School timetable construction

Algorithms and complexity

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Willemen, Robertus J.

School timetable construction : algorithms and complexity /
by Robertus J. Willemen. -
Eindhoven : Technische Universiteit Eindhoven, 2002. Proefschrift. -
ISBN 90-386-1011-4
NUGI 855
Subject headings: timetables / combinatorial optimisation / computational complexity / heuristics.
CR Subject Classification (1998): G.2.3, I.2.8, F.2.2, H.4.2

Cover design: Paul Verspaget
Printed by Universiteitsdrukkerij Technische Universiteit Eindhoven

School timetable construction
Algorithms and complexity

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. R.A. van Santen, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
dinsdag 16 april 2002 om 16.00 uur

door

Robertus Johannes Willemen

geboren te Breda

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. E.H.L. Aarts
en
prof.dr. J.K. Lenstra

Copromotor:
dr.ir. H.M.M. ten Eikelder

The work described in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

IPA Dissertation Series 2002-05.

Contents

1

Introduction

This thesis is concerned with the problem of constructing timetables for schools. School timetable construction problems are interesting objects to study because neither modeling nor solving them is straightforward. It is difficult to make a clear-cut distinction between acceptable and not acceptable timetables. Because of the large diversity in acceptance criteria, realistic timetable construction problems are multi-dimensional. Each dimension may introduce its own characteristic aspects that add to the complexity of the problem. Therefore, only heuristic solution approaches without known performance guarantees are practically feasible.

We pay special attention to timetable construction problems at Dutch secondary schools. A typical characteristic of Dutch secondary schools is the students' freedom of composing their educational programs. Therefore, subjects of the same level can have different sets of students taking classes in this subject. As the number of students for popular subjects often exceeds the classroom's capacity, planners frequently need to section the set of students into smaller groups. This decision is not typical for school timetable construction problems. However, it frequently occurs in university course timetabling problems.

The organization of this chapter is as follows. In Section 1.1, we introduce the reader to timetable construction problems at Dutch secondary schools. We give an outline of the thesis in Section 1.2. In that section, we highlight the main themes of the thesis in the context of previous work.

1.1 Background

1.1.1 Timetabling

Timetabling concerns all activities with regard to making a timetable. According to Collins Concise Dictionary (4th Edition) a *timetable* is a *table of events arranged according to the time when they take place*. The events are usually meetings between people at a particular location. Consequently, a timetable specifies which people meet at which location and at what time. A timetable must meet a number of requirements and should satisfy the desires of all people involved simultaneously as well as possible. The timing of events must be such that nobody has more than one event at the same time.

Since 1995, a large amount of timetabling research has been presented in the series of international conferences on Practice And Theory of Automated Timetabling (PATAT). Papers on this research have been published in conference proceedings, see e.g., [Burke & Carter, 1997] and [Burke & Erben, 2000], and three volumes of selected papers in the Lecture Notes in Computer Science series, see [Burke & Ross, 1996], [Burke & Carter, 1998], and [Burke & Erben, 2001]. Moreover, there is a EURO working group on automated timetabling (EURO-WATT) which meets once a year, regularly sends out a digest via e-mail, and maintains a website with relevant information on timetabling problems, e.g., a bibliography and several benchmarks.

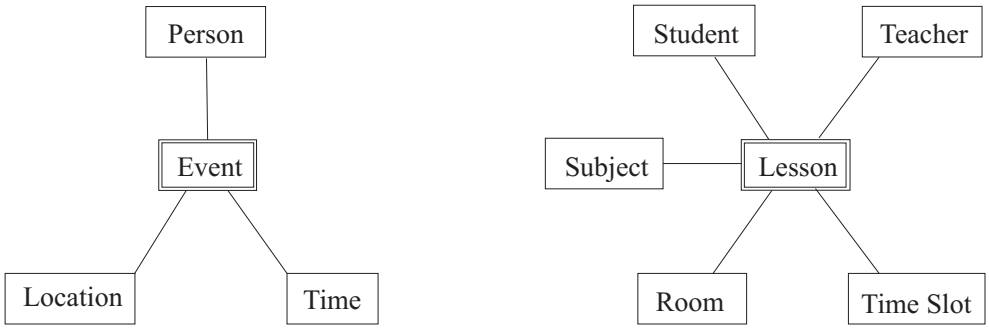
1.1.2 Educational timetabling

Educational timetabling is the sub-class of timetabling for which the events take place at educational institutions. Examples of events in this sub-class are

- tests or examinations at schools or universities (*examination timetabling*),
- lessons, i.e., meetings between a class and a teacher, at schools (*class-teacher timetabling* or *school timetabling*),
- lectures in courses offered at a university (*university course timetabling*).

Each category has its own characteristics. In examination timetabling, there should be sufficient time between consecutive exams of the same student. Teacher availabilities and reducing idle time for students play an important role in class-teacher timetabling. At universities, there are often many different curricula, such that there exist no conflict-free timetable for every student within the given time. Therefore, those in charge of timetabling, the *planners*, try to find the timetable with the least conflicts.

There are several surveys on educational timetabling problems. Schaerf [1995] gives an overview of the literature on problems that belong to the three categories we mentioned. There are also recent overviews on examination timetabling [Carter & Laporte, 1996] and university course timetabling [Bardadym, 1996] [Carter & Laporte, 1998].



(a) (b)

Figure 1.1. Concepts and relations between concepts for timetabling in general (a) and timetable construction at schools in particular (b). Concepts are represented by boxes and relations by lines connecting the boxes. We only present relations with the central concept.

The literature on educational timetabling problems is quite scattered. Different papers may refer to the same type of institution but they mostly deal with different kinds of assignments, i.e., decisions like the timing of events, sectioning students into groups, or assigning events to locations. Moreover, each institution has its own characteristics which are reflected in the problem definition. Therefore, no two problems are exactly the same. Consequently, there are only a few benchmarks, most of them on examination timetabling problems [Di Gaspero & Schaerf, 2001] and some on university course timetabling problems [Lewandowski, 1994]. However, the majority of papers on educational timetabling problems can be classified along the three categories we mentioned above. Our school timetable construction problem is an example of an educational timetabling problem that does not exactly fit within one of these categories.

1.1.3 Timetable construction problems at schools

In this thesis, we focus on timetable construction problems at *schools*. For these timetabling problems, the events are lessons in a subject, taught by a teacher to a group of students, sometimes referred to as a *class*, in a room. In Figure 1.1 we show the concepts of timetabling in general (a) and timetable construction at schools in particular (b). In both cases there is a central concept to which all other concepts, most of them representing the necessary resources, are related.

Decisions. Timetable construction problems are mainly about allocating resources, i.e., teachers, students, rooms, and time slots, to lessons. Three of these decisions, viz. *teacher assignment*, *room assignment*, and *time slot assignment* are

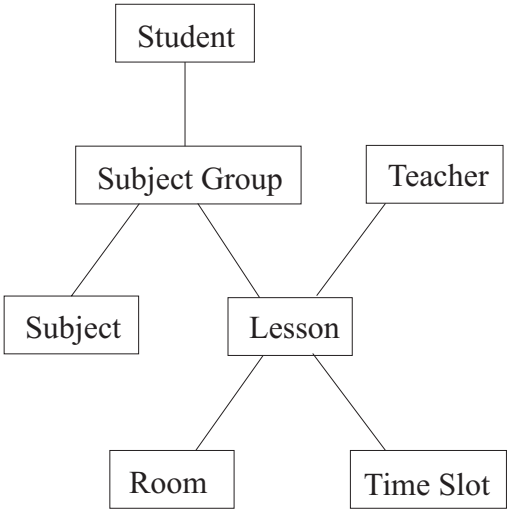


Figure 1.2. Introduction of the concept of subject group. Students are assigned to subject groups and, via these subject groups, to lessons.

conceptually clear: we must assign a teacher, a room, and a time slot to each lesson. The other one, assigning students to lessons, is carried out implicitly. We briefly describe how this is done. Let us look at one specific subject, which can be either mandatory or optional. There is a group of students that take lessons in this subject. That is, all students in the group have chosen this subject in their curricula. If the number of students is small enough, all students fit in one room and therefore can take the same lessons in the subject. On the other hand, if the number of students is large, we must divide the group of students into a number of groups that are small enough. These groups are specific to the subject; therefore, we call them *subject groups* in the remainder. We refer to the decision of assigning each student to one of these subject groups, or dividing the group of students into subject groups, as *subject group assignment*. In the literature, the names *student sectioning*, *section assignment*, and *student scheduling* are frequently used in this context. In Figure 1.2, we show the relation of subject groups to students, subjects and lessons.

There are some other decisions that must also be carried out at schools. In this context, we mention determining the number of subject groups per subject, assigning days off to teachers, and putting restrictions on the students' curricula. In the remainder, we do not focus on these decisions. But, for the sake of completeness, we briefly describe them here.

In practice, school management decides on the number of subject groups for each subject. All subject groups of the same subject have the same number of lessons. Assigning a student to a subject group implies assigning the student to all lessons of

the subject group. Note that we only assign a student to subject groups corresponding to the subjects in the student's curriculum.

Nowadays, many teachers do not have full timetables. That is, the number of lessons they teach is much less than the total number of time slots in a week. This implies that teachers have *idle time* in their timetables. Since labor agreements require that timetables for these *part-time teachers* are compact, their idle time may not be spread randomly over all days of the week. Restrictions of the sort that every teacher that teaches x lessons should have y parts of days off introduce new decisions in the timetabling problem.

In upper years, students have much freedom in composing their curricula. Having no restriction on the subject selection might lead to a large number of different curricula. Also, it may imply many conflict constraints as no two lessons for the same student can be taught at the same time. Therefore, exotic subject combinations are often ruled out by school management.

Conditions. Some combinations of assignments lead to acceptable timetables, others do not. Such restrictions follow from conditions imposed by rooms, students or teachers. We distinguish two types of conditions: conditions that must be met (*requirements*) and conditions that should be fulfilled as well as possible (*desires*).

Requirements. It is impossible for a teacher or student to attend more than one lesson simultaneously. Therefore, every teacher or student can have at most one lesson at the same time. We call these requirements *teacher constraints* and *student constraints*. Similarly, *room constraints* are only satisfied if each room is used for only one lesson at a time.

Teachers can be unavailable for teaching lessons at some time slots. It is clear that a lesson can only be taught at a time the teacher is available. *Teacher availability constraints* force us to find time slot assignments that satisfy these requirements.

At many schools, rooms are dedicated to specific subjects. For example, language rooms are equipped with headsets and physics rooms with water taps, electricity sockets and gas. Moreover, there are large gymnastics halls in which only physical education lessons can be given. Of course, lessons can only be taught in rooms that are suitable. We can group rooms that are equipped for the same set of lessons and say that they have the same *room type*. By means of *room compatibility constraints*, we require that only rooms of a suitable type are assigned to each lesson.

There are two requirements that restrict the possible assignments of different lessons for the same subject group. The first requirement forces a spread of lessons for educational reasons. Lessons of the same subject group must be assigned to different days such that students can absorb the material and are able to do homework for the next lesson. We take *educational constraints* that forbid assignment of lessons of the same subject group to the same day into account.

The other requirement type enforces two or more lessons to be assigned to consecutive time slots on the same day. For example, two crafts or drawing lessons for

the same students must be taught consecutively. We call these two lessons a *block*, and the constraints on these lessons *block constraints*. Block constraints also require that there is no break separating lessons in a block and that all lessons are assigned to the same room.

Large schools often have buildings at different *locations*. Most of the time, planners try to schedule lessons for the same school type at one specific location. But still there may be teachers or even students that have to travel between two locations. Of course, lessons must be scheduled such that they satisfy the *travel time constraints*. That is, there must be sufficient travel time for teachers and students if they have to travel between two different locations.

Desires. Some conditions are less straightforward to define. Usually, these conditions must be fulfilled *as well as possible*. The planner must value the level of fulfillment of each condition. Often, two or more wishes cannot be complied with to the same extent. Then, the planner should weigh the desires and determine which timetable is most acceptable overall.

Some subjects have more than one subject group. In the ideal situation, all subject groups have equal size, i.e., each group has the average subject group size. In practice, it is not required that the subject group assignment is perfectly balanced: we only want to avoid small and large subject groups. Balancing the subject groups as well as possible is enforced by the *subject group size constraints*.

In lower years, all students have the same curriculum. All students at the same *level*, i.e., the same year of education of the school type, are partitioned into groups. We use the term *classes* for these groups of students. Classes are basic units in timetabling for lower years, as students in the same class follow all lessons together. This can be considered as *student group binding*, that is, each student has a large group of fellow students that belong to the same class. Actually, we cannot have a tighter student group binding than the one we described. It is difficult to achieve such a maximal student group binding in upper years, since students choose their own programs in these years. By assigning students to subject groups instead of lessons, a student will have exactly the same fellow students for all lessons in a subject. A planner should try to assign students to subject groups such that an acceptable student group binding is attained.

If a student has fewer lessons than the total number of time slots, he will have some idle time in his schedule. Students generally do not like idle time when it leads to gaps in their timetables. However, gaps in the student's timetable at the beginning or ending of the day are favorable, since a student can come to school later or go home earlier. Although the same applies to teachers, they usually have less trouble with occupying idle time during daytime with meaningful activities. Reducing unpleasant idle time to an acceptable amount per student or teacher is often an objective to planners.

1.1.4 Secondary schools in the Netherlands

This thesis pays special attention to timetabling problems at Dutch secondary schools. Therefore, we give an overview of the system of secondary schools in the Netherlands. A more detailed description can be found in [Wijnhoven, 2000]

Secondary education follows eight years of primary education and is for children aged 12 and over. In contrast to primary education, which is the same for all children, secondary education has different types of schools for students with different interest and talents.

School types. The current school system consists of four *school types*. Recently, the junior general secondary education (Middelbaar Algemeen Voortgezet Onderwijs, MAVO) and the old prevocational secondary education (Voorbereidend Beroepsonderwijs, VBO) have been replaced by a new prevocational secondary school type: Voorbereidend Middelbaar Beroepsonderwijs (VMBO). Like the former two school types, education in the VMBO lasts four years. The remaining two school types of the old system, senior general secondary education (Hoger Algemeen Voortgezet Onderwijs, HAVO) and pre-university education (Voorbereidend Wetenschappelijk Onderwijs, VWO), complete the current system. These types have course lengths of five and six years respectively. In Figure 1.3, we present the different school types before and after the change.

There are two types of VWO schools: the *atheneum* and the *gymnasium*. Education at gymnasiums differs from that at atheneums: Latin and Greek are mandatory at gymnasiums.

In the past, schools only taught one of the school types mentioned. Recently, many schools have merged into large combined schools offering course programs in several school types, if not all. Still, schools remain that offer only one type, but these are a minority.

Students and curricula. For each school type we distinguish two time periods: lower years and upper years. In the *lower years*, basic secondary education is given. This period usually lasts three years. In the *upper years*, students are prepared for their final examinations. The focus of our research has been on timetabling problems for students in upper years at HAVO and VWO schools. But for the sake of completeness, we briefly describe the education in lower years and at VMBO schools as well.

All school types start with a broad core curriculum in the lower years. These curricula consist of the same topics and only vary in level depending on the school type. The first year is a *transition period* at most schools. Students of these schools can postpone choosing the specific type of secondary school until the end of the transition period. Before the second and final stage of secondary education, the upper years, students have to choose their exam subjects.

The exam subjects take up the largest part of the student's curriculum in upper years. Students used to have a lot of freedom in composing their curricula. Apart

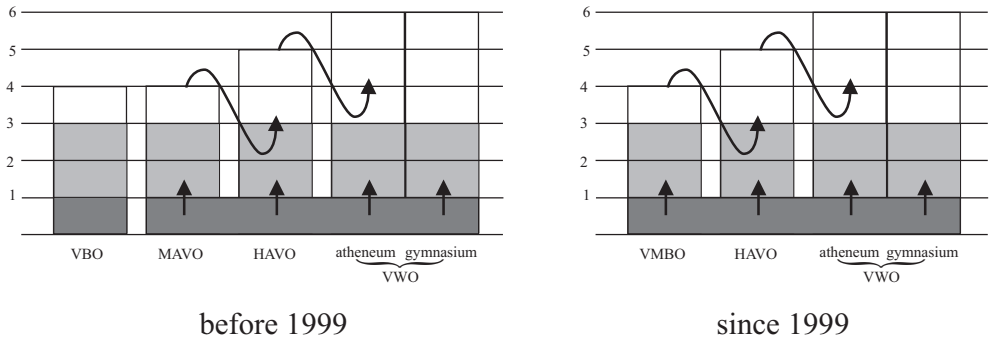


Figure 1.3. System of secondary schools in the Netherlands before and after the merge of VBO and MAVO in 1999. Vertically, we show the number of years of education. The first year is usually a transition period. The lower years end after the third year. In the upper years, students can compose their own program. The arrows indicate how students can flow through the system.

from VWO schools, where students had curricula of seven subjects, the VBO, MAVO, and HAVO students were examined in six subjects. Dutch and another modern language (English, French, or German) used to be mandatory for all students. At VBO schools, at least two subjects should have been related to the vocation the student had chosen.

Often, schools had imposed restrictions on pairs of subjects that must or should not be combined. A few years ago, the student's choice was further restricted. Students at the new VMBO schools sit exams in six subjects. Two subjects are mandatory: Dutch and English. Moreover, two other subjects in their curriculum must be related to the sector they have chosen: engineering and technology, care and welfare, business, and agriculture. Nowadays, students at HAVO and VWO schools have to choose one of four *fixed subject combinations*. The combinations belong to the four themes: (1) science and technology, (2) science and health, (3) economics and society, and (4) culture and society. Each subject combination consists of three parts: a common core of subjects, several specialized subjects, and some optional subjects. The common core consists of subjects mandatory to all students and takes up almost half the student's curriculum. The specialized subjects comprise more than a third of the curriculum and are arranged around the four themes. Finally, optional subjects can be either other subjects offered by the school or extensions of subjects in the common core or specialization chosen by the student.

1.2 Outline of thesis

This thesis discusses school timetable construction problems from both a theoretical and a practical point of view. First, we model timetable construction problems

at Dutch secondary schools. The result is a formal definition of our timetable construction problem TCP. Next, we perform a complexity analysis on TCP. Then, we focus on the core of TCP, i.e., time slot and subject group assignment. This problem is called TSSGAP. We develop a tree search algorithm for TSSGAP and try to construct feasible solutions for real-life instances. We conclude the thesis with a discussion on ways to improve the tree search algorithm and the practical relevance of the algorithm.

1.2.1 Modeling timetable construction problems

In Chapter 2, we give the formal definition of the TIMETABLE CONSTRUCTION PROBLEM (TCP), which is our model for timetable construction problems at Dutch secondary schools. Our functional description comprises the three major assignments of resources to lessons: room assignment, subject group assignment, and time slot assignment. Note that we assume the teacher assignment is given. The objective of TCP is to assign resources to all lessons while satisfying *all* constraints. Therefore, our problem is a *search problem*: we try to find a feasible solution to TCP.

We also define a shorthand in which sub-problems of TCP can be represented by means of several symbols. This prevents recalling the extensive list of input of TCP each time we discuss a sub-problem of TCP. It also enables determining the differences in input, decisions, and constraints between sub-problems more easily.

Little research has been published on timetable construction problems at Dutch secondary schools. No formal definition of the timetable construction problem is known that includes all three decisions we consider. Berghuis, Van der Heiden & Bakker [1964] consider a class-teacher timetabling problem that existed in the early sixties. The introduction of freedom in composing the students' program led to the clustering sub-problem, which is discussed by Simons [1974] and Van Kesteren [1999]. De Gans [1981] describes a heuristic to complete a timetable after the clustering problem has been solved. Bonvanie & Schreuder [1982] model timetable construction as a 0-1 programming problem and conclude that no timetable could be constructed with integer programming software at that time.

Since our problem has characteristics from both class-teacher timetabling and university course timetabling problems, there are also timetable construction problems outside the Netherlands that are of interest. Hertz & Robert [1998], Carter [2001], and Alvarez-Valdes, Crespo & Tamarit [2002] consider timetable construction problems in which lessons are also assigned to rooms, students, and time slots. Some of their conditions, e.g., no conflicts for students, are desires according to them. Therefore, they model them as soft constraints. We consider these conditions requirements and model them as constraints.

The minority of all papers give a formal definition of the problem under consideration. This is mainly because most realistic timetable construction problems have many different characteristics and cannot be defined formally in only one or two

paragraphs. Some authors model timetable construction problems as special cases of general problem formulations. Nuijten [1994] introduces the TIME AND RESOURCE CONSTRAINT SCHEDULING PROBLEM. Ferland [1998] discusses timetabling problems in the context of generalized assignment problems. Kingston [2001] defines a language in which many real-life timetable construction problems can be written. Reis & Oliveira [2001] also represent timetabling problems by means of a timetabling language.

1.2.2 Computational complexity of timetable construction problems

Chapter 3 deals with the computational complexity of timetable construction problems in general and TCP in particular. We present known and new complexity results on timetable construction problems in the context of TC, We give several independent reasons why TC is NP-complete. It is straightforward to show that GRAPH K-COLORABILITY and BIN PACKING [Garey & Johnson, 1979] are special cases of TC. But TC has other characteristics that give different reasons for its inherent complexity. For each characteristic, we define a polynomially solvable sub-problem of TC that does not include this characteristic. Next, we show that the problem becomes NP-complete when the characteristic is added.

In Chapter 3, we discuss three papers that settle the complexity of sub-problems of TCP. The best-known complexity result for a class-teacher timetabling problem with availabilities was published by Even, Itai & Shamir [1976]. Carter & Tovey [1992] give complexity results for several room assignment problems. Among others, they pay attention to the complexity introduced by room types and blocks. Cooper & Kingston [1996] give several independent reasons for the NP-completeness of their timetable construction problem TTC, which shares several characteristics with TC. They reduce known NP-complete problems to TTC but do not consider polynomially solvable variants.

