables and clauses doesn't seem to affect the runtimes of `zChaff4`, as table 8.14 shows.

Table 8.13: Pre-processing remaining period assignment senior department for $T = 10$ after tripling by `hypre`

Student sectioning by	`Jerusat3`	`Jerusat3`	`zChaff4`	`zChaff4`
Timetable vwo by	Jerusat3	zChaff4	Jerusat3	zChaff4
Initial variables	3166600	3166600	3166600	3166600
Initial clauses	8406134	8406134	8386934	8386934
Resulting variables	15678		15678	15673
Resulting clauses	1152462		1157127	1156908
`hypre`	56	3 (unsat)	56	57

Table 8.14: Remaining period assignment senior department for $T = 10$ after tripling by `hypre`

Student sectioning by	`Jerusat3`	`zChaff4`	`zChaff4`
Timetable vwo by	`Jerusat3`	`Jerusat3`	`zChaff4`
Variables	15678	15678	15673
Clauses	1152462	1157127	1156908
Jerusat3	42	61	49
zChaff4	1	1	0

Finally, we arrive at the junior department. The next quantities appeared in the junior department.

K= 45	:	Total number of studentgroups in the senior department, which was divided in the next way : senior department : 6 years junior department : 39 classes
D= 403	:	Total number of subject–teachers, who represent 125 real teachers
U= 5	:	Maximum number of lessons of a subject
S= 5	:	Number of day per week
T= 10	:	Number of periods per day

This comes down to 45 x 403 x 5 x 5 x 10 = 4,533,750 variables

But again, at this amount of variables problems rose. Over 8 million clauses were stated and `hypre` couldn't cope with it, just like `Jerusat3` and `zChaff4`. So we had to come up with a solution. When we looked at the curriculum of the junior department, it turned out that only one subject, `ne` by subject–teacher `vrb-ne` for the first year, was being taught five times a week. No other subject required this amount of lessons. Because there are five classes in the first year, for these five fifth lessons of the subject `ne` U wasn't equal to 4 but to 5. If we could find a way to avoid these fifth lessons, we would reduce the number of variables with 20%.

We could achieve this in the next way. Subject–teacher `vrb-ne` teaches five lessons to class `r1a` and this leads to a '5' in the curriculum. To get rid of this '5' we use the next trick. We add another subject–teacher to the curriculum, who teaches the subject `ne` to `r1a` one lesson. So subject–teacher `vrb-ne` now only has to teach 4 lessons to class `r1a`. If we now also state that the added subject–teacher and subject–teacher `vrb-ne` refer to the same teacher, we actually have the same situation as we had, only with U reduced from 5 to 4. The only consequence is that D increases from 403 to 406. We have to add this extra subject–teachers three times, because we're constructing a timetable for a virtual school of triple size, so also three times the junior department. But this only leads to a slight increase of variables. The results of this trick are shown below.

K= 45	:	Total number of studentgroups in the senior department, which was divided in the next way : senior department : 6 years junior department : 39 classes
D= 406	:	Total number of subject–teachers, who represent 125 real teachers
U= 4	:	Maximum number of lessons of a subject
S= 5	:	Number of day per week
T= 10	:	Number of periods per day

This comes down to 45 x 406 x 4 x 5 x 10 = 3,654,000 variables

So now `hypre` could indeed simplify the instances, which took about 500 seconds to be stated. After simplification, no problems rose at constructing a timetable for both `Jerusat3` and `zChaff4`. Table 8.16 shows that also at this point `zChaff4` is by far quicker than `Jerusat3`.

Concluding we can say that it is possible construct a timetable for a school of this size, but several problems had to be overcome. The first problems rose at student sectioning, so we had to force a student sectioning ourselves. Next, constructing a timetable for the senior department gave some problems, which lead to several improvements of the algorithm to state the clauses, like removing the zero columns. In the end, it turned out that increasing T from 9 to 10 lead to a formula which could be solved by both solvers easily. Another trick was needed to construct a timetable at the junior department. Finally, we can say that at the instances both solvers could handle, `zChaff4` performs by far better than `Jerusat3`.