Some other papers give complexity results for timetable construction problems that are not directly related to TCP. Asratian, Denley & Häggkvist [1998] derive complexity results based on restricted edge coloring models for bipartite graphs. Several results for the CLASS-TEACHER TIMETABLING PROBLEM and UNIVERSITY COURSE TIMETABLING PROBLEM on graph coloring and network flow models are given by De Werra [1985] and De Werra [1997]. De Werra, Asratian & Durand [2002] and Asratian & De Werra [2002] extend the class-teacher timetabling problem by introducing lectures taught by one teacher to more than one class. They give complexity results for some special cases of this problem, e.g., for the professor-lecturer model where some teachers, the professors, only teach groups of classes and others, the lecturers, only teach single classes. Giaro, Kubale & Szyfelbein [2000] derive complexity results for a variant of the CLASS-TEACHER TIMETABLING PROBLEM. They add constraints that forbid interruptions, i.e., unpleasant idle time, and settle the complexity of this problem for special types of graphs.

1.2.3 A core problem in timetable construction

In Chapter 4, we define the TIME SLOT AND SUBJECT GROUP ASSIGNMENT PROBLEM (TSSGAP). This problem is at the core of TCP and concerns time slot assignment, i.e., the construction of a timetable, and subject group assignment, i.e., sectioning students for popular subjects, while satisfying a number of hard constraints. Currently, planners at Dutch secondary schools first solve a variant of this problem before assigning rooms to lessons. We obtain several feasibility checks for TSSGAP, by considering sub-problems of TSSGAP like graph coloring and bin packing. By reviewing the relevant literature on these problems, we determine whether these checks are practically feasible. We conclude Chapter 4 with a discussion on the current solution approach for TSSGAP. This approach first determines clusters of lessons and then assigns these clusters to time slots. We discuss why we implement a completely different approach.

After presenting the formal definition of TSSGAP, we give an overview of problems in the literature and algorithms that were implemented. This overview is restricted to problems that contain both the time slot and the subject group assignment decisions. Many authors only consider one of these assignments. Much work has been done on constructing a timetable when students have been assigned to lessons. See for example Alvarez-Valdes, Martin & Tamarit [1996], Brunetta & De Poli [1997], Colorni, Dorigo & Maniezzo [1998], Costa [1994], and Schaerf [1996]. Student sectioning for a given timetable is studied by Feldman & Golumbic [1990], Graves, Schrage & Sankaran [1993], Laporte & Desroches [1986], and Miyaji, Ohno & Mine [1987] among others.

The clustering problem at Dutch secondary schools has been studied by Simons [1974] and Van Kesteren [1999]. Most popular commercial software packages like Clus and Untis of Deloitte & Touche ICS Adviseurs, Clusterfact and Roosterfact of SMS, Qlus+ and Perspectief of Malmberg, and Clusterbord and Roosterbord of BEP Software follow the same strategy and contain algorithms that solve the clustering problem.

1.2.4 Solving the core problem

In Chapter 5, we describe the solution approach we adopted for TSSGAP and the algorithm we implemented. The main idea is that we extend the subject group assignment step by step while maintaining a feasible timetable. So, we iterate over the subject group assignment and time slot assignment sub-problems. The algorithm is a hybrid, since we assign students to subject groups by means of a tree search algorithm and use a tabu search algorithm to repair the current timetable if it has become infeasible. We give the main procedures of the algorithm in pseudocode and discuss how we try to improve the performance of the tree search and tabu search algorithm by using problem-specific knowledge. The algorithm contains a number of parameters for variable and value selection, dead end handling, and the tabu search routine.

We give computational results for a number of real-life instances in Chapter 6. The instances are obtained from two medium-sized Dutch secondary schools: the Cambreurcollege in Dongen and the Pleincollege Sint-Joris in Eindhoven. Apart from trying to find solutions for all levels in the upper years at the same time, we also consider instances for each individual level. We carry out a large number of experiments to determine a good parameter setting. Among others, we compare several different ways of tackling dead ends like restarting with the bottleneck decision first and backtracking. Moreover, for tough instances, we approximate feasible complete solutions of TSSGAP by relaxing and removing constraints.

1.2.5 Improvements and practical relevance

Finally, in Chapter 7, we discuss how we can improve our algorithm. We suggest improvements for both the tree search algorithm and the tabu search algorithm. The thesis concludes with a discussion on the practical relevance of our research on TSSGAP. Here, we outline how the tree search algorithm can be used in a decision support system for constructing school timetables.

2

Timetable construction

In Section 1.1, we gave a conceptual model for timetable construction problems at Dutch secondary schools. We described concepts and their relationships, the decisions that have to be made, and the conditions for which the timetable is considered acceptable. In this chapter we give a formal definition of our timetable construction problem. The TIMETABLE CONSTRUCTION PROBLEM (TCP) comprises all major decisions and requirements faced by planners when constructing an initial school timetable.

The organization of this chapter is as follows. In Section 2.1, we summarize all practical details that we ignore in our model. Then, we give the functional model for TCP. We define the input, decision variables, constraints, and solutions of TCP by using sets and functions on these sets. In Section 2.3, we describe a shorthand for sub-problems of TCP. We conclude with a summary of this chapter.

2.1 Simplifications

In our timetable construction problem, some decisions are already given or left out of the model. We assume that

- the teacher assignment is given by the school management,
- the school management has determined the number of subject groups for each subject,

- we know the days, or parts of days, at which each teacher is unavailable for teaching lessons, and
- that all student choices for subjects must be granted.

We also simplify several conditions. We restrict ourselves to blocks of size two, because of all blocks these occur most frequently. Moreover, we do not pay attention to the time necessary to travel between rooms for two consecutive lessons. Finally, we ignore idle time for both students and teachers.

2.2 Functional model

In the present section, we define the TIMETABLE CONSTRUCTION PROBLEM (TCP). First, we introduce our notation for sets, functions, and upper bounds on sets. Next, we present all input of TCP. The input is divided into several parts, according to the most important concepts: time, teachers, rooms, subjects, student groups, and lesson. We continue with the definition of all decision variables and constraint types. Finally, we state TCP as a search problem.

2.2.1 Notation

In our functional model, we make use of sets, functions on these sets, and upper bounds on the number of elements in a set. In our definitions, $\mathbb{B}$ denotes the set of all booleans, i.e., $\mathbb{B} := \{\text{true}, \text{false}\}$, and $\mathcal{P}(X)$ denotes the power set of a set X. We represent sets by upper case italics, and elements of a set by the corresponding lower case. For example, L denotes the set of all lessons, D denotes the set of all teachers, and $l \in L$ and $d \in D$ are a lesson and a teacher, respectively.

All function names are of the kind f_{S}. In this name, f corresponds to the name of the function's range; we use lower case if only a single element is returned and upper case if a set or subset is returned. For example, all functions that return a set of lessons start with L, and the function that returns a teacher starts with a d. Moreover, the subscript S in the function name f_{S} is the roman equivalent of the function's domain name. That is, we denote the function that returns all lessons for a given teacher by L_{D}, because the function's domain is D. The inverse function, the one that gives the lesson's teacher for each lesson, is denoted by d_{L} because it returns a teacher d for each lesson in L.

We conclude with the notation of upper bounds. We denote an upper bound on the number of elements of a set X by $\overline{X}$. For example, $\overline{L}$ denotes the maximal number of lessons.

We give an overview of all identifiers we use in the Subject Index.

2.2.2 Input

We first define all input related to time. Since lessons have to be assigned to time slots, and lessons for the same subject group must be assigned to different days, due to the educational constraints, we introduce sets of time slots and days, and a relation

between time slots and days. Block lessons must be assigned to consecutive time slots not separated by breaks. We restrict ourselves to block lessons of size two and model this requirement by means of a constraint. Therefore, we introduce a predicate that denotes whether two time slots can be assigned to two lessons in a block.

Definition 2.1 (Time input). The four-tuple $\mathcal{I}_{\mathrm{T}} = \langle T, W, w_{\mathrm{T}}, \mathsf{BlockTime} \rangle$ defines the *time input*, where

- T denotes a finite set of time slots,
- W denotes a finite set of days,
- $w_{\mathrm{T}} : T \to W$, where $w_{\mathrm{T}}(t)$ denotes the day of a time slot $t \in T$, and
- $\mathsf{BlockTime} : T \times T \to \mathbb{B}$, where $\mathsf{BlockTime}(t_1, t_2)$ indicates whether two time slots $t_1, t_2 \in T$ are consecutive and not separated by a break.

The $\mathsf{BlockTime}$ relation is symmetric but not reflexive. That is, for all time slots $t_1, t_2 \in T$, $\mathsf{BlockTime}(t_1, t_2) \Leftrightarrow \mathsf{BlockTime}(t_2, t_1)$ and, for all time slots t, $\mathsf{BlockTime}(t, t) \Leftrightarrow$ false holds. □

The second input set contains all data with regard to teachers. We model the teacher availability by a function that returns the set of available time slots for each teacher.

Definition 2.2 (Teacher input). The pair $\mathcal{I}_{\mathrm{D}} = \langle D, T_{\mathrm{D}} \rangle$ defines the *teacher input*, where

- D denotes a finite set of teachers, and
- $T_{\mathrm{D}} : D \to \mathcal{P}(T)$, where $T_{\mathrm{D}}(d)$ denotes the set of time slots during which a teacher $d \in D$ is available.

□

Each lesson must be assigned to a room that is suitable. For that reason, we introduce the concept of room types, and an association between a room and a room type.

Definition 2.3 (Room input). The three-tuple $\mathcal{I}_{\mathrm{R}} = \langle R, Y, y_{\mathrm{R}} \rangle$ defines the *room input*, where

- R denotes a finite set of rooms,
- Y denotes a finite set of room types, and
- $y_{\mathrm{R}} : R \to Y$, where $y_{\mathrm{R}}(r)$ denotes the room type for a room $r \in R$.

□

Next, we introduce all data regarding subjects. Two other concepts are related to subjects: levels and subject groups. We model two conditions as constraints for each subject. These conditions are the following:

- no more than a given number of lessons per subject group is taught on the same day and

- all subject groups of the same subject have approximately the same size.

We also introduce the upper bounds that are input for these constraints here.

Definition 2.4 (Subject input). The seven-tuple $\mathcal{I}_{\mathrm{V}} = \langle V, A, K, v_{\mathrm{K}}, a_{\mathrm{V}}, \overline{L_{\mathrm{K}}}, \overline{S_{\mathrm{V}}} \rangle$ defines the *subject input*, where

- V denotes a finite set of subjects,
- A denotes a finite set of levels,
- K denotes a finite set of subject groups,
- $v_{\mathrm{K}} : K \to V$, where $v_{\mathrm{K}}(k)$ denotes the subject of a subject group $k \in K$,
- $a_{\mathrm{V}} : V \to A$, where $a_{\mathrm{V}}(v)$ denotes the level of a subject $v \in V$,
- $\overline{L_{\mathrm{K}}} : K \to \mathbb{Z}^+$, where $\overline{L_{\mathrm{K}}}(k)$ denotes the upper bound on the number of lessons of a subject group $k \in K$ that can be taught on the same day, and
- $\overline{S_{\mathrm{V}}} : V \to \mathbb{Z}^+$, where $\overline{S_{\mathrm{V}}}(v)$ denotes the maximal number of students that can be assigned to any subject group of a subject $v \in V$.

□

We model *student group binding* by introducing *student groups*. Each student group represents a set of students with exactly the same curriculum. All students in a student group attend all lessons together. There are two extremes:

- each student group consists of only one student and
- each student group contains all students that have exactly the same curriculum.

From now on, we formulate all decisions and constraints in terms of student groups, and not in terms of individual students. In this way, we reduce the number of decision variables and constraints that must be verified. But this way of modeling student group binding has some drawbacks. First, we introduce a new reason for the problem's inherent complexity, see Section 3.9. This drawback is related to the subject group size constraints, which we will define on page 20. The other drawback is that we possibly exclude assignments of individual students to subject groups that may be acceptable assignments in practice.

Definition 2.5 (Student group input). The four-tuple $\mathcal{I}_{\mathrm{G}} = \langle G, V_{\mathrm{G}}, a_{\mathrm{G}}, z_{\mathrm{G}} \rangle$ defines the *student group input*, where

- G denotes a finite set of student groups,
- $V_{\mathrm{G}} : G \to \mathcal{P}(V)$, where $V_{\mathrm{G}}(g)$ denotes the set of subjects chosen by every student belonging to a student group $g \in G$,
- $a_{\mathrm{G}} : G \to A$, where $a_{\mathrm{G}}(g)$ denotes the level of every student that belongs to a student group $g \in G$, and
- $z_{\mathrm{G}} : G \to \mathbb{Z}^+$, where $z_{\mathrm{G}}(g)$ denotes the number of students in a student group $g \in G$.

□

Finally, we define input for lessons. Note that each lesson must be taught in a room that is suitable, and that several room types can be suitable for a specific lesson. Each lesson has unit time length. We model block lessons of size two by a predicate that returns true if, and only if, two lessons form a block lesson and must be assigned to block times.

Definition 2.6 (Lesson input). The five-tuple $\mathcal{I}_{\mathrm{L}} = \langle L, Y_{\mathrm{L}}, d_{\mathrm{L}}, k_{\mathrm{L}}, \mathsf{Block} \rangle$ defines the *lesson input*, where

- L denotes a finite set of lessons,
- $Y_{\mathrm{L}} : L \to \mathcal{P}(Y)$, where $Y_{\mathrm{L}}(l)$ denotes the set of room types that are suitable for teaching a lesson $l \in L$,
- $d_{\mathrm{L}} : L \to D$, where $d_{\mathrm{L}}(l)$ denotes the teacher of a lesson $l \in L$, and
- $k_{\mathrm{L}} : L \to K$, where $k_{\mathrm{L}}(l)$ denotes the subject group of a lesson $l \in L$,
- $\mathsf{Block} : L \times L \to \mathbb{B}$, where $\mathsf{Block}(l_1, l_2)$ indicates whether two lessons $l_1, l_2 \in L$ form a block.

The Block relation is symmetric but not reflexive: for all lessons $l, l_1, l_2 \in L$ $\mathsf{Block}(l_1, l_2) \Leftrightarrow \mathsf{Block}(l_2, l_1)$ and $\mathsf{Block}(l, l) \Leftrightarrow$ false holds. Moreover, any lesson can form a block with at most one other lesson. That is, for all lessons $l, l_1, l_2 \in L$ $\mathsf{Block}(l, l_1) \wedge \mathsf{Block}(l, l_2) \Rightarrow l_1 = l_2$ holds. □

Now we can define the input of TCP as the six-tuple

$$\langle \mathcal{I}_{\mathrm{T}}, \mathcal{I}_{\mathrm{D}}, \mathcal{I}_{\mathrm{R}}, \mathcal{I}_{\mathrm{V}}, \mathcal{I}_{\mathrm{G}}, \mathcal{I}_{\mathrm{L}} \rangle.$$

2.2.3 Decision variables

In this section, we show that a solution to TCP consists of three assignments: the subject group assignment, the time slot assignment, and the room assignment. Below, we define each assignment formally.

Each student group must be assigned to a subject group for every subject in the student group's curriculum. Therefore, to simplify the definition of the subject group assignment, we first define which (student group, subject) pairs are relevant.

Definition 2.7 ((Student group, subject) pairs). Let P be the set of all relevant *(student group, subject)-pairs*, or *pairs* for short. That is,

$$P := \{ (g, v) \in G \times V \mid g \in G \wedge v \in V_{\mathrm{G}}(g) \}.$$

□

By g_{P} and v_{P}, we denote the pair's student group and subject, respectively.
Now a subject group assignment contains assignments of (student group, subject) pairs $(g, v) \in P$ to one of the possible subject groups for subject v.

Definition 2.8 (Subject group assignment). Let $K_V : V \to \mathcal{P}(K)$ return the set of possible subject groups for every subject, i.e., for $v \in V$, $K_V(v) := \{ k \in K \mid v_K(k) = v \}$ is the set of subject groups for a subject v. The *subject group assignment* consists of assignments of student groups to one of the possible subject groups for subjects in the student group's curriculum. That is, the subject group assignment is a mapping $k_P : P \to K$, such that $\forall_{(g,v)\in P} \quad k_P(g,v) \in K_V(v)$. □

The other two decisions both assign lessons to resources, in this case: time slots and rooms. Recall that each lesson has unit time length and that we model the assignment of block lessons to two consecutive time slots by means of a constraint.

Definition 2.9 (Time slot assignment). The *time slot assignment* contains assignments of lessons to a time slot: $t_L : L \to T$. □

Definition 2.10 (Room assignment). The *room assignment* consists of assignments of lessons to a room. Therefore, the room assignment is a function $r_L : L \to R$. □

Now, a solution to TCP is a three-tuple $\langle k_P, t_L, r_L \rangle$, i.e., a subject group assignment, a time slot assignment, and a room assignment.

2.2.4 Constraints

Not all solutions are acceptable or, in our definition, feasible. A solution $\langle k_P, t_L, r_L \rangle$ is feasible if, and only if, it satisfies all constraints. In this subsection, we define all constraints that must be satisfied.

The educational constraint forbids that more than a given number of lessons of a subject group are assigned to the same day.

Definition 2.11 (Educational constraints). Let $K_V : V \to \mathcal{P}(K)$ return the set of all possible subject groups for every subject. For a given time slot assignment t_L, let $L_{K,W} : K \times W \to \mathcal{P}(L)$ return the set of all lessons of a subject group assigned to a time slot on a specific day, i.e.,

$$L_{K,W}(k,w) := \{ l \in L \mid k_L(l) = k \ \wedge \ w_T(t_L(l)) = w \}.$$

If the maximal number of lessons of a subject group k on each day is denoted by $\overline{L_K}$, then the *educational constraints* are as follows: for $k \in K$ and $w \in W$, $|L_{K,W}(k,w)| \leq \overline{L_K}(k)$ must hold. □

Two lessons that share a resource may not be assigned to the same time slot. These resource constraints can be grouped according to the resource involved: rooms, student groups, subject groups, and teachers. We first define the general resource constraint type. Then the other four are expressed by means of this constraint type.

Definition 2.12 (Resource constraints). Let t_L be a given time slot assignment. Furthermore, let $L' \subseteq L$ be a set of lessons that share a common resource. Then the *resource constraint* is satisfied if, and only if, $l_1 \neq l_2 \ \Rightarrow \ t_L(l_1) \neq t_L(l_2)$ holds for all

lessons $l_1, l_2 \in L'$. In the remainder of this thesis, we shorten this boolean expression to $\mathsf{Resource}(L')$. □

As stated before, we have four different resource constraint types. The room constraints forbid that two lessons that use the same room are assigned to the same time slot.

Definition 2.13 (Room constraints). Let r_{L} be a given room assignment. We can determine the set of lessons that have been assigned to the same room. Formally, $L_{\mathrm{R}} : R \to \mathcal{P}(L)$ denotes this mapping and it returns for every room $r \in R$ the set $L_{\mathrm{R}}(r) := \{ l \in L \mid r_{\mathrm{L}}(l) = r \}$. For a given time slot assignment and room assignment, the *room constraints* are satisfied if $\mathsf{Resource}(L_{\mathrm{R}}(r))$ holds for every room $r \in R$. □

Two lessons cannot be taught at the same time if a student group has to attend both of them. Assigning student groups to lessons is a decision in our model. Therefore, not all resource constraints for student groups are known in advance. But some of these constraints are. Recall that part of the student group binding is achieved by assigning student groups to subject groups instead of lessons. Therefore, two lessons of the same subject group cannot be taught at the same time.

Definition 2.14 (Subject group constraints). From a given lesson input set $\mathcal{I}_{\mathrm{L}}$, we can determine the set of lessons of each subject group $k \in K$. We denote this mapping by $L_{\mathrm{K}} : K \to \mathcal{P}(L)$, where $L_{\mathrm{K}}(k) := \{ l \in L \mid k_{\mathrm{L}}(l) = k \}$. The time slot assignment satisfies the *subject group constraints* if, and only if, $\mathsf{Resource}(L_{\mathrm{K}}(k))$ holds for all subject groups $k \in K$. □

To simplify the definition of all student group constraints, we first use the subject group assignment to derive the set of lessons to which each student group has been assigned. No two lessons in these sets may be assigned to the same time slot. Note that if the subject group assignment is complete, i.e., all pairs $(g, v) \in P$ have been assigned, then all subject group constraints are implied by the student group constraints.

Definition 2.15 (Student group constraints). If a (student group, subject) pair $(g, v) \in P$ has been assigned to subject group $k = k_{\mathrm{P}}(g, v)$, it implies that student group g has to attend every lesson in $L_{\mathrm{K}}(k)$. Let k_{P} be a given subject group assignment. Now, let $K_{\mathrm{G}}(g)$ be the set of subject groups to which student group g has been assigned, i.e.,

$$K_{\mathrm{G}}(g) := \{ k \in K \mid \exists_{v \in V_{\mathrm{G}}(g)} k_{\mathrm{P}}(g, v) = k \}.$$

Then, for every student group $g \in G$ the *student group lesson set* is the set of lessons to which g has been assigned: $L_{\mathrm{G}}(g) := \cup_{k \in K_{\mathrm{G}}(g)} L_{\mathrm{K}}(k)$. The *student group constraints* forbid that two lessons for the same student group are assigned to the same time slot. Formally, $\mathsf{Resource}(L_{\mathrm{G}}(g))$ must be satisfied for any student group $g \in G$. □

In a similar way, we define the teacher constraints. Since the teacher assignment is given in the lesson input, the teacher lesson sets are also input of the problem.

Definition 2.16 (Teacher constraints). For every lesson $l \in L$, we are given $d_{\mathrm{L}}(l)$, the teacher of l. Then $L_{\mathrm{D}} : D \to \mathcal{P}(L)$ maps every teacher onto the set of lessons that is taught by that teacher. That is, for every teacher $d \in D$, the *teacher lesson set* $L_{\mathrm{D}}(d) := \{\, l \in L \mid d_{\mathrm{L}}(l) = d \,\}$ is the collection of all lessons that teacher d has to teach. A time slot assignment satisfies the *teacher constraints* if, and only if, $\mathsf{Resource}(L_{\mathrm{D}}(d))$ holds for every teacher $d \in D$. □

We also want to forbid lessons to be assigned to time slots during which the teacher is unavailable. These requirements for part-time teachers are modeled by the teacher availability constraints.

Definition 2.17 (Teacher availability constraints). Let t_{L} be a time slot assignment. Then the time slot assignment satisfies the *teacher availability constraints* if for each teacher $d \in D$ and for every lesson $l \in L_{\mathrm{D}}(d)$ of this teacher $t_{\mathrm{L}}(l) \in T_{\mathrm{D}}(d)$ holds. □

We model the room compatibility requirement by constraints that exclude assignments of lessons to a room of an incompatible type.

Definition 2.18 (Room compatibility constraints). Let r_{L} be a given room assignment. For any room $r \in R$, $y_{\mathrm{R}}(r)$ denotes the room type. Furthermore, recall that for lesson $l \in L$, $Y_{\mathrm{L}}(l)$ is the set of compatible room types for l. Then the *room compatibility constraints* are satisfied if $y_{\mathrm{R}}(r_{\mathrm{L}}(l)) \in Y_{\mathrm{L}}(l)$ holds for every lesson $l \in L$. □

We want subject groups of the same subject to be balanced. That is, all subject groups $k \in K_{\mathrm{V}}(v)$ for a subject $v \in V$ must have about the same number of students. We can model this requirement as a hard constraint for each subject by requiring that

- the difference between the largest and smallest subject group is at most a given number of students,
- each subject group has *at least* a given number of students, or
- each subject group has *at most* a given number of students.

We choose to bound the subject group sizes for each subject from above. We model this requirement by means of the subject group size constraints.

Definition 2.19 (Subject group size constraints). Let k_{P} be a given subject group assignment. For every subject group $k \in K$ and corresponding subject $v = v_{\mathrm{K}}(k)$, the set of student groups assigned to k is $G_{\mathrm{K}}(k) := \{\, g \in G \mid v_{\mathrm{K}}(k) \in V_{\mathrm{G}}(g) \ \wedge\ k_{\mathrm{P}}(g, v_{\mathrm{K}}(k)) = k \,\}$.

Recall that $z_{\mathrm{G}}(g)$ denotes the size of student group $g \in G$. A subject group assignment k_{P} satisfies the *subject group size constraints* if for every subject $v \in V$ and

every subject group $k \in K_V(v)$ the total number of students assigned to k is less than or equal to the given upper bound, i.e., $\sum_{g \in G_K(k)} z_G(g) \leq \overline{S_V}(v)$. □

Finally, we impose restrictions on the time slot and room assignment for block lessons. Two lessons that form a block must be assigned to consecutive time slots that are not separated by a break, i.e., block times, and to the same room. The block time constraints and room assignment for blocks constraints exclude solutions that violate these conditions.

Definition 2.20 (Block time constraints). The time slot assignment t_L satisfies the *block time constraints* if, and only if,

$$\mathsf{Block}(l_1, l_2) \Rightarrow \mathsf{BlockTime}(t_L(l_1), t_L(l_2))$$

holds for any pair of lessons $l_1, l_2 \in L$. □

Definition 2.21 (Room assignment for blocks constraints). A given room assignment r_L satisfies all *room assignment for blocks constraints* if, and only if, $\mathsf{Block}(l_1, l_2) \Rightarrow r_L(l_1) = r_L(l_2)$ holds for all lessons $l_1, l_2 \in L$. □

This concludes the definition of all constraints. Note that we have modeled all desires as hard constraints

2.2.5 Problem definition

Definition 2.22 (Timetable construction problem). An instance of the TIMETABLE CONSTRUCTION PROBLEM (TCP) consists of time input, teacher input, room input, subject input, student group input, and lesson input. Formally, it is a six-tuple $\langle \mathcal{I}_T, \mathcal{I}_D, \mathcal{I}_R, \mathcal{I}_V, \mathcal{I}_G, \mathcal{I}_L \rangle$. A solution of TCP, $\langle k_P, t_L, r_L \rangle$, consists of a subject group assignment, time slot assignment, and room assignment respectively. The objective of TCP is to find a feasible solution, i.e., a solution that satisfies all educational, resource, teacher availability, room compatibility, subject group size, block time, and room assignment for blocks constraints as defined in Subsection 2.2.4. The resource constraints force no conflicts for any room, student group, subject group, and teacher. □

2.3 Shorthand for sub-problems

In the following chapter, we discuss the computational complexity of sub-problems of TCP. Formulating the problem definitions like we did in the previous section is cumbersome. Moreover, by writing out the problem types in a compact way, e.g., by only focussing on the relevant decisions and constraint types, we can exhibit the differences between two related sub-problems more easily. Therefore, we introduce a shorthand for sub-problems.

A *sub-problem* of TCP is a problem that follows from the definition of TCP by carrying out at least one of the following actions:

- restricting the input,
- ignoring all decision variables of one or more types,
- fixing the values of decision variables of a certain type, or
- ignoring one or more constraint types.

This is reflected in the way we abbreviate the sub-problems under consideration. Each sub-problem of TCP can be defined by means of the three-field notation $[\ \alpha\ \|\ \beta\ \|\ \gamma\ ]$, where α denotes the restrictions on the input, β denotes all relevant decision variables, and γ gives all constraint types that are taken into account. Below we describe the possible values for the three fields, which we call α-field, β-field, and γ-field.

α-field. There are several ways how we may want to restrict the input. First, we may want to limit the sizes of one or more sets. By writing $|S| \leq n$ or $|S| = n$ in the α-field, we show that we only consider instances for which the set S has at most n elements or exactly n elements, respectively. We can generalize this by writing $|S| \in N$, stating that the cardinality of S must be in the set $N \subseteq \mathbb{N}$.

Second, our focus may be on instances for which the upper bounds $\overline{L_{\mathrm{K}}}$, $\overline{S_{\mathrm{V}}}$, or the student group sizes z_{G} are given. In that case, we use a straightforward notation as well, e.g., $\overline{L_{\mathrm{K}}} = m$. If no restrictions are put on the instance, we write $\circ$ in the α-field.

We can also have more complex restrictions. These are usually combinations of several cardinality restrictions or given upper bounds. In that case we introduce a predicate in the text and use the predicate name in the field. An example is the class binding restriction, see page 35.

β-field. We consider three decision types: room assignment, r_{L}, subject group assignment, k_{P}, and time slot asignment, t_{L}. Each assignment type that is of interest, i.e., not trivial and not fixed in advance, is mentioned in the β-field. If an assignment is given and fixed, we mention that in the γ-field.

γ-field. Constraint types determine which assignments are feasible and which are not. Restrictions on the feasibility of solutions that are of interest are stated in the γ-field. There are ten constraint types in TCP which can appear in the γ-field. Instead of writing out the constraint type completely, we use abbreviations as defined in Table 2.1.

If for a sub-problem some assignment is given and must be fixed, then we mention this in the γ-field as well. We assume that $\boldsymbol{r}$, $\boldsymbol{k}$, and $\boldsymbol{t}$ are a fixed room assignment, subject group assignment, and time slot assignment, respectively. Then fixing the time slot assignment is abbreviated to $t_{\mathrm{L}} = \boldsymbol{t}$.

Example 2.23. The problem $[\ \circ\ \|\ k_{\mathrm{P}}\ \|\ ss\ ]$ is a sub-problem in which we focus on the subject group assignment decision, k_{P}, and the subject group size constraints only. Formally, we want to find a subject group assignment $k_{\mathrm{P}} : P \to K$ that satisfies

constraint type	abbreviation	see page
block time	*bt*	21
educational	*ed*	18
room	*ro*	19
room assignment for blocks	*rb*	21
room compatibility	*rc*	20
student group	*st*	19
subject group	*su*	19
subject group size	*ss*	20
teacher	*te*	20
teacher availability	*ta*	20

Table 2.1. Abbreviations for constraint types.

the constraints *ss*:

$$\forall_{v \in V} \forall_{k \in K_{\mathrm{V}}(v)} \quad \sum_{g \in G_{\mathrm{K}}(k)} z_{\mathrm{G}}(g) \leq \overline{S_{\mathrm{V}}}(v).$$

We assume all relevant input is given. In this case, the sets G, V, K, the mappings $v_{\mathrm{K}} : K \to V$, $z_{\mathrm{G}} : G \to \mathbb{Z}^+$, and the upper bound $\overline{S_{\mathrm{V}}} : V \to \mathbb{Z}^+$ are given. Later we show that BIN PACKING is polynomially reducible to $[\circ \parallel k_{\mathrm{P}} \parallel ss]$. □

Example 2.24. Assume that we only look at a small part of the timetable, namely two consecutive time slots. So, $|T| = 2$. Moreover, assume that we have assigned all lessons to one of these two time slots. That is, the time slot assignment is given and fixed: $t_{\mathrm{L}} = \boldsymbol{t}$. Now, suppose that we want to find a room assignment for single and block lessons while taking room types into account. This sub-problem can be modeled as $[\, |T| = 2 \parallel r_{\mathrm{L}} \parallel ro, rc, rb, t_{\mathrm{L}} = \boldsymbol{t}\,]$, as the objective is to find a room assignment $r_{\mathrm{L}} : L \to R$ such that

- no two lessons taught at the same time are assigned to the same room (*ro*):

$$\forall_{r \in R} \quad \mathsf{Resource}(L_{\mathrm{R}}(r)),$$

- each lesson is taught in a suitable room (*rc*):

$$\forall_{l \in L} \quad y_{\mathrm{R}}(r_{\mathrm{L}}(l)) \in Y_{\mathrm{L}}(l),$$

- and both lessons of a block lesson are assigned to the same room (*rb*):

$$\forall_{l_1, l_2 \in L} \quad \mathsf{Block}(l_1, l_2) \Rightarrow r_{\mathrm{L}}(l_1) = r_{\mathrm{L}}(l_2).$$

Carter & Tovey [1992] show that $[\, |T| = 2 \parallel r_{\mathrm{L}} \parallel ro, rc, rb, t_{\mathrm{L}} = \boldsymbol{t}\,]$ is NP-complete. The variant without blocks $[\, |T| = 2 \parallel r_{\mathrm{L}} \parallel ro, rc, t_{\mathrm{L}} = \boldsymbol{t}\,]$ is polynomially solvable. We describe the details of these results in Chapter 3. □

2.4 Summary

In this chapter, we defined the timetable construction problem we investigate further in this thesis. This problem was called the TIMETABLE CONSTRUCTION PROBLEM (TCP) and covers most characteristics of timetable construction problems at Dutch secondary schools. We considered group binding, different room types, and block lessons that occupy two time slots. Student group binding was achieved by determining student groups, i.e., groups of students with the same curriculum. Instead of considering students individually, we defined all constraints and decisions in terms of these student groups. Idle time and rooms at different locations were not taken into account.

TCP comprises three kinds of decisions. First, for every subject, all students have to be sectioned into groups of almost equal size. These groups are called subject groups and subject group assignment is the name of the corresponding decision. Second, we have to determine a timetable. Since a timetable is an assignment of lessons to time slots, we call this decision time slot assignment. Third, the model of TCP comprises room assignment.

A solution is feasible if it satisfies all constraints. In contrast to many other timetabling models, we do not have a cost measure that values the acceptance of a timetable. We consider ten different types of constraints that *must be* satisfied.

- A timetable must be conflict-free: no room, student group, subject group, or teacher may have two lessons at the same time. We refer to these resource constraints by the names room, student group, subject group, and teacher constraints, respectively.
- A lesson can only be taught at a time at which the teacher is available.
- From an educational point of view, it is required to spread lessons in the same subject over different days. We defined educational constraints for each subject, such that no more than a given number of lessons will be on the same day.
- Subject groups of the same subject must have about the same number of students. Therefore, we bounded the subject group sizes by given maxima.
- A lesson can only be taught in a room that is suitable. Room compatibility constraints force that rooms and lessons match.
- Finally, block lessons must be taught at consecutive time slots, may not be separated by a break, and must be assigned to the same room. By means of block time constraints and room assignment for blocks constraints, we modeled the requirements for the time slot assignment and the room assignment, respectively.

All constraints were summarized in Table 2.1, together with their abbreviations. In the following chapter, we use these abbrevations in a three-field notation for subproblems of TCP.

3

Computational complexity

In the previous chapter we have formulated TCP, the timetable construction problem we consider. Timetable planners are interested in algorithms that return feasible solutions to instances of this problem. Finding such a solution or even showing that there exists one is not a trivial task. Complexity theory provides evidence why solving these problems may take unacceptably long running times.

In this chapter we show that we cannot expect to develop fast algorithms for solving TCP. By proving that TCP belongs to the class of hard problems, i.e., by showing that its decision variant TC is NP-complete, we make this expectation explicit. We give several independent reasons why TC is NP-complete. In addition, we identify hard-to-solve special cases of TC that become easy after removal of one constraint type. In our results, each constraint type of TC causes the difference between easy and hard at least once.

The organization of this chapter is as follows. We start by recalling some concepts from complexity theory in Section 3.1. For more details, the reader is referred to Garey & Johnson [1979]. Proving that a particular timetabling problem is hard (easy) to solve is usually done by transforming from (to) a well-known hard (easy) problem. Therefore, we present well-known problems that we use together with their complexity status in Section 3.2. In Section 3.3, we prove that a certificate of TC can be verified in polynomial time, implying that TC $\in \mathcal{NP}$. Next, in Sections 3.4 to 3.9 we settle the complexity of a number of sub-problems that are special cases of TC by

focusing on several decisions and constraint types. We conclude with an overview of all results in Section 3.10.

3.1 Complexity theory

In this chapter, we distinguish two kinds of combinatorial problems: feasibility problems, like the timetable construction problem TCP we defined in the previous chapter, and decision problems that are related to feasibility problems. We define a feasibility problem by describing its input, decision variables, and set of constraints that must be met. The objective is to find a feasible solution, i.e., an assignment of values to all decision variables such that all constraints are satisfied. An *instance* is specified by the problem's input. For the decision variant of a feasibility problem, we are only interested in the answer to the question whether a feasible solution exists for a specific instance. We distinguish "yes instances", i.e., instances for which such an assignment exists, from "no instances".

In order to discuss the computational complexity of combinatorial problems and algorithms, we introduce the notions of *instance size* and *time complexity*. The size of an instance is defined as the number of symbols required to represent the instance in a compact way. The time complexity of a given algorithm defines for each possible instance size the maximum amount of time needed for solving instances of that size.

We say that an algorithm solves an instance of a feasibility problem if it returns a correct answer for this instance. That is, an algorithm that solves the problem must return a feasible solution if, and only if, such a solution exists. We say that an algorithm solves a problem if it solves all instances of that problem.

An algorithm is called *polynomial-time* if, and only if, there exists a polynomial function that bounds the time complexity of the algorithm. Otherwise, the algorithm is called *superpolynomial-time*. A problem is said to be polynomially solvable if, and only if, there exists a polynomial algorithm that solves the problem. Informally, problems that are polynomially solvable are called easy, and problems for which no polynomial-time algorithm that solves them is known are called hard.

The class $\mathcal{P}$ consists of all decision problems that are polynomially solvable. The class $\mathcal{NP}$ is the set of decision problems for which each yes instance has a *compact certificate*, which is an amount of data with a size that is polynomial in the instance size, such that it is possible to verify in polynomial time that an instance is a yes instance. A feasible solution of the feasibility problem is an example of a compact certificate for the decision variant. It has been proven that $\mathcal{P} \subseteq \mathcal{NP}$ holds.

To relate the computational complexity of two decision problems we use the concept of polynomial reducibility.

Definition 3.1. Let Π and Π' be two problems in $\mathcal{NP}$. The problem Π is *polynomially reducible* to Π' if, and only if, there exists an algorithm that maps instances i of problem Π into instances i' of problem Π', such that

- the algorithm is polynomial-time and
- i is a yes instance for Π if, and only if, i' is a yes instance for Π'.

If Π is polynomially reducible to Π', we write $\Pi \propto \Pi'$, and informally say that Π is a *special case* of Π'. □

We consider the class $\mathcal{NPC} \subseteq \mathcal{NP}$ of so-called NP-complete problems, which are the hardest ones in $\mathcal{NP}$. A problem is in $\mathcal{NPC}$ if, and only if, it is in $\mathcal{NP}$ and each problem in $\mathcal{NP}$ is polynomially reducible to it. It will be cumbersome to prove NP-completeness for all our problems in this way. However, the concept of polynomial reducibility is extremely useful when we have an NP-complete problem Π. To prove NP-completeness, it then suffices to show that the problem under consideration is in $\mathcal{NP}$ and that Π is polynomially reducible to it.

Note that if one NP-complete problem is polynomially solvable, then all problems in $\mathcal{NP}$ are polynomially solvable. This would imply that $\mathcal{NP} \subseteq \mathcal{P}$. However, it is generally believed that $\mathcal{NPC} \not\subseteq \mathcal{P}$ because nobody has succeeded in finding a polynomial-time algorithm for an NP-complete problem.

The computational complexity of feasibility problems and their decision variants are closely related. If a combinatorial decision problem is NP-complete, then the feasibility variant is called NP-hard. On the other hand, a decision problem is in $\mathcal{P}$ if we have a polynomial-time algorithm A that solves the feasibility variant. An algorithm that applies A and returns "yes" if, and only if, A has constructed a feasible solution solves the decision problem.

3.2 Complexity of basic combinatorial problems

Our proofs of computational complexity of sub-problems of TC are based on transformations from problems that are NP-complete or to problems that are polynomially solvable. Here, we give an overview of the polynomially solvable problems and NP-complete problems that we use in this respect. We refer to papers that give polynomial-time algorithms or proofs of NP-completeness.

3.2.1 Polynomially solvable problems

Several decisions in timetabling problems can be seen as assignments of resources to activities. The most common constraints are restrictions in the sense that a resource can be assigned to at most one event and not every resource can be assigned to every event. These problems are often modeled as matching problems for bipartite graphs. Here, the vertex sets represent the resources and events, respectively, and the edge set corresponds to possible assignments of resources to events.

Definition 3.2 (Bipartite matching problem). [Cook et al., 1998] An instance of the BIPARTITE MATCHING PROBLEM is a three-tuple $\langle \mathcal{V}, \mathcal{W}, \mathcal{E} \rangle$, where

- $\mathcal{V}$ and $\mathcal{W}$ are finite sets of vertices, often refered to as the left and right vertex set, respectively, and

- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{W}$ is a finite set of edges between vertices in $\mathcal{V}$ and $\mathcal{W}$.

A matching is a subset of edges $\mathcal{M} \subseteq \mathcal{E}$ such that no two edges in $\mathcal{M}$ are incident with the same vertex. The objective of the BIPARTITE MATCHING PROBLEM is to find a maximum matching, i.e., a matching of maximum cardinality. □

When we have to partition a set of items into subsets of items that are mutually independent, we can see it as finding a feasible coloring of all vertices of a graph. The items are represented by vertices, and two vertices are not connected if, and only if, they are independent. A coloring with $\mathcal{K}$ colors is a function $f : \mathcal{V} \to \{1, \ldots, \mathcal{K}\}$ that assigns one of the $\mathcal{K}$ colors to each vertex of the graph. A coloring is *proper* if $f(u) \neq f(v)$ holds whenever $\{u, v\} \in \mathcal{E}$. The *chromatic number* of a graph, χ, is the smallest number for which there exists a proper coloring.

In the next subsection, we will recall that graph coloring is hard for general graphs. For interval graphs, the coloring problem can be solved in polynomial time. A graph is said to be an *interval graph* if every vertex can be represented as a time interval and an edge exists between two vertices if, and only if, the corresponding time intervals overlap.

Definition 3.3 (Interval graph coloring problem). [Golumbic, 1980] An instance of the INTERVAL GRAPH COLORING PROBLEM is an interval graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The objective of the INTERVAL GRAPH COLORING PROBLEM is to determine the chromatic number of the interval graph $\mathcal{G}$. □

3.2.2 NP-complete problems

The most straightforward way to prove NP-completeness of a problem Π is to give a polynomial-time reduction from a problem $\Pi' \in \mathcal{NPC}$ to Π. This is only possible if there is one problem for which NP-completeness is proven according to the definition. This basic NP-complete problem is SATISFIABILITY. In this problem, we are given a set of boolean variables and a boolean formula, represented by a set of clauses. Each clause consists of a disjunction of literals; a literal is either a boolean variable or its negation. The boolean formula is satisfiable if there exists an assignment of each boolean variable to either true or false such that every clause evaluates to true.

Definition 3.4 (Satisfiability (SAT)). [Cook, 1971] An instance of SAT problem is a pair $\langle \mathcal{U}, \mathcal{C} \rangle$ where

- $\mathcal{U}$ is a set of variables and
- $\mathcal{C}$ is a collection of clauses over $\mathcal{U}$.

The objective of SAT is to decide whether there exists a satisfying truth assignment for $\mathcal{C}$. □

There are special cases of SAT that are already NP-complete. First, SAT is NP-complete if we restrict ourselves to clauses of exactly three literals.

Definition 3.5 (Three-Satisfiability (3SAT)). [Cook, 1971] An instance of 3SAT is a pair $\langle \mathcal{U}, \mathcal{C} \rangle$ where $\mathcal{U}$ is a set of variables and $\mathcal{C}$ is a collection of clauses over $\mathcal{U}$ such that each clause $c \in \mathcal{C}$ has exactly three literals. The objective of 3SAT is to decide whether there exists a satisfying truth assignment for $\mathcal{C}$. □

Moreover, if we also require that in each clause at least one literal evaluates to false, the problem remains hard to solve.

Definition 3.6 (Not-All-Equal-3SAT (NAE3SAT)). [Garey & Johnson, 1979] An instance of NAE3SAT is a pair $\langle \mathcal{U}, \mathcal{C} \rangle$ of variables $\mathcal{U}$ and three-literal clauses $\mathcal{C}$. The objective of NAE3SAT is to decide whether there exists values for the variables such that all clauses evaluate to true and not all literals in each clause have the same value. □

In the previous subsection, we have stated that it is easy to find a feasible coloring of vertices of an interval graph. If we have an arbitrary graph, the coloring problem turns out to be NP-complete.

Definition 3.7 (Graph K-Colorability). [Karp, 1972] An instance of GRAPH K-COLORABILITY problem is a pair $\langle \mathcal{G}, \mathcal{K} \rangle$ where

- $\mathcal{G}$ is a graph $(\mathcal{V}, \mathcal{E})$ and
- $\mathcal{K}$ is a positive integer.

The objective of GRAPH K-COLORABILITY is to decide whether $\mathcal{G}$ is $\mathcal{K}$-colorable, i.e., whether there exists a coloring $f : \mathcal{V} \to \{1, \ldots, \mathcal{K}\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in \mathcal{E}$. □

Graph coloring problems model partitioning sets of items into subsets of independent items. Sometimes we do not have a dependency between items, but we do have a size for each item. If we are given a maximal number of subsets and an upper bound on the total size of each subset, then we obtain a problem that is known as BIN PACKING.