Table 8.15: Pre-processing period assignment junior department for $U = 4$ after tripling by `hypre`

Student sectioning by	Jer3	zCh4	zCh4	Jer3	zCh4	zCh4
Timetable vwo by	Jer3	Jer3	zCh4	Jer3	Jer3	zCh4
Senior department by	Jer3	Jer3	Jer3	zCh4	zCh4	zCh4
Initial variables	3654000	3654000	3654000	3654000	3654000	3654000
Initial clauses	6997226	6997292	6997308	6997226	6997292	6997308
Resulting variables	36820	36789	36777	36941	36808	36863
Resulting clauses	1520389	1517535	1515927	1529414	1521715	1523453
`hypre`	30	30	30	30	30	29

Table 8.16: Period assignment junior department for $U = 4$ after tripling by `hypre`

Student sectioning by	Jer3	zCh4	zCh4	Jer3	zCh4	zCh4
Timetable vwo by	Jer3	Jer3	zCh4	Jer3	Jer3	zCh4
Senior department by	Jer3	Jer3	Jer3	zCh4	zCh4	zCh4
Variables	36820	36789	36777	36941	36808	36863
Clauses	1520389	1517535	1515927	1529414	1521715	1523453
`Jerusat3`	367	315	318	173	547	199
`zChaff4`	6	7	7	5	6	5

8.2.3 Adjustments by solver

We've already seen that adjustments are an essential part of timetabling. Now we'll take a look at the performance of the solvers with our current size. We analyze situations similar to section 7.2 and 7.3. In this subsection we look at the performance of the solvers when an existing timetable has to be adjusted, like blocking a single period of a day. In the next subsections we look at daily adjustments and weekly adjustments. So for now, we take a look at the same situations as in section 7.2 and compare the performance of the solvers. We should point out that four changes have taken place. First, the size of the school has tripled. Second, we now pre–process the instances, so the solvers find a satisfying instance for a simplified formula created by `hypre`. Also, we use a newer version of `zChaff`, `zChaff 2004.5.13`, and we use an Intell Pentium 4 CPU 3.06 GHz computer, which is a different (and better) computer.

Change one day of a class and fix no lessons
We first took a look at the situation when we want to change a single day of a single class. The results are shown in table 8.17. When an instance which has to be simplified by `hypre` is unsatisfiable, `hypre` returns unsatisfiable, otherwise `hypre` indicates that it is yet unknown if the instance is satisfiable or not, and returns the simplified formula. We see that all instances are pre–processed and solved pretty quickly, as was the case in section 7.2, when the performance of the solvers at the Rotterdamsch Lyceum was examined. Another thing that strikes, is that the runtimes of every instance were almost equal, as the standard deviation of both `hypre` and `zChaff4` shows. This will occur on all of the following tests.

Table 8.17: Change one day of a class and fix no lessons

	`hypre`		`zChaff4`	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	26		8	
Standard deviation	0		0	
Total runs	20			

Change one day and fix three days of a class
Next we added the extra constraint that three days have to be fixed for the same class a day is being changed. Results are shown in table 8.18. Actually the same pattern is to be seen as the previous case. Even though we added several extra constraints the runtimes didn't seem to rise a bit.

Table 8.18: Change one day and fix three days of a class

	`hypre`		`zChaff4`	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	26		8	
Standard deviation	0		0	
Total runs	20			

Block one period of a class and fix no lessons
Next we blocked a single period of a class, but fixed no lessons. Results are shown in table 8.19. Again, the same patterns as the two previous cases.

Block two periods and fix three days of a class
We now fix three days of a class, and block two periods of the remaining days for the same class. Results are shown in table 8.20. Again, all instances are solved pretty easily, but now a different picture rises than in section 7.2. In section 7.2 only 70 % was satisfiable and now

Table 8.19: Block one period of a class and fix no lessons

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	26		8	
Standard deviation	0		0	
Total runs	20			

all instances were satisfiable.

Table 8.20: Block two periods and fix three days of a class

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	26		6	
Standard deviation	0		2	
Total runs	20			

Now we've looked at the similar situations as in section 7.2, we can conclude that with the four differences mentioned in the beginning of this section, all cases were solved pretty easily and mostly showed the same picture as in section 7.2. So tripling the size of the school doesn't have a real effect on the adjustments by the solver.

8.2.4 Daily adjustments

The second kind of adjustments we looked at were the daily adjustments. These were adjustments to an existing timetable needed when teachers or classes are absent. We look at the same situations as in section 7.3.1, starting with the situation when only one teacher is absent.