Definition 3.8 (Bin Packing). [Garey & Johnson, 1979] An instance of BIN PACKING is a four-tuple $\langle \mathcal{U}, s, \mathcal{B}, \mathcal{K} \rangle$ where

- $\mathcal{U}$ is a finite set of items,
- s is a vector of size $|\mathcal{U}|$, with $s(u)$ is a positive integer denoting the size of item u,
- $\mathcal{B}$ is a positive integer, denoting the bin capacity, and
- $\mathcal{K}$ is a positive integer.

The objective of BIN PACKING is to decide whether there exists a partition of $\mathcal{U}$ into disjoint sets $\mathcal{U}_1, \ldots, \mathcal{U}_\mathcal{K}$ such that the sum of the sizes of the items in each bin $\mathcal{U}_i$ is $\mathcal{B}$ or less. □

Finally, we give an extension of the bipartite matching problem to matching in three dimensions.

Definition 3.9 (Three-Dimensional Matching (3DM)). [Karp, 1972] An instance of 3DM is a four-tuple $\langle \mathcal{W}, \mathcal{X}, \mathcal{Y}, \mathcal{M} \rangle$ where

- $\mathcal{W}$, $\mathcal{X}$, $\mathcal{Y}$ are disjoint sets having the same number q of elements and
- $\mathcal{M} \subseteq \mathcal{W} \times \mathcal{X} \times \mathcal{Y}$ is a set of triples.

The objective of 3DM is to decide whether $\mathcal{M}$ contains a matching $\mathcal{M}'$ of size q, i.e., a subset $\mathcal{M}' \subseteq \mathcal{M}$ such that $|\mathcal{M}'| = q$ and no two elements in $\mathcal{M}'$ agree in any coordinate. □

3.3 Complexity of sub-problems of Timetable Construction

In the following sections, we show that various sub-problems of TC are NP-complete. Below, we show that each sub-problem under consideration is a special case of TC. If TC is an element of $\mathcal{NP}$, then this implies that TC is also NP-complete.

We can check in polynomial time whether a given room, subject group, and time slot assignment satisfies all constraints of TC. Therefore, TC $\in \mathcal{NP}$. Consequently, for every sub-problem Π of TC, $\Pi \in \mathcal{NP}$.

We can show that every sub-problem Π of TC is a special case of TC by giving a transformation of every instance of Π to an instance of TC. The transformation must map yes (no) instances of Π onto yes (no) instances of TC. The sub-problems differ from TC in three possible ways: (1) there are restrictions on the input, (2) some constraints in TC are ignored in the sub-problem, and (3) one or more assignments are fixed. Some constraints of TC can simply be ignored by choosing the parameters such that any assignment satisfies the constraints. This holds for the block time (*bt*), educational (*ed*), room assignment for blocks (*rb*), room compatibility (*rc*), subject group size (*ss*), and teacher availability (*ta*) constraints.

Resource constraints that are not defined in the sub-problem can be ignored by introducing a dedicated resource for each lesson in TC. For example, if the teacher constraints (*te*) are ignored in the sub-problem, we can introduce a different teacher for each lesson in TC.

By introducing a dedicated teacher for each lesson in TC and restricting the availability of this teacher to a single time slot, we can take a given fixed time slot assignment into account. We must introduce new subjects, subject groups, lessons, and student groups in TC to transform a fixed subject group assignment. Here, each subject must have only one subject group.

By using these transformation rules, we can reduce instances of all our sub-problems to instances of TC in polynomial time. Therefore, the sub-problems we consider are all special cases of TC.

Each of the sub-problems that we show to be NP-complete focuses on different characteristics of TC. We thereby give several independent reasons why TC is NP-complete. Apart from the resource constraints, every characteristic of TC occurs as the characteristic that makes the problem hard.

In Sections 3.4 to 3.9, we focus on one sub-problem at a time. We begin each section with the description of the sub-problem under consideration. Then, we show that removal of one characteristic leaves us with a polynomially solvable problem. Finally, we prove that the sub-problem under consideration is NP-complete. We always consider the *decision variant* of the sub-problem. Sometimes, we prove polynomial solvability of a sub-problem by giving a polynomial-time algorithm for the feasibility variant. These algorithms can easily be transformed into polynomial-time algorithms for the decision variant.

3.4 Time slot assignment for classes and teacher availabilities

First, we show that the presence of teacher availabilities may turn an easy sub-problem into a hard one. Suppose we are only interested in teachers. That is, we do not consider student groups, their curricula, and the possible ways to assign student groups to subject groups. However, we want to find an assignment of each lesson to a time slot such that no two lessons of a teacher or a subject group conflict and the lesson's teacher is available at the time the lesson is given.

If each teacher is always available, the problem can be modeled as the CLASS-TEACHER problem. This well-known timetabling problem was first introduced by Gotlieb [1963].

Definition 3.10 (Class-teacher). An instance of the CLASS-TEACHER problem is a four-tuple $\langle m, n, p, r \rangle$ where

- m is the number of classes,
- n is the number of teachers,
- p is the number of time slots, and
- r is an $m \times n$ matrix, with r_{cd} denoting the number of lessons taught by teacher d to class c.

The time slot assignment is modeled by a set of variables $x_{cdt} \in \{0,1\}$ (for $c = 1,\ldots,m$, $d = 1,\ldots,n$, $t = 1,\ldots,p$), with $x_{cdt} = 1$ if a lesson for class c taught by teacher d is taught at time slot t and 0 otherwise. The objective is to find a time slot assignment satisfying

$$\begin{array}{ll} \forall_{c=1,\ldots,m}\forall_{d=1,\ldots,n} & \sum_{t=1}^{p} x_{cdt} = r_{cd}, \\ \forall_{c=1,\ldots,m}\forall_{t=1,\ldots,p} & \sum_{d=1}^{n} x_{cdt} \leq 1, \\ \forall_{d=1,\ldots,n}\forall_{t=1,\ldots,p} & \sum_{c=1}^{m} x_{cdt} \leq 1. \end{array}$$

□

The transformation from $[\circ \parallel t_{\mathrm{L}} \parallel su, te]$ to CLASS-TEACHER is straightforward: for each subject group, we introduce a class in the CLASS-TEACHER problem.

Corollary 3.11. $[\circ \parallel t_{\mathrm{L}} \parallel su, te] \propto$ CLASS-TEACHER. □

De Werra [1971] and Even, Itai & Shamir [1976] model CLASS-TEACHER as the problem of finding matchings in bipartite multigraphs. Even et al. [1976] focus on determining perfect matchings, whereas De Werra [1971] uses a network flow model. The left and right set of vertices represent the sets of classes and teachers, respectively. Each lesson between class c and teacher d is modeled by an edge (c,d) in the bipartite multigraph. There may be parallel edges.

Typically, teachers and classes have different numbers of lessons. Therefore, the degrees of vertices are not equal. Let δ be the maximum degree of any vertex. We obtain a regular bipartite multigraph of degree δ by introducing dummy vertices in the left and right vertex set and drawing the missing edges from the dummy vertices to vertices with degree less than δ in the original graph. By adding the right number of dummy vertices in the left and right vertex sets and adding edges between dummy vertices, we can extend the edge set such that each dummy vertex has degree δ as well.

Now, a perfect matching corresponds to a conflict-free set of lessons for all classes and teachers that can be assigned to some time slot. If the class (teacher) vertex is matched to a dummy vertex, then this class (teacher) is idle. Finding the time slot assignment x_{cdt} corresponds to determining δ edge-disjoint perfect matchings.

Theorem 3.12. [De Werra, 1971] CLASS-TEACHER *is polynomially solvable.*

Proof. We present the algorithm by Even et al. [1976]. There exists a feasible solution to CLASS-TEACHER if there are δ edge-disjoint perfect matchings. The algorithm repeatedly determines maximum matchings in a bipartite multigraph. If a perfect matching exists, then an algorithm for determining maximum matchings finds it. In this case, all edges of the perfect matching found are removed and the algorithm continues on the remaining bipartite multigraph. Execution halts when an empty graph remains or not all vertices are matched in the maximum matching that is found. In the first case, problem is solved, and in the latter case, no feasible solution exists. A polynomial-time algorithm based on network flows is given by De Werra [1971]. □

Corollary 3.13. $[\circ \parallel t_{\mathrm{L}} \parallel su, te]$ *is polynomially solvable.* □

Adding availabilities to the CLASS-TEACHER model can be done in a straightforward way.

Definition 3.14 (Class-teacher with teacher availabilities (CTTA)). An instance of CTTA is a five-tuple $\langle m,n,p,r,a\rangle$ where m, n, p, and r are as defined in CLASS-TEACHER and a is an $n \times p$ matrix, with $a_{dt} = 1$ if, and only if, teacher d is available for time slot t and 0 otherwise. Again, the objective is to find a time slot assignment $x_{cdt} \in \{0,1\}$ satisfying all constraints of CLASS-TEACHER and the availability constraints, i.e., $\forall_{d=1,\ldots,n}\forall_{t=1,\ldots,p}\, x_{cdt} \leq a_{dt}$. □

Now, consider the case that there are three time slots only. Moreover, we have two kinds of teachers: 2-teachers that teach two lessons and 3-teachers that teach three lessons. Finally, we restrict the instances to teachers that are *tight*. That is, the number of time slots at which each teacher is available equals the number of lessons that the teacher will teach. Even et al. [1976] prove that this special case of CTTA, which they call RESTRICTED TIMETABLING, is NP-complete.

Theorem 3.15. [Even et al., 1976] CTTA *is NP-complete, even for three time slots and tight teachers.*

Proof. 3SAT is polynomially reducible to RESTRICTED TIMETABLING. This problem is equivalent to CTTA with three time slots and tight teachers. □

We can transform each instance of CTTA to an instance of $[\circ \parallel t_{\mathrm{L}} \parallel su, te, ta]$. Because lessons of a subject group can be taught by more than one teacher, the subject groups play the role of classes. Since this transformation is a polynomial-time reduction, it gives us an NP-completeness result for $[\circ \parallel t_{\mathrm{L}} \parallel su, te, ta]$.

Theorem 3.16. CTTA $\propto [\circ \parallel t_{\mathrm{L}} \parallel su, te, ta]$. □

Corollary 3.17. $[\circ \parallel t_{\mathrm{L}} \parallel su, te, ta]$ *is NP-complete.* □

There is a special case of $[\circ \parallel t_{\mathrm{L}} \parallel su, te, ta]$ that is easy. If there are only two time slots, then the remaining problem is polynomially solvable. That is, $[|T_{\mathrm{D}}(d)| = 2 \parallel t_{\mathrm{L}} \parallel su, te, ta]$ is polynomially solvable. Even et al. [1976] give a polynomial-time algorithm for this problem.

To conclude, we give a schematic overview of the most important complexity results presented in this section in Figure 3.1.

3.5 Time slot assignment and different curricula

At Dutch secondary schools, students in upper years have some freedom regarding their program. Due to this freedom, there can be a variety of different curricula. This implies that subject groups represent many different collections of student groups. And, consequently, many pairs of lessons are conflicting and should be assigned to different time slots. We show that this characteristic is another source of complexity.

The sub-problem we consider has a subject group assignment as input. We try to find a time slot assignment that respects the subject group and student group constraints. That is, no two lessons of the same subject group or student group can be taught at the same time.

If we do not take student group constraints into account, then the remaining sub-problem is easy. There is a straightforward transformation of an instance of $[\circ \parallel t_{\mathrm{L}} \parallel su, k_{\mathrm{P}} = \boldsymbol{k}]$ to a very special type of instance of GRAPH K-COLORABILITY: introduce a vertex for every lesson and an edge between lessons $\{l_1, l_2\}$ for any two lessons $l_1, l_2 \in L$ of the same subject group. For this transformation, all vertices of

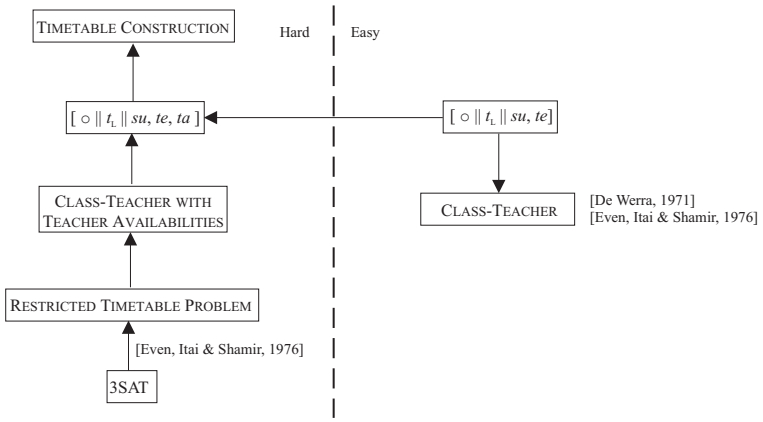


Figure 3.1. Complexity results for time slot assignment with teacher availabilities. We make a distinction between easy and hard problems. That is, we separate polynomially solvable problems from those that are NP-complete. The arcs between problems indicate polynomial reducibility.

the same subject group form a clique. The resulting graph consists of a number of components, each component being a clique of vertices of some subject group. All subject groups and the number of lessons for each subject group are given in advance, so are the completely connected components of the graph and their cardinalities. Since we can determine if each clique can be colored with the given number of colors in constant time, we have a polynomial-time algorithm for determining if a solution to $[\circ \parallel t_L \parallel su, k_P = \boldsymbol{k}]$ exists.

Theorem 3.18. $[\circ \parallel t_L \parallel su, k_P = \boldsymbol{k}]$ *is polynomially solvable.* □

By introducing student group constraints, we forbid any two lessons of the same student group to be assigned to the same time slot. Now, we cannot use the algorithm of the previous sub-problem anymore: the resulting graph may have components that are not completely connected.

It turns out that adding this constraint type makes the problem hard to solve. In fact, GRAPH K-COLORABILITY can be seen as a special case of $[\circ \parallel t_L \parallel su, st, k_P = \boldsymbol{k}]$. For the reduction, we can restrict ourselves to student groups with only two subjects and one lesson for each subject group.

Theorem 3.19. [Cooper & Kingston, 1996] $[\circ \parallel t_L \parallel su, st, k_P = \boldsymbol{k}]$ *is NP-complete.*
Proof. There is a straightforward polynomial-time reduction from GRAPH K-COLORABILITY. For every vertex, we introduce one subject with one subject group and one lesson for this subject group. For every edge $\{u, v\} \in \mathcal{E}$, we define a student group with a curriculum consisting of subjects u and v only. The number of time slots is equal to $\mathcal{K}$, the number of colors. So $[\circ \parallel t_L \parallel su, st, k_P = \boldsymbol{k}]$ is NP-complete, even

if we restrict ourselves to student groups with only two subjects in their curricula and subject groups with only one lesson. □

This complexity result does not necessarily hold for lower year instances of the sub-problem. In the lower years, all students of the same level have the same curriculum. Most schools choose to split the students of one level into a small number of groups, which they call *classes*. These classes are considered the new entity in the timetabling problem. Students of class A follow all lessons together and they have no lessons with students from a different class B. We call this restriction *class binding*. Note that it can be considered as an optimal student group binding.

Definition 3.20 (Class binding constraints). Let k_{P} be a given subject group assignment. The set of all subject groups to which each student group has been assigned can be modeled by a function $K_{\mathrm{G}} : G \to \mathcal{P}(K)$, where

$$K_{\mathrm{G}}(g) := \{\, k \in K \mid v_{\mathrm{K}}(k) \in V_{\mathrm{G}}(g) \ \wedge\ k_{\mathrm{P}}(g, v_{\mathrm{K}}(k)) = k \,\}.$$

We say the subject group assignment satisfies the *class binding constraints* if for all student groups g_1 and g_2

$$K_{\mathrm{G}}(g_1) = K_{\mathrm{G}}(g_2) \quad \vee \quad K_{\mathrm{G}}(g_1) \cap K_{\mathrm{G}}(g_2) = \emptyset$$

holds. □

We abbreviate the class binding constraints by means of the symbol cb. The constraints are dependent on the subject group assignment, so we write $cb(k_{\mathrm{P}})$.

Imposing the class binding constraints does not necessarily make the problem more difficult to solve. In fact, finding a feasible time slot assignment while respecting student group and class binding constraints is easy. If the given subject group assignment $\boldsymbol{k}$ satisfies the class binding constraints, i.e., $cb(\boldsymbol{k})$ holds, then determining a feasible time slot assignment is trivial.

Theorem 3.21. $[\, cb(\boldsymbol{k}) \ \|\ t_{\mathrm{L}} \ \|\ su, st, k_{\mathrm{P}} = \boldsymbol{k} \,]$ *is polynomially solvable.*

Proof. We can group all student groups with exactly the same subject group assignment. These groups of student groups are the classes in our problem. Modeling $[\, cb(\boldsymbol{k}) \ \|\ t_{\mathrm{L}} \ \|\ su, st, k_{\mathrm{P}} = \boldsymbol{k} \,]$ as GRAPH K-COLORABILITY results again in a graph with disconnected components, with one component for each class. Each component is completely connected, because it contains all lessons of a specific class. On the other hand, a lesson corresponds to one class only. So, we have to find colorings for a set of disconnected cliques, like we did for $[\, \circ \ \|\ t_{\mathrm{L}} \ \|\ su, k_{\mathrm{P}} = \boldsymbol{k} \,]$. Note that all subject group constraints are implied by the student group constraints. □

Again, we summarize the most important complexity results in a diagram. Figure 3.2 shows the border between easy and hard for sub-problems concerning time slot assignment with different curricula.

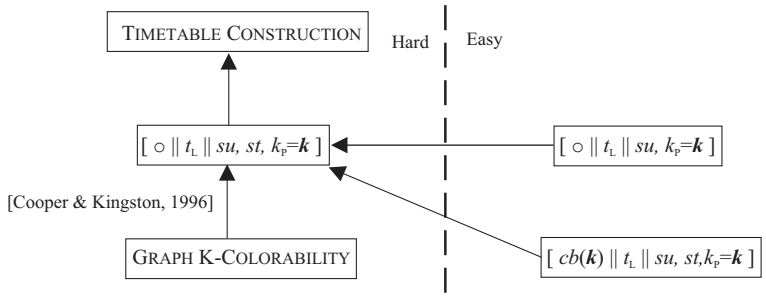


Figure 3.2. Complexity results for time slot assignment with different curricula.

3.6 Time slot assignment and educational requirements

Now, we focus on the educational requirement. Informally, it compels the planner to distribute students' lessons in the same subject over different days. We show that this constraint type can change the complexity of a timetable construction problem from easy to hard.

In the following, we restrict ourselves to only one student group g, i.e., $|G| = 1$. The student group is assigned to several subject groups, one for every subject in its curriculum. For every subject group, we have a number of lessons. The resulting set of lessons for g is denoted by $L' = L_G(g)$. We try to find a time slot assignment such that no two lessons of g conflict and all teacher availabilities are respected. Moreover, all lessons of the same subject group must be assigned to different days. That is, for every subject group $k \in K$, $\overline{L_K}(k) = 1$ holds. Note that the subject group constraints are implied by the student group constraints.

When we leave out the educational constraints, the remaining sub-problem is polynomially solvable. We only have to match the lessons to time slots and take availabilities into account. That is, the problem can be modeled as BIPARTITE MATCHING with vertex sets L' and T. In the bipartite graph, there is an edge (l,t) if, and only if, lesson l's teacher is available at time slot t. This leads to the following theorem.

Theorem 3.22. $[\,|G| = 1, \overline{L_K}(k) = 1 \;\|\; t_L \;\|\; st, su, ta, k_P = \boldsymbol{k}\,]$ *is polynomially solvable.* □

Now, we add the educational constraints to the model. Clearly, many time slot assignments that correspond to feasible matchings of the current model violate these constraints. Therefore, we add several characteristics to the bipartite matching problem such that we can model the educational constraints. First, we divide the lesson set L' and time slot set T into subsets $L_1, \ldots, L_m$ of lessons of the same subject group and subsets $T_1, \ldots, T_n$ of time slots on the same day. Here, $m = |K|$ and $n = |W|$. The edge set $\mathcal{E}$ can now be partitioned into subsets $\mathcal{E}_{kw}$ of edges from lessons of subject group $k \in K$ to time slots on day $w \in W$. The educational constraints for-

bid matchings that have more than one edge of each subset $\mathcal{E}_{kw}$. Due to its relation to MULTIPLE CHOICE MATCHING [Garey & Johnson, 1979], we call this problem MULTIPLE CHOICE MATCHING WITH PARTITIONED VERTEX SETS.

Definition 3.23. Multiple choice matching with partitioned vertex sets (MCMPVS). An instance of MCMPVS is a 5-tuple $\langle L, T, \mathcal{E}, L_1 \cup \ldots \cup L_m, T_1 \cup \ldots \cup T_n \rangle$, where

- $\langle L, T, \mathcal{E} \rangle$ denotes a bipartite graph with vertex sets L and T and edge set $\mathcal{E} \subseteq L \times T$,
- $L_1 \cup \ldots \cup L_m$ is a partitioning of L into m disjoint subsets, and
- $T_1 \cup \ldots \cup T_n$ is a partitioning of T into n disjoint subsets.

Let $\mathcal{E}_{ij} := (L_i \times T_j) \cap \mathcal{E}$ be the partitioning of the edge set. The objective is to find a matching $\mathcal{M} \in \mathcal{E}$ such that every vertex $l \in L$ is saturated and at most one edge of $\mathcal{E}_{ij}$ is included, i.e.,

$$\forall_{l \in L} \, \exists_{t \in T} \quad (l,t) \in \mathcal{M}, \tag{3.1}$$

$$\forall_{i=1,\ldots,m} \, \forall_{j=1,\ldots,n} \quad |\mathcal{E}_{ij} \cap M| \leq 1. \tag{3.2}$$

□

Lemma 3.24. MCMPVS *is NP-complete.*

Proof. We can verify whether a given matching satisfies constraints (3.1) and (3.2) in polynomial time. Therefore, MCMPVS $\in \mathcal{NP}$.

We show that SAT is polynomially reducible to MCMPVS. Let $\langle \mathcal{U}, \mathcal{C} \rangle$ be an instance of SAT, consisting of p variables $\mathcal{U} = \{x_1, \ldots, x_p\}$ and r clauses $\mathcal{C} = \{c_1, \ldots, c_r\}$. By r_i, we denote the number of clauses in which variable x_i appears. In Figure 3.3, we show how we transform the instance $x_1 \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (x_1 \vee \overline{x}_2 \vee x_3)$ to an instance of MCMPVS. Below, we give the formal transformation for an arbitrary instance. It is straightforward to show that the transformation is polynomial.

We construct the vertex sets L and T as follows. With each clause c_j, we associate one clause vertex $l_j \in L$. Furthermore, for every variable x_i occuring in clause c_j we introduce a (variable, clause) vertex $l_{(i,j)} \in L$ and two truth assignment vertices $t_{(i,j)}, f_{(i,j)} \in T$. The edge set E is defined as follows. For every clause c_j, we connect the corresponding clause vertex l_j to all truth assignment vertices that correspond to values that make c_j true. In our example of Figure 3.3, for the clause $c_3 = x_1 \vee \overline{x}_2 \vee x_3$, we draw edges from clause vertex l_3 to truth assignment vertices $t_{(1,3)}$, $f_{(2,3)}$, and $t_{(3,3)}$. Finally, we connect the (variable, clause) vertices $l_{(i,j)}$ to the two corresponding truth assignment vertices $t_{(i,j)}, f_{(i,j)}$. The transformation is completed by the definition of the partitionings of L and T. We divide L into $p + r$ subsets by introducing a subset L_{c_j} for each clause vertex l_j and a subset L_{x_i} for all (variable, clause) vertices $l_{(i,j)}$ for the same variable x_i. The set T is partitioned into $\hat{r}$ subsets, where $\hat{r} := \max_{i=1,\ldots,p} r_i$ is the maximum number of clauses in which one

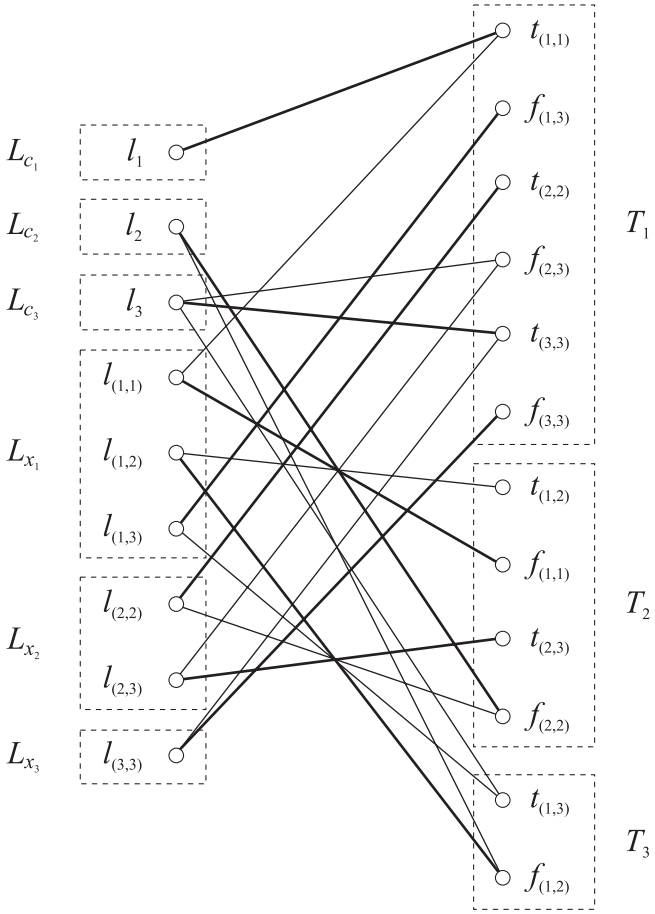


Figure 3.3. The transformation of SAT formula $x_1 \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (x_1 \vee \overline{x}_2 \vee x_3)$ to the corresponding instance of MCMPVS. The bold edges are chosen in the matching corresponding to the truth assignment $x_1 = \text{true}$, $x_2 = \text{false}$, and $x_3 = \text{true}$.

variable appears. For each variable x_i, the corresponding $2r_i$ truth assignment vertices $t_{(i,j)}, f_{(i,j)}$ are put in the first r_i subsets. More precisely, we put k-th truth assignment vertex $t_{(i,j)}$ in T_k and the k-th truth assignment vertex $f_{(i,j)}$ in $T_{(k+1) \mod \hat{r}}$.

In the remainder we show that the SAT instance $\langle \mathcal{U}, \mathcal{C} \rangle$ is satisfiable if, and only if, there exists a feasible matching, i.e., a matching $\mathcal{M}$ satisfying (3.1) and (3.2). Assume that we are given such a matching $\mathcal{M}$. Then, we can easily derive a truth assignment for the SAT formula by looking at the edges in the matching for the (variable, clause) vertices in L_{x_i}. Due to the constraints (3.2), either all vertices $l_{(i,j)}$ are connected to the vertices $t_{(i,j)}$, or all vertices are connected to $f_{(i,j)}$. In the first case, we set x_i equal to false, in the other case x_i equal to true. As all clause vertices l_j must be matched, due to constraint (3.1), this truth assignment satisfies the SAT formula: for every clause vertex l_j, its mate from T in the matching shows which literal makes the clause true.

We conclude by saying that a satisfactory truth assignment can be easily mapped onto a feasible matching. □

It is not hard to show that MCMPVS is a special case of $[\,|G| = 1, \overline{L_{\mathrm{K}}}(k) = 1 \;\|\; t_{\mathrm{L}} \;\|\; ed, st, su, ta, k_{\mathrm{P}} = \boldsymbol{k}\,]$.

Theorem 3.25. MCMPVS $\propto [\,|G| = 1, \overline{L_{\mathrm{K}}}(k) = 1 \;\|\; t_{\mathrm{L}} \;\|\; ed, st, su, ta, k_{\mathrm{P}} = \boldsymbol{k}\,]$.

Proof. Based on the sets $L_1, \ldots, L_m$, we introduce m subjects and m subject groups. For each set L_i, the corresponding subject groups k_i has $|L_i|$ lessons. The set $T = T_1 \cup \ldots \cup T_n$ represents the collection of all time slots, spread over n days. Finally, the edges between L and T are transformed to teacher availabilities. Since we can have different teachers for lessons of the same subject group, we can transform an arbitrary edge set $\mathcal{E}$, and therefore an arbitrary instance of MCMPVS to our timetable construction sub-problem. In fact, these two problems are essentially the same. □

Corollary 3.26. $[\,|G| = 1, \overline{L_{\mathrm{K}}}(k) = 1 \;\|\; t_{\mathrm{L}} \;\|\; ed, st, su, te, ta, k_{\mathrm{P}} = \boldsymbol{k}\,]$ *is NP-complete.* □

So we identified an NP-complete sub-problem of TC that becomes easy when we ignore all educational constraints. Figure 3.4 summarizes the complexity results of this section.

3.7 Room assignment and room types and blocks

Suppose the time slot assignment is given and we focus on the room assignment and all constraint types related to rooms. That is, each lesson is assigned to a time slot and we want to assign single lessons and block lessons to rooms such that there are no room conflicts, each lesson is taught in a suitable room, and both lessons of a block lesson are given in the same room. In our shorthand, this problem is $[\,\circ \;\|\; r_{\mathrm{L}} \;\|\; ro, rc, bt, rb, t_{\mathrm{L}} = \boldsymbol{t}\,]$. We show that this sub-problem is NP-complete if we take both blocks and room types into account.

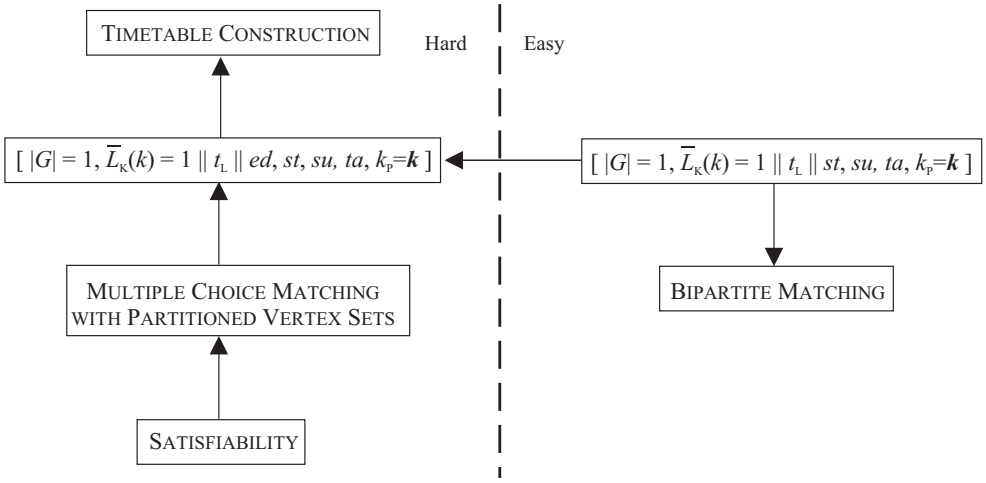


Figure 3.4. Complexity results for time slot assignment with educational requirements.

Carter & Tovey [1992] have studied the computational complexity of several room assignment problems. We present some of their results in our context here. First, we show that by leaving out either room types or blocks the room assignment problem turns out to be polynomially solvable. Suppose that there are no blocks. We can solve the room assignment problem with room types by considering each time slot independently. For any time slot $t \in T$, let L_t be the set of lessons assigned to t. Moreover, R is the set of rooms. The room assignment problem for time slot t can be modeled as the BIPARTITE MATCHING PROBLEM. The vertex sets are L_t and R, respectively. There is an edge $e = (l, r) \in \mathcal{E}$ if, and only if, room r is suitable for lesson l. So, we can solve the room assignment problem with room types $[\circ \parallel r_L \parallel ro, rc, t_L = \boldsymbol{t}]$ by finding matchings saturating all vertices in L_t in a bipartite graph for every time slot t.

If, for some time slot t, we find a maximum matching that does not saturate some vertex in L_t, then the problem does not have a feasible solution. So, this algorithm solves the problem in polynomial time and, consequently, we obtain the following result.

Theorem 3.27. $[\circ \parallel r_L \parallel ro, rc, t_L = \boldsymbol{t}]$ *is polynomially solvable.* □

On the other hand, assigning rooms with block constraints without taking the room types into account is polynomially solvable as well. The room assignment problem with blocks $[\circ \parallel r_L \parallel ro, bt, rb, t_L = \boldsymbol{t}]$ can be modeled as an interval graph coloring problem. Each lesson corresponds to a vertex in the graph, and there is an edge between two lessons if these two lessons overlap in time. By modeling the rooms

as colors, a feasible color assignment corresponds to a feasible room assignment and vice versa. The graph we have constructed is an interval graph, and there is a polynomial-time algorithm that colors these graphs with a minimum number of colors.

Theorem 3.28. [Carter & Tovey, 1992] $[\circ \parallel r_{\mathrm{L}} \parallel ro, bt, rb, t_{\mathrm{L}} = \boldsymbol{t}]$ *is polynomially solvable.* □

We can use this result in the case that every lesson can be taught in rooms of only one specific room type. The room assignment problem can be solved for every room type independently.

Corollary 3.29. $[|Y_{\mathrm{L}}(l)| = 1 \parallel r_{\mathrm{L}} \parallel ro, rc, bt, rb, t_{\mathrm{L}} = \boldsymbol{t}]$ *is polynomially solvable.* □

Carter & Tovey [1992] prove NP-completeness of the INTERVAL CLASSROOM ASSIGNMENT SATISFICE problem for only two time slots. In our notation, this problem is called $[|T| = 2 \parallel r_{\mathrm{L}} \parallel ro, rc, bt, rb, t_{\mathrm{L}} = \boldsymbol{t}]$. This complexity result strengthens a previous one for an arbitrary number of time slots due to Arkin & Silverberg [1987].

Theorem 3.30. [Carter & Tovey, 1992] $[|T| = 2 \parallel r_{\mathrm{L}} \parallel ro, rc, bt, rb, t_{\mathrm{L}} = \boldsymbol{t}]$ *is NP-complete.*

Proof. The NP-completeness follows from a polynomial-time reduction from 3SAT. As an intermediary result, a reduction from 3SAT to a sub-class of SAT is given. Instances of this sub-class have clauses of two literals (2-clauses) or three literals (3-clauses), with the additional restriction that no two 2-clauses (3-clauses) have any literal in common. Then, this sub-class of SAT is reduced to $[|T| = 2 \parallel r_{\mathrm{L}} \parallel ro, rc, bt, rb, t_{\mathrm{L}} = \boldsymbol{t}]$ in polynomial time. □

Corollary 3.31. $[\circ \parallel r_{\mathrm{L}} \parallel ro, rc, bt, rb, t_{\mathrm{L}} = \boldsymbol{t}]$ *is NP-complete.* □

In Figure 3.5, we summarize the complexity results for room assignment problems with room types and blocks.

3.8 Time slot assignment and blocks and teacher availabilities

Another source of complexity comes from teachers, their availabilities, and block lessons. Suppose we have only one student group g and that we want to assign lessons to time slots, while taking a given subject group assignment into account. Apart from assigning each lesson to a time slot for which the teacher is available, we must ensure that the student group and every teacher have at most one lesson at the same time. Furthermore, two lessons of a block must be taught at consecutive time slots.

We show that finding a time slot assignment that satisfies these constraints can take superpolynomial time. On the other hand, leaving out the block time constraints leads to a polynomially solvable problem.

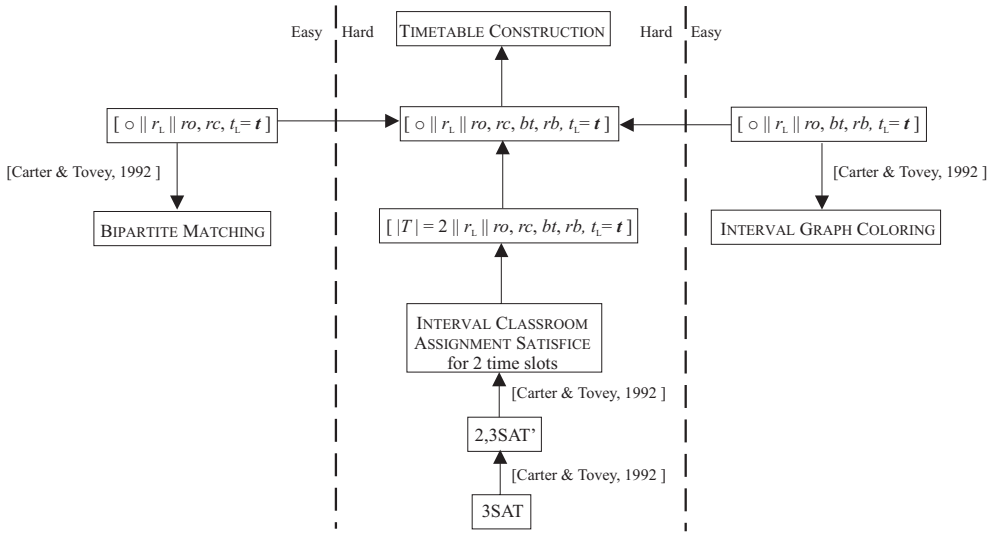


Figure 3.5. Complexity results for room assignment with room types and blocks. 2,3SAT' is a sub-class of SAT, proven NP-complete by Carter & Tovey [1992].

In the absence of blocks, the remaining problem is polynomially solvable. Actually, finding a feasible time slot assignment while taking the student group, teacher, and teacher availability constraints into account reduces to finding a maximum matching in a bipartite graph.

Theorem 3.32. $[\,|G| = 1 \;\|\; t_{\mathrm{L}} \;\|\; st, te, ta, k_{\mathrm{P}} = \boldsymbol{k}\,]$ *is polynomially solvable.*

Proof. We only consider the set of lessons $L_{\mathrm{G}}(g)$ of student group g, since these lessons must satisfy the student group constraints plus all other constraints. Let $L' = L_{\mathrm{G}}(g)$ be the set of lessons, T be the set of time slots, and let $\mathcal{E} \subseteq L' \times T$ give the possible time slots for each lesson. Then, in the absence of blocks, a time slot assignment is simply a matching $\mathcal{M}$ in the bipartite graph $(L', T, \mathcal{E})$ with each lesson in L' matched. Such a matching can be found in polynomial time if it exists. Hence, in this situation finding the time slot assignment if it exists is easy. □

In the presence of blocks, the two lessons that form a block must be assigned to two time slots that are consecutive, occur on the same day, and are not separated by a break. Due to this constraint, some lessons and time slots are related and an additional restriction on the matching in the bipartite graph is imposed. Suppose that the lessons l_1 and l_2 and time slots t_1 and t_2 are related, because they form a block lesson and a block time slot, respectively. Now, if the edge (l_1, t_1) is in the matching, then the edge (l_2, t_2) must also be in the matching, and vice versa.

In the remainder, we assume that the breaks divide all time slots into sets of two consecutive time slots. That is, for every time slot $t \in T$, we can determine

the time slot with which t forms a block time slot. We extend the matching model of $[\ |G| = 1\ \|\ t_{\mathrm{L}}\ \|\ st, te, ta, k_{\mathrm{P}} = \boldsymbol{k}\ ]$ by the constraints on blocks. We call the decision version of this extended matching problem BIPARTITE MATCHING WITH RELATIONS (BMR). Below, we prove that BMR is NP-complete.

Definition 3.33 (Bipartite matching with relations (BMR)). An instance of BMR is a 6-tuple $\langle L, T, \mathcal{E}, B, b, bt \rangle$, where

- $(L, T, \mathcal{E})$ denotes a bipartite graph with vertex sets L and T and edge set $\mathcal{E} \subseteq L \times T$,
- $B \subseteq L$ denotes the set of lessons that are part of a block lesson, so $|B|$ is even,
- $b : B \to B$ denotes a lesson's mate in a block, and
- $bt : T \to T$ denotes a time slot's mate in a block time slot.

The objective is to find a matching $\mathcal{M} \subseteq \mathcal{E}$ such that every $l \in L$ is saturated and both lessons of a block are assigned to a block time slot, i.e.,

$$\forall_{l \in B} \forall_{t \in T} \qquad (l,t) \in \mathcal{M} \ \Rightarrow\ (b(l), bt(t)) \in \mathcal{M}.$$

□

Lemma 3.34. [Ten Eikelder & Willemen, 2001] BMR *is NP-complete.*

Proof. Checking whether a given matching satisfies all conditions can be done in polynomial time. Hence BMR $\in \mathcal{NP}$.

Next, we show that the NP-complete problem 3SAT is polynomially reducible to BMR. Consider an instance $\langle \mathcal{U}, \mathcal{C} \rangle$ of 3SAT that consists of m clauses $\mathcal{C} = \{c_1, \ldots, c_m\}$ and variables $\mathcal{U} = \{x_1, \ldots, x_n\}$. Without loss of generality, we assume that each variable occurs in at least two clauses.

We now construct an instance of BMR in the following way. With each clause we associate three new blocks, each corresponding to one of the variables in the clause. For clause c_p we have the blocks l_{pi}, l'_{pi} for $i = 1, 2, 3$. The block l_{pi}, l'_{pi} can only be taught at block time slots t_{pi}, t'_{pi} or at block time slots f_{pi}, f'_{pi}. These possibilities correspond to the values true and false, respectively, for the corresponding variable.

Furthermore, for clause c_p there is one clause lesson cl_p. This is a single lesson, i.e., not a block, that can be taught at three time slots determined in the following way. If the variable corresponding to block l_{pi}, l'_{pi} occurs in the clause without negation, the clause lesson cl_p can be given at time slot f_{pi}. If the variable occurs in the clause in the negated form, the clause lesson can be given at time slot t_{pi}. The construction is illustrated by Figure 3.6, where the construction for the clause $c_p = x_1 \vee \bar{x}_2 \vee \bar{x}_3$ is shown.

Note that the necessity of a suitable time slot for the clause lesson implies that the blocks must be assigned to time slots in such a way that the clause holds. More precisely, there exists a time slot assignment, or matching, for the situation above if, and only if, the corresponding clause holds.

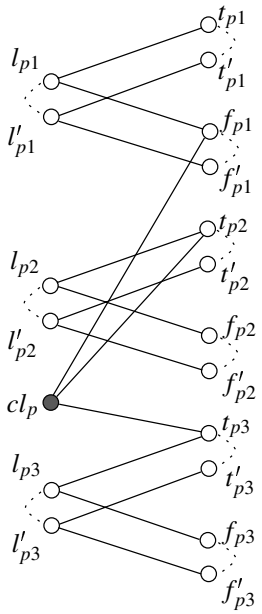


Figure 3.6. Construction for the clause $C_p = x_1 \vee \bar{x}_2 \vee \bar{x}_3$. Lessons and time slots that are connected by a dotted line form a block and a block time slot, respectively.

There is only one problem with this construction. In general, a variable occurs in several clauses, i.e., a variable corresponds to several blocks. To obtain a consistent value for such a variable, either all blocks corresponding to that variable must be matched with block time slots t_{**}, t'_{**} , or all blocks corresponding to that variable must be matched with block time slots f_{**}, f'_{**}.

So far, this condition does not necessarily hold. To enforce this condition, we use the second time slots (t'_{**}, f'_{**}) of the various block time slots. Consider all blocks that correspond to one variable in some arbitrary order, see Figure 3.7. Between two successive blocks, we add an additional consistency lesson, which can be given at the f'_{**} time slot of the first block, or at the t'_{**} time slot of the second block. We also add such a consistency lesson between the f'_{**} time slot of the last block and the t'_{**} time slot of the first block. The additional lessons are the dark lessons in Figure 3.7.

Clearly, a matching with each lesson matched is now only possible if the blocks are all matched with the t_{**}, t'_{**} time slots or all matched with the f_{**}, f'_{**} time slots. In other words, the various occurrences of a variable now have consistent values. We repeat this construction for each variable x_i.