One teacher and no classes absent
We try to construct a timetable where no idle periods are allowed at all from the 2^{nd} period on. In table 8.21 the results are shown. We see that all runtimes are faster than the runtimes of the adjustments by the solver. This is because we now only consider a single day. Furthermore, every instance was solved quickly, which was not the case in section 7.3.1.

Table 8.21: One teacher and no classes absent without idle periods

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	4		0	
Standard deviation	0		0	
Total runs	20			

Five teachers and no classes absent

The next situation involved five absent teachers. Still, no classes were absent. The results are shown in table 8.22. Although now we see again that every time very quick a solution was found or unsatisfiability was determined, this is a rather different picture as in section 7.3.1, because there a larger part was unsatisfiable. This can be explained because we now look at a school which is three times the size of the Rotterdamsch Lyceum. The impact of five absent teachers is not that big in the current situation. So we take a comparable situation, namely when fifteen teachers are absent.

Table 8.22: Five teachers and no classes absent without idle periods

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	90 %	10 %	100 %	0 %
Average	4	0	0	
Standard deviation	1	0	0	
Total runs	20			

Fifteen teachers and no classes absent

Indeed, we now see a pattern that corresponds with section 7.3.1 as table 8.23 shows. And again, we would like to achieve a better ratio of satisfiable and unsatisfiable instances. This could be achieved by allowing one idle period. And if we look at table 8.24, this indeed is the case. Although in section 7.3.1 we needed to allow two idle periods per day, we now need only one.

Table 8.23: Fifteen teachers and no classes absent without idle periods

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	90 %	10 %	56 %	44 %
Average	5	0	0	0
Standard deviation	1	0	0	0
Total runs	20			

Table 8.24: Fifteen teachers and no classes absent with one idle period allowed

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	85 %	15 %
Average	4		0	0
Standard deviation	0		0	0
Total runs	20			

Five teachers and five classes absent
Finally, we look at the situation where five teachers and five classes are absent. It turned out these situations can be handled quite easily, as is shown in table 8.25.

Table 8.25: Five teachers and five classes absent without idle periods

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	95 %	5 %
Average	4		0	0
Standard deviation	0		0	0
Total runs	20			

No real difficulties rise at daily adjustments, when the size of the school is tripled. Also the case which was the most difficult in section 7.3.1 is handled easily.

8.2.5 Weekly adjustments

At the weekly adjustments, the teachers and the classes are not absent for a day, but for a week.

One teacher and no classes absent
We first look at the situation when one teacher is absent for a week. The results in table 8.26 show that this can be achieved quickly. In section 7.3.2 several times occurred when over an hour was needed to find a result, now no such situation occurred.

Five teachers and no classes absent
Again, no difficulties rise at this stage, as is shown in table 8.27. So we try to do the same as the previous subsection, and raise the number of absent teachers to fifteen. Table 8.28 shows this has no real impact.

Table 8.26: One teacher and no classes absent without idle periods from 2^{nd} until 6^{th} period

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	26		0	1
Standard deviation	0		0	
Total runs	20			

Table 8.27: Five teachers and no classes absent without idle periods from 2^{nd} until 5^{th} period

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	25		1	
Standard deviation	1		0	
Total runs	20			

Table 8.28: Fifteen teachers and no classes absent without idle periods from 2^{nd} until 5^{th} period

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	24		1	
Standard deviation	1		0	
Total runs	20			

Five teachers and five classes absent

The last case considers the absence of five teachers and five classes. No difficulties rise at solving these instances, as table 8.29 shows.

Table 8.29: Five teachers and five classes absent without idle periods from 2^{nd} until 5^{th} period

	hypre		zChaff4	
	Unknown	Unsatisfiable	Satisfiable	Unsatisfiable
Percentage	100 %	0 %	100 %	0 %
Average	25		0	
Standard deviation	0		0	
Total runs	20			

We see that all these cases are solved pretty easily by the solver. In section 7.3.2 several difficulties occurred. Several instances needed over an hour to be solved. Now, no such situation occurred. Also, when several teachers are absent, now no difficulties rise, whereas this gave several problems in section 7.3.2. The last thing is that the runtimes are now by far quicker than in section 7.3.2. This can again be explained by the four differences we pointed out at the beginning of this section.