Note that the clause lessons can only be given at first time slots of a block time slot, while the consistency lessons can only be given at second time slots of a block time slot. Hence the mechanisms used for making the clauses satisfiable and for

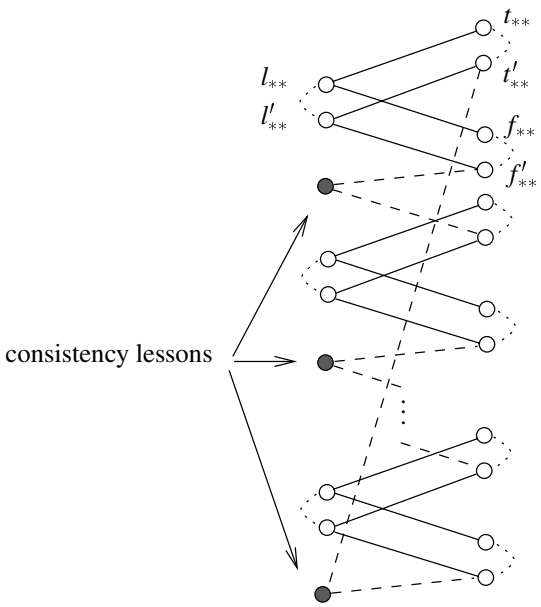


Figure 3.7. Enforcing consistency for blocks corresponding to the same variable by introducing consistency lessons.

obtaining consistent values for the various occurrences of a variable do not influence each other. In fact, blocks make it possible to couple these two mechanisms, which is essential for this reduction.

Now a matching that saturates all single lessons and blocks exists if, and only if, there are values for the variables x_i such that all clauses hold. Hence we have reduced the 3SAT problem to our BMR problem. It is trivial to show that the transformation is polynomial. This completes the proof of the NP-completeness of BMR. □

We have thus shown that the introduction of blocks makes the time slot assignment NP-complete. It is not hard to transform an instance of BMR to an instance of $[\,|G|=1 \parallel t_{\mathrm{L}} \parallel st, te, ta, bt, k_{\mathrm{P}} = \boldsymbol{k}\,]$. In fact, the transformation is a polynomial-time reduction.

Theorem 3.35. BMR $\propto [\,|G|=1 \parallel t_{\mathrm{L}} \parallel st, te, ta, bt, k_{\mathrm{P}} = \boldsymbol{k}\,]$.

Proof. We introduce a subject with only one subject group for each pair of block lessons and for every single lesson. The student group's curriculum consists of the collection of all these subjects. Moreover, each block and every single lesson have its own teacher, with availabilities that follow from the edge set of the bipartite graph. □

Corollary 3.36. $[\,|G|=1 \parallel t_{\mathrm{L}} \parallel st, te, ta, bt, k_{\mathrm{P}} = \boldsymbol{k}\,]$ *is NP-complete.* □

Note that our blocks consist of only two lessons. For blocks of three lessons,

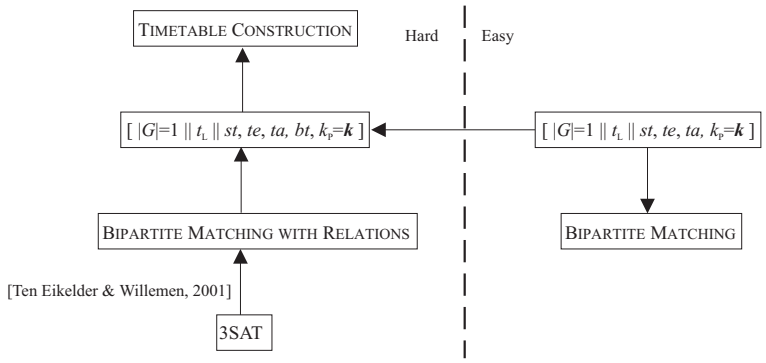


Figure 3.8. Complexity results for time slot assignment with blocks and teacher availabilities.

NP-completeness results have already been given by Cooper & Kingston [1996]. In Figure 3.8, all complexity results and their relationships are presented in a diagram.

3.9 Subject group assignment and capacitated subject groups

We have achieved student group binding by formulating decisions and constraints in terms of student groups instead of individual students. This introduces a new source of NP-completeness, because items with a given size, i.e., student groups, have to be assigned to containers with a given capacity, i.e., subject groups. Actually, assigning student groups to subject groups for one specific subject while satisfying the subject group size constraints is a bin packing problem.

Theorem 3.37. *[$|V| = 1 \parallel k_P \parallel ss$] is NP-complete.*
Proof. BIN PACKING is polynomially reducible to [$|V| = 1 \parallel k_P \parallel ss$]. □

Corollary 3.38. *[$\circ \parallel k_P \parallel ss$] is NP-complete.* □

Now, suppose we are given a time slot assignment and that we want to find a feasible subject group assignment, while satisfying the subject group size and the student group constraints. We show that the corresponding decision problem [$\circ \parallel k_P \parallel st, ss, t_L = \boldsymbol{t}$] is NP-complete. The sub-problem turns out to be polynomially solvable if we relax the subject group size constraints.

First, we prove that if every subject group has only one lesson, then determining a subject group assignment such that no student group has more than one lesson at the same time is easy.

Theorem 3.39. *[$|L_K(k)| = 1 \parallel k_P \parallel st, t_L = \boldsymbol{t}$] is polynomially solvable.*
Proof. Let $g \in G$ be a student group and $V' = V_G(g)$ be the set of subjects chosen by g. Furthermore, let T be the set of all time slots. We construct a bipartite graph in

the following way. With each subject in V', we identify a vertex in the left vertex set of the bipartite graph. Also, each time slot in T corresponds to one vertex in the right vertex set. We draw an edge (v,t) if there is a subject group of v that has its lesson assigned to t.

Now, a matching in this bipartite graph corresponds to subject group assignment that satisfies the student group constraint. We can solve the subject group assignment problem for each student group independently. An algorithm that determines a maximum matching for one student group at a time solves the problem in polynomial time. □

We have seen that finding a subject group assignment that satisfies all student group constraints can be seen as finding matchings in bipartite graphs. We can identify an edge (v,t) in a matching of g with an assignment of g to one specific subject group for subject v. If there is no bound on the number of students that can be assigned to the same subject group, then the matching problems for all student groups are independent.

In the remainder, we assume every student group consists of a single student. By introducing capacity constraints on the subject groups, we introduce a dependency between the matching problems of different student groups, or, students. For every subject v, we are given a maximal subject group size $\overline{S_V}(v)$. We can model this constraint as a constraint on the occurrence of edges in several matchings. We require that an edge (v,t) in the bipartite graphs may appear in no more than $\overline{S_V}(v)$ matchings. Below, we define this problem as MULTIPLE BIPARTITE MATCHING WITH MAXIMA (MBMM).

Definition 3.40 (Multiple bipartite matching with maxima (MBMM)). An instance of MBMM is a six-tuple $\langle V,T,S,\mathcal{E},\mathcal{X},u\rangle$, where

- V denotes the finite set of subject vertices,
- T denotes the finite set of time slot vertices,
- S denotes the finite set of students,
- $\mathcal{E} \subseteq V \times T$ denotes the collection of possible subject group assignments,
- $\mathcal{X} = \{ V_s \mid s \in S \}$ denotes the collection of curricula for all students, where for all $s \in S$, $V_s \subseteq V$ holds, and
- $u : V \to \mathbb{Z}^+$ denotes the maximal subject group sizes for each subject group of a given subject.

Let $\mathcal{E}_s \subseteq \mathcal{E}$ be the set of possible subject group assignments, relevant for student s, i.e., $\mathcal{E}_s := \{ (v,t) \in \mathcal{E} \mid v \in V_s \}$. This gives us $|S|$ matching problems on bipartite graphs $(V_s,T,\mathcal{E}_s)$ with maxima $u(v)$. The objective of MBMM is to find $|S|$ matchings $\mathcal{M}_s \subseteq \mathcal{E}_s$ of size $|V_s|$ such that every edge $(v,t) \in \mathcal{E}$ appears in no more than $u(v)$ matchings, or, formally, $|\{ s \in S \mid (v,t) \in \mathcal{M}_s \}| \leq u(v)$ holds for all $(v,t) \in \mathcal{E}$. □

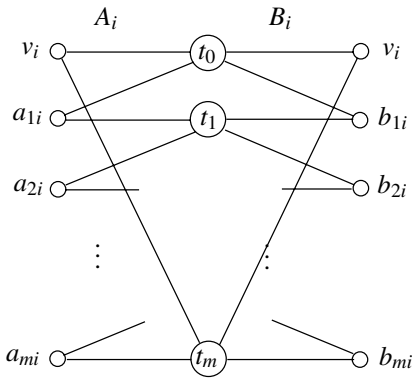


Figure 3.9. Matching problems for the students A_i and B_i.

In contrast to the multiple bipartite matching problem without maxima, it is unlikely that there exists a polynomial-time algorithm for the problem with maxima.

Lemma 3.41. [Ten Eikelder & Willemen, 2001] MBMM *is NP-complete.*

Proof. In the following, we use subject (time slot) if we mean subject vertex (time slot vertex). Suppose we are given p matchings $\mathcal{M}_i$. Checking whether the given matchings $\mathcal{M}_i$ satisfy all conditions can easily be done in polynomial time. Hence MBMM $\in \mathcal{NP}$.

Next, we show that the NP-complete problem NAE3SAT can be reduced to MBMM in polynomial time. Consider an instance $\langle \mathcal{U}, \mathcal{C} \rangle$ of NAE3SAT that consists of m clauses $\mathcal{C} = \{c_1, \ldots, c_m\}$ and n variables $\mathcal{U} = \{x_1, \ldots, x_n\}$. We now construct an MBMM instance with

- a time slot t_p for every clause c_p,
- an additional time slot t_0,
- a subject v_i and $2m$ subjects a_{pi}, b_{pi} for each variable x_i, and
- two students, A_i and B_i, both with a curriculum consisting of $m+1$ subjects, viz. student A_i with subjects v_i and a_{pi} and student B_i with subjects v_i and b_{pi}, for every variable x_i.

So, $T = \{t_0, t_1, \ldots, t_m\}$, $V = \{ v_i \mid i = 1, \ldots, n \} \cup \{ a_{pi}, b_{pi} \mid i = 1, \ldots, n, p = 1, \ldots m \}$, $V_{A_i} = \{v_i, a_{1i}, \ldots, a_{mi}\}$, and $V_{B_i} = \{v_i, b_{1i}, \ldots, b_{mi}\}$, where $1 \leq i \leq n$ and $1 \leq p \leq m$. Furthermore, we assume that all subjects v_i are taught twice, at time slot t_0 and at time slot t_m, and all subjects a_{pi} and b_{pi} are taught twice too, at time slot t_p and at time slot t_{p-1}.

The two matching problems for students A_i and B_i are shown in Figure 3.9. Since each subject must be matched, we conclude that either student A_i can follow all his a_{pi} subjects ($1 \leq p \leq m$) at time slots t_p and his v_i subject at time slot t_0, or he can follow all his a_{pi} subjects at time slots t_{p-1} and his v_i subject at time slot t_m. Also

A_i follows a_{pi} at	t_p	t_{p-1}
A_i follows v_i at	t_0	t_m
B_i follows b_{pi} at	t_{p-1}	t_p
B_i follows v_i at	t_m	t_0
corresponds to x_i is equal to	true	false

Table 3.1. Possible matchings for students A_i and B_i and corresponding value of variable x_i.

either student B_i can follow all his b_{pi} subjects ($1 \leq p \leq m$) at time slots t_p and his v_i subject at time slot t_0, or he can follow all his b_{pi} subjects at time slots t_{p-1} and his v_i subject at time slot t_m.

In other words, there exist two possible matchings for student A_i and two possible matchings for student B_i. Now suppose that $u(v_i) = 1$. Consequently, not both students A_i and B_i can follow the same lesson in subject v_i. That is, A_i and B_i must follow v_i at different times. Only two possibilities for students A_i and B_i remain, which correspond to the value of the variable x_i as given in Table 3.1.

So far, we have obtained a situation where the possibilities for students A_i and B_i represent the value of the variable x_i. It remains to reduce the possibilities of all students further, such that we have a yes instance of NAE3SAT if, and only if, the corresponding instance MBMM is a yes instance. Recall that in the NAE3SAT problem the question is whether there exist values for the boolean variables such that each clause holds true and not all literals in the clause are equal.

Consider an arbitrary clause, for instance $c_p = x_i \vee \bar{x}_j \vee x_k$. The question is whether there exist variables such that the following two conditions hold. First, the clause must hold, so we do not want that $x_i = \text{false}$, $x_j = \text{true}$, and $x_k = \text{false}$. In terms of the students A_* and B_*, we do not want that at time slot t_p student B_i follows subject b_{pi}, student A_j follows subject a_{pj}, and student B_k follows subject b_{pk}. Second, not all literals in the clause may have the same value. Since at least one literal must be true, this simply implies that not all literals may be true. Hence, we do not want that $x_i = \text{true}$, $x_j = \text{false}$, and $x_k = \text{true}$. In terms of the students A_* and B_*, we do not want that at time slot t_{p-1} student B_i follows subject b_{pi}, student A_j follows subject a_{pj}, and student B_k follows subject b_{pk}.

Note that the three subjects b_{pi}, a_{pj}, and b_{pk} are all only taught at time slots t_p and t_{p-1}. Hence, we can replace the b_{pi}, a_{pj}, and b_{pk} subjects by one new subject. By requiring that for this new subject the maximum group size is at most two, not all three students B_i, A_j and B_k can follow this subject at the same time slot. Hence, both above conditions must hold. We follow this procedure for all clauses. For each clause we identify three subjects, chosen as follows. If clause c_p contains variable x_i we select subject b_{pi}, if the clause contains $\bar{x}_i$ we select a_{pi}. By identifying these three

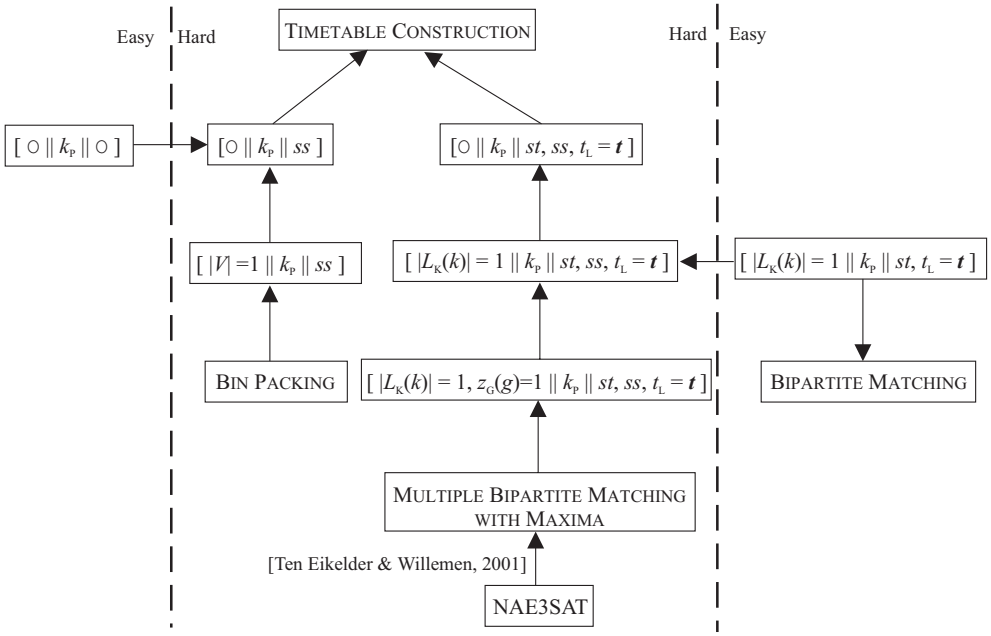


Figure 3.10. Complexity results for subject group assignment problem with capacitated subject groups.

subjects and giving the identified subject a maximal group size of two, we enforce that not all literals in the clause are false and also that not all literals in the clause are true.

Hence, we have reduced the NAE3SAT problem to our MBMM problem. This completes the proof of the NP-completeness of MBMM. □

We thus have shown that the addition of subject group size constraints can change the subject group assignment into a difficult problem. In the proof above we used maximal group sizes one and two. By introducing more students with the same curricula as A_i and B_i, we can reduce NAE3SAT to MBMM instances with more realistic maximal subject group sizes.

It is not difficult to see that our abstract MBMM problem captures the same problem as the subject group assignment we mentioned before.

Theorem 3.42. $\text{MBMM} \propto [\,|L_K(k)| = 1, z_G(g) = 1 \;\|\; k_P \;\|\; st, ss, t_L = \boldsymbol{t}\,]$. □

Corollary 3.43. $[\,|L_K(k)| = 1 \;\|\; k_P \;\|\; st, ss, t_L = \boldsymbol{t}\,]$ *is NP-complete.* □

In Figure 3.10, we give a schematic overview of the complexity results we have obtained for the subject group assignment problem with capacitated subject groups.

3.10 Conclusion

TCP, as defined in Chapter 2, is hard to solve. Or, more formally, TCP is NP-hard. We have determined its computational complexity by considering its decision variant, TC, and showing that several special cases of TC are NP-complete. We have concentrated on special cases that are sub-problems of TC. Each sub-problem focuses on different characteristics of TC. We have shown that removing one characteristic can make the hard-to-solve sub-problems easy, i.e., polynomially solvable.

We considered the following sub-problems.

- Determine a timetable while taking the subject group, teacher, and teacher availability constraints into account. Here, the latter constraint type makes the problem hard to solve.
- Determine a timetable for upper years, when all students have been assigned to subject groups and two lessons of a student must be given at different times. If conflicts are allowed, or if students have the same curricula and are sectioned into classes, the problem is polynomially solvable.
- Determine a timetable for only one student group in which lessons in the same subject are spread over different days while taking the teacher availabilities into account. When we require that there can only be one lesson in each subject on the same day, finding a feasible timetable becomes difficult. Without this restriction, we can easily determine a feasible timetable.
- Choose a suitable room for each lesson when a timetable is given and some lessons are blocks and are therefore taught at consecutive time slots. Without these blocks, it is much easier to find a timetable. Also, if the type of room does not matter, we are able to find rooms for all lessons, single or blocks, quickly.
- Determine a conflict-free timetable for students and teachers when some lessons are blocks and teachers have part-time contracts. Again, blocks make the problem more complex, even if there is only one student group and all lessons for this group are given in advance.
- For each subject, section the group of students into subject groups with a given capacity. The variant with unbounded capacities is trivial.
- For each subject, section the group of students into subject groups with a given capacity, such that each student has a conflict-free timetable. Without bounds on subject group size, we can solve this problem fast. But finding a feasible solution can be much harder when we do have to take these bounds into account.

In Figure 3.11, we summarize all major complexity results that were presented in this chapter. The sub-problems have been defined using the shorthand discussed in Section 2.3. Note that some sub-problems are theoretical problems, which consider only one student group or one lesson in each subject. By showing that these

special cases of TC are NP-complete, we know that solving TC is already hard when we restrict to only the corresponding subset of instances of TC. Although this is sufficient to prove NP-completeness of the general problem TC, one may wonder if real-life instances of TC belong to this specific subset, and are therefore hard to solve too. However, these sub-problems do show that some decompositions of TC do not reduce the computational complexity.

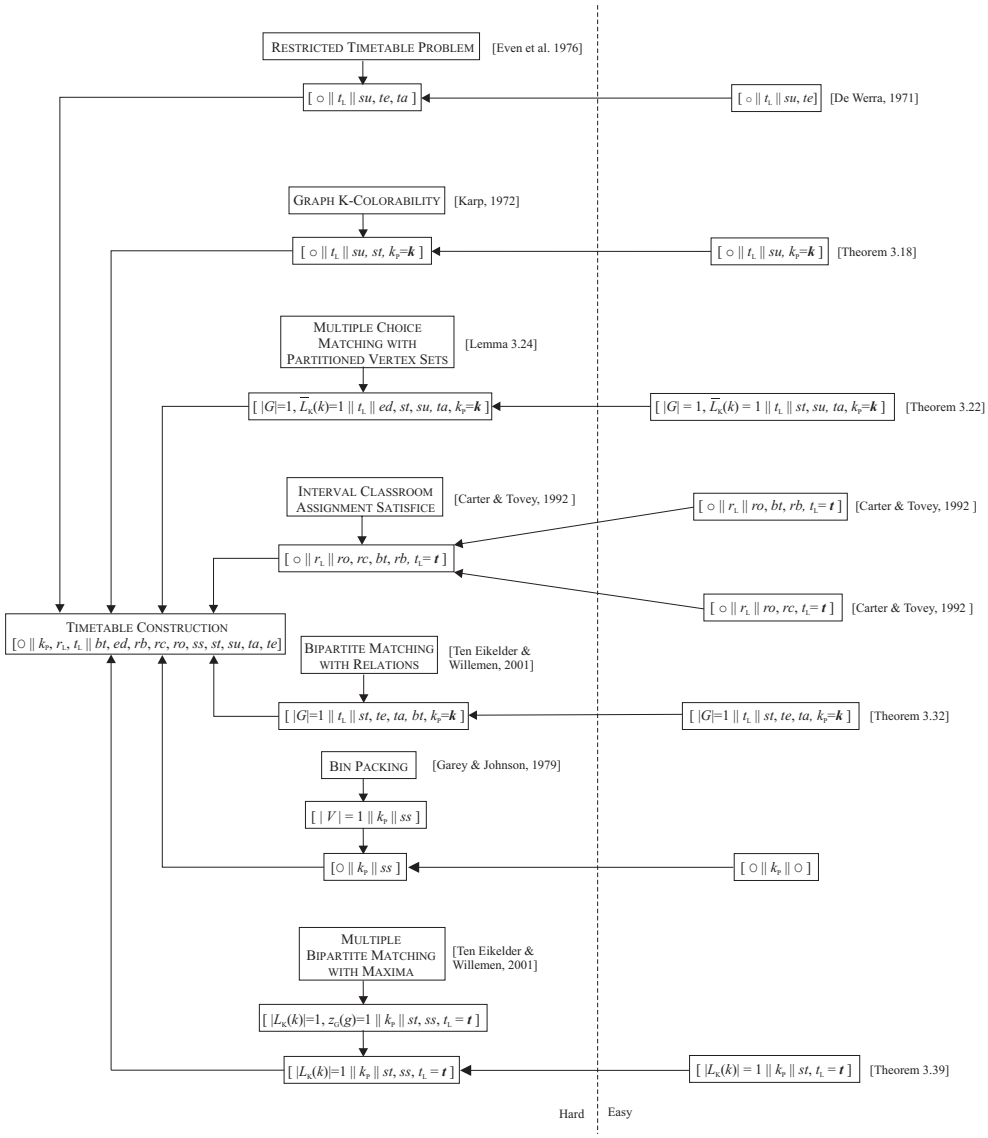


Figure 3.11. An overview of all complexity results. All problems on the left hand side are NP-complete, those to the right are polynomially solvable.