Chapter 9

Conclusion and further research

In this thesis we described how the school timetable problem can be transformed into a SAT problem. We showed how several demands of a real life school, the Rotterdamsch Lyceum, can be transformed into constraints. Even though most demands could be handled quite easily, several needed a little bit of creativity.

After this, we looked at the performance of the solvers in creating a new timetable and adjusting an existing timetable. It turned out that it is possible to construct a timetable in this way, but the solver `Jerusat` was the only solver that performed solidly. At adjusting an existing timetable, in most cases a satisfying assignment was found pretty quick. Also, when the formula was unsatisfiable, this result was returned quickly.

Because the Rotterdamsch Lyceum is considered to be a small school, we expanded the school virtually to gain a larger school. Even though for a school of double size of the Rotterdamsch Lyceum the timetable was easily constructed, several difficulties rose at a school of triple size. But with a little creativity we could also construct a timetable for this school. Now the solver `zChaff` was by far the best solver. We also looked at adjustments to the timetable of the school of triple size. These were easily handled and results were returned quickly.

Even though the results are promising, still much research can be done on this subject. It will be clear that other schools have other rules, other requirements and other desires. Also, a total new way of organization of the school is possible. It is interesting to see if this method also suits other demands of other schools. We already saw that difficulties rise at larger schools, so especially this aspect should be examined and large schools should be picked to test this method.

We've seen that at constructing timetables for the Rotterdamsch Lyceum `Jerusat` performed best, whereas at the school of triple size of the Rotterdamsch Lyceum `zChaff` performed best. It is interesting to see if a specific solver or a specific method of a solver is suited best for school timetabling.

We didn't use satisfiability in all aspects of timetabling. Instead, we used simple algorithms to assign rooms or to force a clustering because of the size of the school. These algorithms are surely not the best one can think of, and if this method of constructing timetables matures, these algorithms should be improved.

Finally, it is interesting if this method can also be applied to other scheduling problems. We already saw that the student sectioning problem can be transformed to the period as-

signment problem, which we knew how to handle. So why can't other scheduling problems be modelled in this way? Other scheduling problems would need other requirements, and it would be interesting to see if these requirements could also be stated as constraints as we did.

Appendix: Survey

To gain insight in the effort of timetable designers in The Netherlands to construct a timetable we distributed a survey among Dutch timetable designers. Nine surveys were returned and the results are shown below. We've translated only the interesting parts of the surveys into English. It should be noted that every timetable designer follows different steps to construct a timetable, so any comparison will be difficult.

Bernardius College, Heerlen

Students ca 1400
Students in senior department ca 650

Steps in constructing a timetable	runtime software	working time timetable designer
Students per cluster	2 days	2 weeks
Collect data on lessons	1 day	2 weeks
Add teachers of physical education (double periods, separate sexes)	3 days	3 days
Add clusters senior department	2 days	1 week
Add junior department	2 days	1 day
Add desires teachers	1 day	2 weeks
Finish up	1 day	3 days

Bisschoppelijk College Broekhin, Roermond

Students 1235
Students in senior department 776

Steps in constructing a timetable	runtime software	working time timetable designer
Collecting data teachers	?	4-6 weeks
Input software		
Timetabling		
Improving timetable		
Printing timetable		

Bogerman, Sneek

Students ca 2280
Students in senior department ca 620

Steps in constructing a timetable	runtime software	working time timetable designer
Clustering	3 hours	3 hours
Timetabling vmbo 3 & 4	5 days	5 days
Timetabling senior department of havo & vwo	2 weeks	2 weeks
Timetabling junior department	1 week	1 week
Optimization room assignment	1 week	1 week

Christelijk Gymnasium Sorghvliet, Den Haag

Students ca 575
Students in senior department 255

Steps in constructing a timetable	runtime software	working time timetable designer
Add students curricula		2 hours
Clustering	20 minutes per try	several tries for one month
Add data for clustering		30 minutes
Final clustering	2 hours	
Add data for timetabling		2 hours
Timetabling	30 minutes per try	4 days

Emelwerda College, Emmeloord

Students 1550
Students in senior department 830

Steps in constructing a timetable	runtime software	working time timetable designer
Proposal teachers curricula	30 minutes	20 hours
Add data timetabling		50 hours
Clustering	60 minutes	35 hours
Timetabling senior department	10 hours	140 hours
Timetabling junior department	4 hours	60 hours

***r*S*g* Lucia Petrus, Rotterdam**

Students ca 350
Students in senior department ca 125

Steps in constructing a timetable	runtime software	working time timetable designer
Clustering		2-3 days
Add data timetabling		2 days
Timetabling physical education, etc		1.5 days
Timetabling rest		2 days
Room assignment		0.5 days
Printing and distribution		0.5 days

Note: Most of timetabling is done by hand. Software is only used several minutes.

rSg Rotterdamsch Lyceum, Rotterdam

Students ca 550
Students in senior department 215

Steps in constructing a timetable	runtime software	working time timetable designer
Processing students curricula		3 days
Renew data timetabling		2 days
Data hard lessons, like crafts		0.5 days
Timetabling first time	30 min	
Adjusting settings		2 hours
Timetabling other lessons	2 hours	2 days
Adjusting timetable		2 days
Printing and distribution		1 days

Luzac College Breda, Breda

Students 60
Students in senior department 60

Steps in constructing a timetable	runtime software	working time timetable designer
Timetabling by hand		several days
Input timetable in software	several days	several days

Note: Luzac College doesn't have a junior department

Mollercollege Steenbergen, Steenbergen

Students ca 550
Students in senior department ca 300

Steps in constructing a timetable	runtime software	working time timetable designer
Teacher curricula		1 week
Add data timetabling		1 week
Timetabling senior department		2 weeks
Timetabling junior department		0.5 week
Checking timetable		0.5 week

Note: large part of timetabling is still done by hand.

We also asked the timetable designers to point out the most important part of timetable software. The next features were mentioned:

- possibilities of element in timetable
- user-friendly
- communication with other software (twice)
- setting features of timetable, like distribution of studyload over a week (twice)
- possibility of adjustments by hand (twice)
- timetabling

Only once the performance on timetabling is mentioned as the most important feature of timetable software. To timetable designers is it obviously more important to be able to work easily with the software. They don't want software that can construct a timetable for them, they want easily usable software that can help them at constructing a timetable.

Bibliography

[Bac 03] Bacchus F. and J. Winter, *Effective Preprocessing with Hyper-Resolution and Equality Reduction*, Sixth International Symposium on Theory and Applications of Satisfiability Testing 2003.

[Ber 03] Berends. M. and Leo Wijnhoven, *Voorgezet Onderwijs, gids voor ouders, verzorgers en leerlingen 2003-2004*, Ministerie van Onderwijs, Cultuur en Wetenschap, Woerden, 2003.

[CBS 04] Aart, S.A. van der, et al., *Jaarboek onderwijs in cijfers 2003-2004 Feiten en cijfers over het onderwijs in Nederland*, Kluwer, 2004. ISBN 90 13 0 0251 X

[DIM 93] DIMACS, *Satisfiability Suggested Format*, 1993 www.cs.ubc.ca/∼hoos/SATLIB/Benchmarks/SAT/satformat.ps.

[Kes 99] Kesteren, Bernard van, *The Clustering Problem in Dutch High Schools, Changing Metrics in Search Space*, Leiden University, 1999, Internal report.

[Kra 92] Krautz, Henry and Bart Selman, *Planning as Satisfiability*, In Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92), Vienna, 1992

[Mos 01] Moskewicz, Matthew W., Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. *Chaff: Engineering an Effëcient SAT Solver*, In Proceedings of the 38th Design Automation Conference (DAC'01), June 2001.

[Nad 02] Nadel, Alexander, *Backtrack Search Algorithms for Propositional Logic Satisfiability: Review and Innovations*, Hebrew University of Jerusalem, 2002, Master of Science thesis.

[Sca 95] Schaerf, A., *A survey of automated timetabling*, CWI Report CS-R9567, Amsterdam, 1995.

[Scr 84] Schreuder, J.A.M., J.A. van der Velde, *Timetables in Dutch High Schools*, Operations Research '84, 1984.

[Wil 02] Willemen R.J., *School timetable construction. Algorithms and complexity*, Technische Universiteit Eindhoven, 2002, PhD thesis, ISBN 90-386-1011-4.