4

Time slot and subject group assignment

In this chapter we are concerned with the TIME SLOT AND SUBJECT GROUP ASSIGNMENT PROBLEM (TSSGAP). This problem plays a central role in the remainder of this thesis and addresses the issue of finding feasible assignments of student groups to subject groups and lessons to time slots. Constructing a feasible solution of TSSGAP can be seen as a first step in finding acceptable timetables at Dutch secondary schools. Most schools consider this a bottleneck problem and therefore tackle it first.

TSSGAP contains decisions and constraints of the TIMETABLE CONSTRUCTION PROBLEM (TCP) which we defined in Chapter 2. A main difference with TCP is that the room assignment decision is discarded. Moreover, we ignore blocks and room types.

The organization of this chapter is as follows. Section 4.1 concerns the definition of TSSGAP. Although TSSGAP itself has never been discussed in literature, there are several course timetabling and student sectioning problems that share characteristics with it. In Section 4.2, we review the relevant literature. Conditions for determining infeasibility of an instance of TSSGAP are the topic of Section 4.3. We discuss the approach currently used at most Dutch secondary schools in Section 4.4. We conclude this chapter with motivating our choice for a totally different approach.

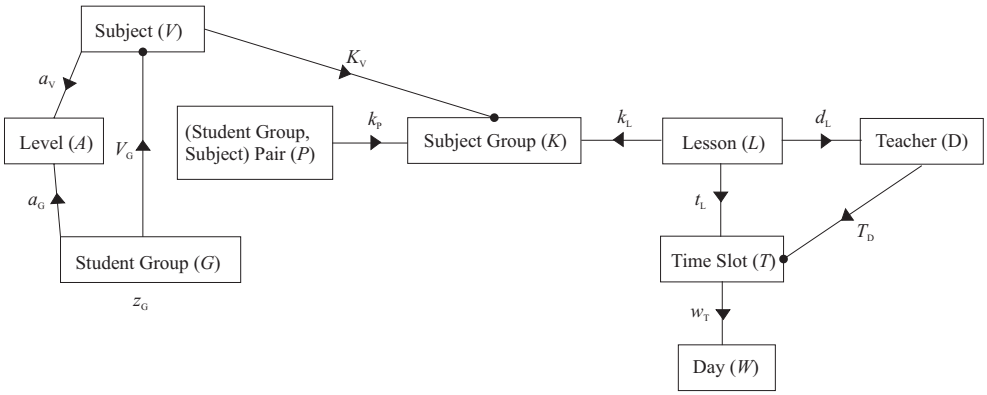


Figure 4.1. Sets and functions on these sets that define an instance of TSSGAP. Boxed concepts represent the sets and arcs represent the functions. Each arc is labeled with the function's name and points from the domain to the range. If a function returns a set instead of an element, then the end of the corresponding arc is marked by a bullet.

4.1 Problem definition

An instance of TSSGAP consists of all relevant data for time, teachers, subjects, student groups, and lessons. In Figure 4.1, we give a graphical representation of the sets and functions on these sets we use in our definition of TSSGAP. For a formal definition of these sets and functions, we refer to Section 2.2.2. For each subject group $k \in K$, the integer $\overline{L_K}(k)$ denotes the maximal number of lessons of k on the same day. Furthermore, $\overline{S_V}(v)$ denotes the maximal number of students that can be assigned to a subject group $k \in K_V(v)$ of subject $v \in V$. Finally, the input contains a positive integer denoting $|R|$, the total number of rooms.

A solution of TSSGAP is a subject group assignment k_P and a time slot assignment t_L. A subject group assignment is complete if every pair $(g,v) \in P$ is assigned to a proper subject group $k \in K_V(v)$. In this case, we say the subject group assignment satisfies the curriculum constraint (*cu*). A time slot assignment is complete if every lesson is assigned to a time slot. A solution is complete if both the subject group assignment and the time slot assignment are complete. Otherwise, a solution is called partial.

A solution is feasible if it satisfies all educational, room availability, student group, subject group, teacher availability, and teacher constraints. The room availability constraints are satisfied if at any time $t \in T$ the total number of lessons at each time slot $|L_T(t)|$ does not exceed the total number of rooms $|R|$. Here, $L_T(t) := \{ l \in L \mid t_L(l) = t \}$ is the set of all lessons assigned to time slot t. The other constraints have been defined in Section 2.2.4.

Input

Sets:	levels A, teachers D, student groups G, subject groups K, lessons L, time slots T, subjects V, and days W.
Mappings:	$a_{\mathrm{G}} : G \to A$, $a_{\mathrm{V}} : V \to A$, $d_{\mathrm{L}} : L \to D$, $k_{\mathrm{L}} : L \to K$, $K_{\mathrm{V}} : V \to \mathcal{P}(K)$, $T_{\mathrm{D}} : D \to \mathcal{P}(T)$, $V_{\mathrm{G}} : G \to \mathcal{P}(V)$, $w_{\mathrm{T}} : T \to W$, $z_{\mathrm{G}} : G \to \mathbb{Z}^+$.
Bounds:	$\lvert R \rvert \in \mathbb{Z}^+$, $\overline{L_{\mathrm{K}}} : K \to \mathbb{Z}^+$, $\overline{S_{\mathrm{V}}} : V \to \mathbb{Z}^+$.

Objective

Find	$k_{\mathrm{P}} : P \to K$ and $t_{\mathrm{L}} : L \to T$, satisfying all constraints and	
such that	$\forall_{(g,v)\in P} \quad k_{\mathrm{P}}(g,v) \in K_{\mathrm{V}}(v)$	(cu)
where	$P := \{\, (g,v) \in G \times V \mid g \in G \wedge v \in V_{\mathrm{G}}(g) \,\}$	

Constraints

$\forall_{k\in K}\forall_{w\in W}$	$\lvert L_{\mathrm{K,W}}(k,w) \rvert \leq \overline{L_{\mathrm{K}}}(k)$	(ed)
$\forall_{t\in T}$	$\lvert L_{\mathrm{T}}(t) \rvert \leq \lvert R \rvert$	(ra)
$\forall_{g\in G}$	$\mathsf{Resource}(L_{\mathrm{G}}(g))$	(st)
$\forall_{k\in K}$	$\mathsf{Resource}(L_{\mathrm{K}}(k))$	(su)
$\forall_{v\in V}\forall_{k\in K_{\mathrm{V}}(v)}$	$\sum_{g\in G_{\mathrm{K}}(k)} z_{\mathrm{G}}(g) \leq \overline{S_{\mathrm{V}}}(v)$	(ss)
$\forall_{l\in L}$	$t_{\mathrm{L}}(l) \in T_{\mathrm{D}}(d_{\mathrm{L}}(l))$	(ta)
$\forall_{d\in D}$	$\mathsf{Resource}(L_{\mathrm{D}}(d))$	(te)

where

$\forall_{k\in K}$	$G_{\mathrm{K}}(k)$	$:= \{\, g \in G \mid v_{\mathrm{K}}(k) \in V_{\mathrm{G}}(g) \wedge k_{\mathrm{P}}(g, v_{\mathrm{K}}(k)) = k \,\}$
$\forall_{g\in G}$	$K_{\mathrm{G}}(g)$	$:= \{\, k \in K \mid \exists_{v\in V_{\mathrm{G}}(g)}\, k_{\mathrm{P}}(g,v) = k \,\}$
$\forall_{d\in D}$	$L_{\mathrm{D}}(d)$	$:= \{\, l \in L \mid d_{\mathrm{L}}(l) = d \,\}$
$\forall_{g\in G}$	$L_{\mathrm{G}}(g)$	$:= \cup_{k\in K_{\mathrm{G}}(g)} L_{\mathrm{K}}(k)$
$\forall_{k\in K}$	$L_{\mathrm{K}}(k)$	$:= \{\, l \in L \mid k_{\mathrm{L}}(l) = k \,\}$
$\forall_{k\in K}\forall_{w\in W}$	$L_{\mathrm{K,W}}(k,w)$	$:= \{\, l \in L \mid k_{\mathrm{L}}(l) = k \wedge w_{\mathrm{T}}(t_{\mathrm{L}}(l)) = w \,\}$
$\forall_{t\in T}$	$L_{\mathrm{T}}(t)$	$:= \{\, l \in L \mid t_{\mathrm{L}}(l) = t \,\}$
$\forall_{L'\subseteq L}$	$\mathsf{Resource}(L')$	$:\Leftrightarrow \forall_{l_1,l_2\in L'}\, l_1 \neq l_2 \Rightarrow t_{\mathrm{L}}(l_1) \neq t_{\mathrm{L}}(l_2)$

Figure 4.2. Definition of the Time Slot and Subject Group Assignment Problem.

The objective of TSSGAP is to find a feasible complete solution of TSSGAP. In Figure 4.2, we give the formal definition of TSSGAP.

4.2 Related work

In Subsection 1.1.2, we presented an overview of educational timetabling problems. Timetable construction problems at Dutch secondary schools share characteristics with both school, or class-teacher, and university models. Since these two problems have been studied extensively, there are several papers in educational timetabling that are of interest. Consequently, one should focus more on the type of decisions included in the model than on the type of institution involved. Therefore, we restrict ourselves to problems that involve at least the course timetabling and the student sectioning decisions.

To the best of our knowledge no two papers deal with exactly the same problem. Alvarez-Valdes, Crespo & Tamarit [2002], Carter [2001], Hertz & Robert [1998], and Robert & Hertz [1996] deal with problems with many characteristics; Van Kesteren [1999], Lewandowski [1994], and Tripathy [1992] consider problems with only a few constraint types. Although the objective is always to determine an acceptable timetable, the problems differ in the way acceptance is defined. The criteria for an acceptable timetable are usually modeled by means of constraints and an objective function. This reflects the division of criteria into requirements that must be fulfilled, called hard constraints, and desires that should be fulfilled as well as possible, called soft constraints. Most objective functions are weighted sums of penalties for violated soft constraints; Sampson et al. [1995] and Tripathy [1992] maximize a weighted sum of preferences.

In Table 4.1, we give an overview of the relevant models we found in the literature. We indicate how the conditions we take into account in TSSGAP are modeled by others. If a condition must be met, then we mark it as a hard constraint ($\bullet$), otherwise the condition is considered a soft constraint ($\circ$) or simply ignored (.). Recall that TSSGAP is a feasibility problem, i.e., there is no objective function and we try to find an assignment that satisfies all constraints. By choosing this model, we do not have to trade off violations of soft constraints.

In the majority of the papers the problem under consideration is decomposed into a course timetabling sub-problem and a student sectioning sub-problem. That is, one of the decisions is made for a given, fixed assignment of the other. Alvarez-Valdes et al. [2002], Aubin & Ferland [1989], Hertz [1991], and Robert & Hertz [1996] iterate between the two sub-problems. Carter [2001] and Lewandowski [1994] first assign students to sections while roughly minimizing the number of lessons that are mutually conflicting. Next, a timetable is constructed and finally the section assignment is revised. Van Kesteren [1999] and Wood & Whitaker [1998] first determine which sections are taught simultaneously and then carry out student sectioning, whereas

	requirements						
reference	*cu*	*ed*	*ra*	*ss*	*st*	*ta*	*te*
Alvarez-Valdes, Crespo & Tamarit [2002]	○	●	●	○	○	●	●
Aubin & Ferland [1989], Ferland & Fleurent [1992]	●	.	○	○	○	○	○
Banks, Van Beek & Meisels [1998]	●	.	●	●	●	.	●
Carter [2001]	●	.	●	○	○	●	.
Hertz [1991]	●	.	●	●	○	●	○
Van Kesteren [1999]	○	.	.	○	●	●	.
Hertz & Robert [1998], Robert & Hertz [1996]	●	●	○	●	○	●	○
Lewandowski [1994]	●	.	●	●	○	.	.
Rudová & Matyska [2000]	●	.	●	●	○	●	●
Sampson, Freeland & Weiss [1995]	●	.	●	●	●	.	●
Tripathy [1992]	○	.	●	●	●	.	.
Wood & Whitaker [1998]	○	○	○	○	●	.	●
TSSGAP	●	●	●	●	●	●	●

Table 4.1. Differences between TSSGAP and related models in the literature. Conditions, modeled as constraints in Section 4.1, are either modeled as hard constraints (●), soft constraints in an objective function (○), or not included in the model (.). The following constraints are considered: the student sectioning is complete (*cu*), lessons in the same subject are on different days (*ed*), there are sufficient rooms (*ra*), the sections are balanced (*ss*), there are no conflicts for students (*st*), teachers are available (*ta*), and there are no conflicts for teachers (*te*).

	sub-problem	
reference	course timetabling	student sectioning
Alvarez-Valdes, Crespo & Tamarit [2002]	CH, TS	CH, TS
Aubin & Ferland [1989]	BI	B&B, BI
Banks, Van Beek & Meisels [1998]	FC	CH
Carter [2001]	CH	CH, BM
Ferland & Fleurent [1992]	BI, CH	B&B, BI
Hertz [1991]	TS	TS
Van Kesteren [1999]	B&B	BT
Lewandowski [1994]	TA	CH, BM
Hertz & Robert [1998], Robert & Hertz [1996]	TS	B&B, NF, TS
Rudová & Matyska [2000]	CP	CH
Sampson, Freeland & Weiss [1995]	TA	CH
Tripathy [1992]	CH	trivial
Wood & Whitaker [1998]	BT, SA	BM

Table 4.2. Algorithms used for solving sub-problems used by authors tackling related problems. The algorithms are based on constructive heuristics (CH), tree search (branch & bound (B&B), backtracking (BT), constraint programming (CP), forward checking (FC)), local search heuristics (best improvement (BI), simulated annealing (SA), tabu search (TS), threshold algorithm (TA)) and graph algorithms for bipartite matching (BM) and network flow (NF).

Tripathy [1992] first assigns students to sections, and then tries to cluster sections that are taught at the same time. Sampson et al. [1995] start with a course timetable and try to improve it, solving the student sectioning sub-problem for each timetable they examine.

A variety of algorithms is used for solving the two sub-problems. In Table 4.2 we summarize the algorithms used for each sub-problem. Some authors use local search heuristics, i.e., best improvement, tabu search, or threshold accepting. Others use enumerative methods, i.e., backtracking, branch & bound, constraint programming or forward checking. Constructive heuristics are also sometimes used.

4.3 Infeasibility checks

In Chapter 3, we have proven that TC is NP-complete. TSSGAP is a special case of TCP and several reasons for the hardness of TCP also apply to TSSGAP. Consequently, TSSGAP is NP-hard, so it is very unlikely that a polynomial-time algorithm exists that solves TSSGAP. Therefore, it is helpful to determine infeasibility of TSSGAP by means of fast computable conditions that must hold. Bin packing and graph coloring problems can be seen as sub-problems of TSSGAP. A feasible solution to TSSGAP exists only if there exists a solution to each of these sub-problems. In this section, we review the relevant literature and present several infeasibility checks for

TSSGAP based on solutions for the sub-problems.

4.3.1 Bin packing conditions

For each subject $v \in V$, we must partition the set $G_V(v) = \{ g \in G \mid v \in V_G(g) \} = \{g_1, \ldots, g_n\}$ of students for v into $m = |K_V(v)|$ subject groups. Any subject group $k \in K_V(v)$ may have at most $\overline{S_V}(v)$ students assigned to itself. This problem is equivalent to that of packing n items of size $z_G(g_1), \ldots, z_G(g_n)$ into m equal-sized bins with capacity $\overline{S_V}(v)$.

In one variant of the one-dimensional bin packing problem, the objective is to find the minimum number of bins β in which all items can be packed. This problem is NP-hard. There are several simple heuristics with $\mathcal{O}(n \log n)$ computation time, like *first fit decreasing* and *best fit decreasing*, that have an asymptotical worst-case performance of 1.22 [Coffman, Garey & Johnson, 1997]. Martello & Toth [1990] present several optimization algorithms. Scholl, Klein & Jürgens [1997] developed a more efficient optimization algorithm based on branch-and-bound and tabu search. They are able to solve instances of up to 500 items and bins with capacities of 150 or less.

One can also determine the minimum capacity κ for which there exists a packing of all items in $|K_V(v)|$ bins. This is equivalent to solving the NP-hard scheduling problem $P||C_{\max}$. The *longest processing time first* heuristic computes a solution in $\mathcal{O}(n \log n)$ time that is at most a factor $\frac{1}{3}$ off the optimum, see [Graham, 1969]. The *multifit* algorithm, due to Coffman, Garey & Johnson [1978], has a somewhat better performance ratio, viz. slightly larger than 1.22.

The bin packing conditions combine these two results. There exists no feasible solution of TSSGAP if there exists a subject $v \in V$ for which either $\beta > |K_V(v)|$ or $\kappa > \overline{S_V}(v)$ holds. Heuristics for these two variants give upper bounds $\hat{\beta}$ for β and $\hat{\kappa}$ for κ, respectively. It is clear that if $\hat{\beta} \leq |K_V(v)|$ and $\hat{\kappa} \leq \overline{S_V}(v)$, then the necessary conditions are satisfied for subject v.

4.3.2 Graph coloring and maximum clique condition

When we determine a feasible timetable for a given subject group assignment k_P, the resource constraints of TSSGAP exclude timetables with resource conflicts. Determining a conflict-free timetable is often seen as an application of graph coloring problems. In graph coloring problems, either the vertices or the edges are colored. As in this specific application vertices are colored, the term vertex coloring is sometimes used.

In the graph coloring model, a vertex and a color represent a lesson and a time slot, respectively. Two lessons $l_i, l_j \in L$ that share a resource map onto two vertices that are connected by an edge $\{l_i, l_j\}$. So, the set $E(k_P)$ of all edges in the graph following from subject group assignment k_P equals

$$\{ \{l_i, l_j\} \in \mathcal{P}(L) \mid G_L(l_i) \cap G_L(l_j) \neq \emptyset \ \vee \ k_L(l_i) = k_L(l_j) \ \vee \ d_L(l_i) = d_L(l_j) \}.$$

Here, $G_{\mathrm{L}}(l) := \{\, g \in G \mid l \in L_{\mathrm{G}}(g) \,\}$ denotes all student groups that attend lesson $l \in L$. A resource conflict in the timetable t_{L}, i.e., a pair of lessons l_i, l_j with a resource constraint for which $t_{\mathrm{L}}(l_i) = t_{\mathrm{L}}(l_j)$ holds, appears as a monochromatic edge $\{l_i, l_j\}$ in the corresponding graph. We denote the set of monochromatic edges for a given solution $\langle k_{\mathrm{P}}, t_{\mathrm{L}} \rangle$ of TSSGAP by $F(k_{\mathrm{P}}, t_{\mathrm{L}})$. A coloring is called *proper* if there are no monochromatic edges, i.e., $|\, F(k_{\mathrm{P}}, t_{\mathrm{L}}) \,| = 0$.

It is NP-hard to determine a proper coloring for a graph with a given number of colors; see Section 3.2. It is obvious that no proper coloring exists that uses at most $|\,T|$ colors if more than $|\,T|$ lessons are pairwise conflicting. In terms of graphs, no feasible solution to TSSGAP exists if a clique of the graph has size of $|\,T|$ or larger. Deciding whether a given graph has a clique of a given size is NP-complete, so finding the maximum clique size γ of a graph is NP-hard. The maximum clique condition $\check{\gamma} > |\,T|$ states that no feasible solution exists if a lower bound $\check{\gamma}$ on the maximum clique size γ exceeds the number of time slots.

Several variants of the graph coloring problem can be obtained by relaxing one of the constraints. For example, we can permit any number of colors and determine the minimum number of colors χ for which a proper coloring exists. It is clear that no feasible solution of TSSGAP exists if the chromatic number χ exceeds the number of time slots $|\,T|$. The graph coloring condition states that there exists no feasible solution of TSSGAP if $\hat{\chi} > |\,T|$, where $\hat{\chi}$ denotes a heuristically determined upper bound on χ.

On the other hand, we can determine a coloring with minimum number of monochromatic edges for a given number of colors. Then, we ignore the restriction that a coloring must be proper. This variant is clearly NP-hard as well.

Both graph coloring variants have received attention in the literature. The earliest graph coloring algorithms are heuristics based on the idea of successive augmentation. That is, in each step, an uncolored vertex with highest priority is colored. Heuristics vary in the way the priority is defined. Welsh & Powell [1967] choose the vertex with highest degree first. In DSATUR, Brélaz [1979] selects the vertex with the highest number of colors already used by its neighbors. Ties are broken by selecting the vertex with maximum degree. A different approach is followed in RLF [Leighton, 1979]. Here, a large independent set of uncolored vertices is colored with a new color in each step. Also, optimization algorithms based on branch-and-bound have been implemented, see e.g., Kubale & Jackowski [1985]. More recently, the majority of implementations have been based on local search techniques. The interested reader is referred to papers of Johnson, Aragon, McGeoch & Schevon [1991], Costa, Hertz & Dubuis [1995], Galinier & Hao [1999], Laguna & Martí [2001], and those that appeared in the DIMACS volume on graph coloring [Johnson & Trick, 1996].

Usually, the performance of graph coloring algorithms is determined on random graphs or artificial graphs with a known chromatic number. For random graphs $G_{n,p}$ with a given number of vertices n and estimated density p, there exist good estima-

tions of χ [Morgenstern, 1990]. The DIMACS benchmarks [Johnson & Trick, 1996] contain two course timetabling graphs. Lewandowski [1994] has developed a graph generator for graphs modeling these course scheduling graphs. It is not clear whether graphs encountered while solving TSSGAP share characteristics with random or artificial graphs. Coudert [1997] states that graphs that model real-life applications are easily solved by an optimization algorithm. This is mainly because these graphs are *1-perfect*, i.e., for these graphs $\gamma = \chi$ holds. His conjecture is supported by computational experiments on both real-life and theoretical graphs.

It turns out that there is no algorithm that dominates all others. Usually, fast heuristics have the worst possible performance ratio and are outperformed on large instances by local search heuristics. DSATUR has a time complexity of $\Theta(|E|)$ and usually performs better than RLF and the heuristic of Welsh & Powell [1967]. Sewell [1996] and Coudert [1997] report on optimization algorithms that find optimal colorings for 1-perfect graphs with up to 6760 vertices extremely fast. These algorithms first determine a maximum clique, then color all vertices in this clique, and finally determine an optimum coloring for the remaining vertices.

4.3.3 Restricted graph coloring conditions

The graph coloring problem considers the resource constraints of TSSGAP only. But we can derive other conditions if we also take the person availability or room availability conditions into account.

First, we consider the sub-problem of TSSGAP with resource and teacher availability constraints. Apart from a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a color set $\mathcal{C} = \{1, \ldots, |T|\}$, we are given a set of possible colors $\phi(v) \subseteq \mathcal{C}$ for each vertex $v \in \mathcal{V}$. The set of possible colors $\phi(v)$ for a vertex v represents the set of time slots for which the teacher for the corresponding lesson is available.

Now, suppose we have a clique $\mathcal{Q} \subseteq \mathcal{V}$ of $\mathcal{G}$. Since each vertex in $\mathcal{Q}$ must be assigned to a different color, at least $|\mathcal{Q}|$ colors must be possible for vertices in $\mathcal{Q}$. This implies that there is no feasible solution of TSSGAP if $|\cup_{v \in \mathcal{Q}} \phi(v)| < |\mathcal{Q}|$ for a clique $\mathcal{Q}$. For TSSGAP, we are given several cliques in advance. For example, all vertices corresponding to the same subject group, teacher, or student group represent a clique in the underlying graph.

De Werra [1999] gives necessary conditions for an extension that includes room availability constraints as well. Next to the input of the previous condition, i.e., the graph $\mathcal{G}$, the color set $\mathcal{C} = \{1, \ldots, |T|\}$, and the sets of possible colors $\phi(v)$, we are given $|T|$ nonnegative integers $h(c)$. Apart from the constraints that both vertices of an edge must have different colors, a vertex can only be colored to one of its possible colors and at most $h(c)$ vertices can be colored with a color $c \in \mathcal{C}$. In our case, $h(c) = |R|$ for all $c \in \mathcal{C}$.

Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_p$ be any partitioning of $\mathcal{V}$ into cliques. Moreover, let $\mathcal{A} \subseteq \mathcal{V}$ be a subset of vertices, and define $\mathcal{A}_i = \mathcal{Q}_i \cap \mathcal{A}$ to be the clique $\mathcal{Q}_i$ restricted to vertices in

$\mathcal{A}$. Finally, let $\mathcal{C}^* \subseteq \mathcal{C}$ be a subset of colors. Then, $|\cup_{v\in\mathcal{A}_i}\phi(v)\cap(\mathcal{C}-\mathcal{C}^*)|$ is equal to the number of colors not in $\mathcal{C}^*$ that is possible for at least one vertex in $\mathcal{A}_i$. If we sum this term for all cliques $\mathcal{A}_i$ and require that a feasible coloring must exist, then this sum may not be smaller than the number of vertices in $\mathcal{A}$ that will get colors in $\mathcal{C}-\mathcal{C}^*$. On the other side, in any feasible coloring the number of vertices in $\mathcal{A}$ that will be assigned a color from $\mathcal{C}^*$ is at most $\sum_{c\in\mathcal{C}^*} h(c)$. Summarized, no feasible solution to TSSGAP exists if for subsets $\mathcal{A}\subseteq\mathcal{V}$, $\mathcal{C}^*\subseteq\mathcal{C}$, and a partitioning $\mathcal{Q}_1,\ldots,\mathcal{Q}_p$ of $\mathcal{V}$ into p vertex-disjoint cliques,

$$\sum_{i=1}^{p}|\cup_{v\in\mathcal{A}_i}\phi(v)\cap(\mathcal{C}-\mathcal{C}^*)|+\sum_{c\in\mathcal{C}^*}h(c)<|\mathcal{A}|$$

holds.

We recall that several partitionings are known in advance. For example, we can consider the division of all vertices into subsets for each subject group, or for each teacher. In practice, checking each possible condition consumes far too much time.

4.4 Toward a solution approach

In this section, we discuss the approach for finding a time slot and subject group assignment currently used by the majority of planners at Dutch secondary schools. We give a detailed description of this approach. We start by giving an outline of the approach. In the subsequent paragraphs, the main steps of the approach are discussed in more detail. We conclude with a discussion on the pros and cons of this approach, supporting our choice for a different approach than the one currently used.

4.4.1 Outline of clustering approach

Many planners at Dutch secondary schools determine a time slot and subject group assignment in the same way. First, they construct clusters of lessons, i.e., subsets of lessons that will be taught simultaneously. At the same time, they determine a subject group assignment. Next, all clusters of lessons and the lessons that are not clustered are assigned to time slots. The result is a feasible time slot and subject group assignment. Since these assignments must satisfy the student group constraints, each student group may have at most one lesson in the same cluster.

The main steps of the clustering approach can be summarized as follows.

1. For each level, consider the subjects, subject groups, lessons, and pairs for this level only, and
   - *i.* determine an acceptable feasible *pattern*, i.e., clusters of subject groups, and a subject group assignment for all optional subjects,
   - *ii.* determine an acceptable subject group assignment for all mandatory subjects, and

iii. copy the pattern to obtain clusters of lessons, and combine incomplete clusters to obtain a feasible clustering.

2. Assign all clusters in the clustering and the remaining lessons to time slots.

The main steps of the aproach are described in more detail in the following paragraphs. We illustrate these steps by means of an example.

Example 4.1. We carry out the example for the *JEvwo4* instance. This is an instance of TSSGAP representing the fourth level of the VWO at the Pleincollege Sint-Joris in Eindhoven, which is one of the two schools that we use in our experiments of Chapter 6. The instance consists of nineteen subjects: nine mandatory subjects and ten optional subjects. All mandatory subjects and one optional subject, *ak*, have two subject groups, the other optional subjects have only one subject group. There is only one teacher for all lessons of a subject group. In Table 4.3, we give the teachers and the number of lessons for each subject group. Also, we give the number of students that have chosen an optional subject. For the sake of simplicity, we assume that each student group corresponds to a single student. There are 53 students.

subject	teachers	# lessons
bi	SVS, ZGS	2
ee	JNS, VEN	1
du	BAK, VBK	2
en	DIR, DMN	2
fa	AER, AER	2
gs	THO, THO	2
ne	KSM, RUY	3
lb	DKS, DKS	2
lo	RYP, RYP	2
wi	HAU, DNK	4

subject	teachers	# lessons	# students
ak	MYR, MYR	2	49
et	HVT	2	25
gr	WEE	3	4
la	WEE	5	1
mf	BRK	2	10
na	HAF	3	27
sk	TAP	3	27
sp	LRS	4	12
tf	BEE	2	26

Table 4.3. School data for the *JEvwo4* instance. The left table contains the data for all mandatory subjects, the right table those for all optional subjects. We give the teacher and number of lessons for each subject group, and the number of students who chose a particular optional subject.

□

4.4.2 Determine an acceptable feasible pattern for optional subjects

Lessons are not clustered in an arbitrary way. First, a pattern is constructed and this pattern is repeated several times to obtain a clustering of lessons. Formally, a *pattern* is a partitioning q_K of the set K of subject groups into m subsets. A subset q is called a cluster of subject groups. We denote the set of all these clusters by Q .

A pattern is *feasible* if, and only if, every student group has at most one subject group in each cluster. That is, a pattern is feasible if, and only if, there exists a

feasible subject group assignment $k_\mathrm{P} : P \to K$ such that $k_\mathrm{P}(g,v) \in K_\mathrm{V}(v)$ for all pairs $(g,v) \in P$ and

$$\forall_{g \in G} \forall_{k_1,k_2 \in K_\mathrm{G}(g)} \qquad k_1 \neq k_2 \;\Rightarrow\; q_\mathrm{K}(k_1) \neq q_\mathrm{K}(k_2).$$

Existence of a feasible subject group assignment for a given pattern can be determined in polynomial time, even if no subject group assignment is given. For each student group $g \in G$, a maximum matching $\mathcal{M}(g)$ in the bipartite graph $(V_\mathrm{G}(g), Q, \mathcal{E})$ must be found, where the edge set $\mathcal{E}$ is defined by

$$\forall_{v \in V_\mathrm{G}(g)} \forall_{q \in Q} \qquad (v,q) \in \mathcal{E} \quad :\Leftrightarrow \quad \exists_{k \in K_\mathrm{V}(v)}\, q_\mathrm{K}(k) = q.$$

A student group g fits in the pattern if $\mathcal{M}(g)$ saturates all vertices in $V_\mathrm{G}(g)$. Consequently, a pattern q_K is feasible if, and only if, every student group $g \in G$ fits in the pattern.

Now, suppose a subject group assignment k_P is given. Let $G_\mathrm{V}(v)$ denote the set of all student groups that have chosen subject $v \in V$, and $G_\mathrm{K}(k)$ the set of all student groups assigned to subject group k, given by

$$G_\mathrm{V}(v) := \{\, g \in G \mid v \in V_\mathrm{G}(g) \,\} \quad \text{and} \quad G_\mathrm{K}(k) := \{\, g \in G_\mathrm{V}(v) \mid k_\mathrm{P}(g,v) = k \,\}.$$

We define $z_\mathrm{K}(k)$, the *size* of a subject group $k \in K$, as the number of students assigned to k: $z_\mathrm{K}(k) := \sum_{g \in G_\mathrm{K}(k)} z_\mathrm{G}(g)$. If this size is close to the average subject group size $\sum_{g \in G_\mathrm{V}(v)} z_\mathrm{G}(g) / |\, K_\mathrm{V}(v) \,|$ for every subject $v \in V$, then the subject group assignment is acceptable. In TSSGAP, we have modeled this condition by the subject group size constraints (*ss*), see page 20. In contrast to verification of the feasibility condition, which can be done in polynomial time, determination of the acceptance of the subject group assignment is NP-hard. MULTIPLE BIPARTITE MATCHING WITH MAXIMA, as defined in Section 3.9, is a special case of this problem.

The problem of constructing a pattern and subject group assignment that is both acceptable and feasible was studied by Simons [1974] and Van Kesteren [1999]. Planners at Dutch secondary schools use software packages like Clusterfact, Clusterbord, Qlus+, and Curs to find solutions for this problem. The underlying algorithms enumerate a huge number of possible patterns, if not all. Often, the number of patterns evaluated is reduced by

- having no more clusters than the maximum number of subjects in a student group's curriculum, or
- forbidding clusters with two or more subject groups of the same subject.

In general, patterns are ranked according to several criteria. First, the number of students that do not fit is minimized. Furthermore, the subject group sizes must be close to their optima. Van Kesteren [1999] also takes the number of time slots, necessary to assign the resulting clustering of lessons to time slots, into account. The planner selects a pattern and subject group assignment that best meets his criteria.

subject	*ak*	*et*	*gr*	*la*	*mf*	*na*	*sk*	*sp*	*tf*
ak	49								
et	25	25							
gr			4						
la			1	1					
mf	9	5	1		10				
na	24		3		5	27			
sk	23		3		5	27	27		
sp	11	7	1	1		4	4	12	
tf	25	12				15	15		26

Table 4.4. Number of students that have chosen a certain pair of optional subjects. On the diagonal, the total number of students that has chosen a particular subject is given. Each subject, except *ak*, has only one subject group. For these subjects, the entries are equal to the number of conflicts that occur if both subjects are taught simultaneously.

Example 4.1. (Continued) We first determine which optional subjects can be taught simultaneously. For all combinations of two subjects, we calculate the number of students that have chosen both subjects. This data is presented in Table 4.4. The entries in the table show the number of conflicts that occur when subject groups of the two corresponding subjects are assigned to the same cluster. For example, if subjects *et* and *mf* are in the same cluster, then five students have a conflict. On the other hand, no conflicts are caused by teaching the subjects *et* and *gr* at the same time. One should take into account that *ak* is the only subject with two subject groups; all other optional subjects have only one subject group.

In Table 4.5, an acceptable feasible pattern is given: every student fits in the pattern and the subject groups *ak A* and *ak B* for *ak* have optimal sizes. Note that we represent a subject group by the subject's name followed by a capital. For each cluster q, we also give the number of students that have idle time. These students have not been assigned to any of the subject groups in q. □

4.4.3 Determine a subject group assignment for mandatory subjects

Next, we determine an assignment of all student groups G to subject groups for mandatory subjects. Or, equivalently, for each mandatory subject v, we partition the set G into $|K_V(v)|$ subsets. Therefore, finding a subject group assignment for mandatory subjects such that all subject groups have equal size can be seen as finding a partitioning into balanced subsets.

A commonly used approach is to copy the partitioning of G into subsets from a cluster in the pattern created in the previous step. Let $q \in Q$ be a cluster with m subject groups. That is, the set $K_q(q)$ of subject groups in cluster q consists of m elements, denoted $\{k_1, \ldots, k_m\}$. Furthermore, assume the subject group assignment for

	subject groups (# students)							
	1		2		3		4	
q_1:	*ak A*	(25)	*la A*	(1)	*na A*	(27)	*idle*	(0)
q_2:	*et A*	(25)	*sk A*	(27)			*idle*	(1)
q_3:	*ak B*	(24)	*gr A*	(4)			*idle*	(25)
q_4:	*sp A*	(12)					*idle*	(41)
q_5:	*tf A*	(26)	*mf A*	(10)			*idle*	(17)

Table 4.5. An acceptable feasible pattern for *JEvwo4*. The two subject groups for *ak* are denoted *ak A* and *ak B*. There are five clusters of subject groups. For each cluster, the set of subject groups and a dummy subject group (*idle*) are given. For each subject group, the number of students is given between parentheses.

optional subjects is feasible for the given pattern. Then the sets $G_{\mathrm{K}}(k_1),\ldots,G_{\mathrm{K}}(k_m)$, and the set of student groups with idle time $G-\cup_{i=1}^{m}G_{\mathrm{K}}(k_i)$ represent a partitioning of G into disjoint subsets.

A good candidate for a partitioning of G for our mandatory subject v is a cluster for which there are no students with idle time and the subject group sizes $z_{\mathrm{K}}(k_1),\ldots,z_{\mathrm{K}}(k_m)$ are almost equal. One could also choose a cluster with as many students with idle time as the sizes of the subject groups. Finally, we can combine two or more subsets of student groups to obtain a new subset with a total size closer to the average subject group size.

Example 4.1. (Continued) In Table 4.5, we see that every cluster has at least one small subset: cluster q_1 has only one student in *la A*, cluster q_2 has one student with idle time, etc. However, we can obtain a good partitioning by choosing a cluster and combining two subject groups, e.g., *ak A* and *la A* in cluster q_1. The resulting partitioning into two subsets has size 26 and 27 respectively. □

4.4.4 Obtain a clustering

Once we have chosen an acceptable feasible pattern q_{K}, we can use it to obtain a feasible clustering $c_{\mathrm{L}}: L\to C$ of lessons into $|C|$ clusters. We do that by copying each cluster $q\in Q$ with subject groups $K_q(q)=\{k_1,\ldots,k_m\}$ several times and replacing each subject group by lessons of the subject group. A cluster q is repeated $n(q):=\max_{i=1,\ldots,m}|L_{\mathrm{K}}(k_i)|$ times. This leads to $n(q)$ clusters c_i with lessons $L_c(c_i)=\{l_{i1},\ldots,l_{im}\}$, where $k_{\mathrm{L}}(l_{ij})=k_j$. To obtain the right set of lessons, we remove for each subject group k with $|L_{\mathrm{K}}(k)|<n(q)$ in total $n(q)-|L_{\mathrm{K}}(k)|$ lessons from the clusters $c_1,\ldots,c_{n(q)}$.

Some clusters are incomplete. That is, there are clusters for which some student groups have idle time. Among others, this is caused by copying clusters with unequal number of lessons per week. We can combine two or more incomplete clusters c_i and c_j, with $L_c(c_i)=\{l_{i1},\ldots,l_{im_i}\}$ and $L_c(c_j)=\{l_{j1},\ldots,l_{jm_j}\}$, if the corresponding

student groups have an empty intersection. That is,

$$\cup_{k=1,\ldots,m_i} G_{\mathrm{L}}(l_{ik}) \;\cap\; \cup_{k=1,\ldots,m_j} G_{\mathrm{L}}(l_{jk}) \quad = \quad \emptyset.$$

Example 4.1. (Continued) Cluster q_4 only consists of *sp A* and is repeated four times, because there are four lessons in subject *sp*. There are 41 students who did not choose *sp*. Cluster q_5 with subject groups *tf A* and *mf A*, repeated twice, is also incomplete. Since no student has chosen *sp* together with either *tf* or *mf*, we can combine the two clusters of lessons following from q_5 and two of the four clusters from q_4 to obtain two combined clusters with lessons from these three subject groups. □

4.4.5 Assign clusters of lessons and lessons to time slots

In the final step, we obtain a feasible timetable by assigning all clusters of lessons in optional subjects and all lessons in mandatory subjects to time slots. In this step, we have to take *all* constraints, defined in Section 4.1, into account. Note that the subject group constraints are satisfied implicitly by the way we obtain a clustering c_{L}.

We make the following observations. No two complete clusters for a given level can be assigned to the same time slot. However, we can assign an incomplete cluster and one or more mandatory lessons of the same level to the same time slot. Furthermore, a cluster c with lessons $L_c(c) = \{l_1, \ldots, l_m\}$ can only be assigned to a time slot t if all teachers in c are available at t, i.e.,

$$t \;\in\; T_{\mathrm{D}}(d_{\mathrm{L}}(l_1)) \cap \ldots \cap T_{\mathrm{D}}(d_{\mathrm{L}}(l_m))$$

holds. Consequently, the set of possible time slots of a cluster is equal to the intersection of the teacher availabilities of all teachers in the cluster.

Example 4.1. (Continued) In Figure 4.3, we give the final timetable for *JEvwo4*. There are five days of eight time slots each. In this timetable, some lessons in optional subjects, more specifically: lessons for *sp A*, *la A*, and *gr A*, are taught simultaneously with lessons for the subject group of the mandatory subjects denoted by the capital *A*. This implies that all students that have chosen *sp*, *la*, or *gr* have been assigned to the subject groups of the mandatory subjects denoted by the capital *B*. □

4.4.6 Discussion

The clustering approach discussed in this section has been used since the seventies. The time slot and subject group assignment problem is decomposed in two different ways. First, a clustering is determined for each level individually. Second, a pattern is determined and the clustering follows from this pattern. As timetabling was done manually during the early years, the need for decomposing time slot and subject group assignment is clear: by reducing the number of decisions to be made in each step, the problem became manageable.

From a computational complexity point of view, it is reasonable to determine a pattern of subject groups first before constructing a clustering. Let us assume a

mo1	② *et A* (HVT)	*sk A* (TAP)		
mo2	*bi A* (SVS)	*wi B* (DNK)		
mo3	*ee A* (JNS)	*lb B* (DKS)		
mo4	*wi A* (HAU)	*sp A* (LRS) ④		
mo5	*ne A* (KSM)	*ne B* (RUY)		
mo6	*gs A* (THO)	*ee B* (VEN)		
mo7	① *ak A* (MYR)	*la A* (WEE)	*na A* (HAF)	
mo8	*gs B* (THO)			
tu1	⑤ *tf A* (BEE)	*mf A* (BRK)	*sp A* (LRS) ④	
tu2	*lo A* (RYP)	*en B* (DIR)		
tu3	*lo A* (RYP)	*bi B* (ZGS)		
tu4	① *la A* (WEE)	*et A* (HVT)	*sk A* (TAP) ②	
tu5	③ *gr A* (WEE)	*ak B* (MYR)		
tu6	*wi A* (HAU)	*fa B* (AER)		
tu7	*bi A* (SVS)	*gs B* (THO)		
tu8				
we1	*bi B* (ZGS)			
we2	*ne B* (RUY)			
we3	① *na A* (HAF)			
we4	*du A* (VBK)	*sp A* (LRS) ④		
we5	*wi B* (DNK)	*wi A* (HAU)		
we6	*gs A* (THO)	*gr A* (WEE)③		
we7	*du B* (BAK)			
we8	① *la A* (WEE)			

th1	*ne A* (KSM)	*la A* (WEE) ①		
th2	*en A* (DMN)	*wi B* (DNK)		
th3	*wi A* (HAU)	*fa B* (AER)		
th4	③ *gr A* (WEE)	*ak B* (MYR)		
th5	*lb A* (DKS)	*en B* (DIR)		
th6	*fa A* (AER)	*du B* (BAK)		
th7	⑤ *tf B* (BEE)	*mf A* (BRK)	*sp A* (LRS) ④	
th8				
fr1	① *ak A* (MYR)	*la A* (WEE)	*na A* (HAF)	
fr2	*du A* (VBK)	*lb B* (DKS)		
fr3	*fa A* (AER)	*wi B* (DNK)		
fr4	*en A* (DMN)	*lo B* (RYP)		
fr5	*lb A* (DKS)	*lo B* (RYP)		
fr6	*ne A* (KSM)	*ne B* (RUY)		
fr7	② *sk A* (TAP)			
fr8				

Figure 4.3. A feasible timetable for *JEvwo4*. The lessons are denoted by the subject group's name and the teacher's name between parentheses. A cluster is represented by a shaded box and an encircled number which refers to the cluster's number.

clustering, i.e., a set of clusters of lessons, is given. Now, proving the existence of a feasible subject group assignment is NP-complete for arbitrary numbers of lessons per subject group. 3DM is a special case of the problem for one student with two lessons per subject group. But if there is exactly one lesson for each subject group, then the clustering is a pattern, i.e., a set of clusters of subject groups. We have shown that finding a feasible subject group for a pattern can be done in polynomial time.

During the last decades, the situation at Dutch secondary schools has changed. Among others, the number of part-time teachers has increased. Teacher availability is important during construction of a pattern. However, the teacher assignment and the unavailabilities are not taken into account in most currently used software packages. As a consequence, the set of possible time slots for a cluster can be empty, such that the cluster cannot be assigned to any time slot at all.

Moreover, the time slot and subject group assignment problems for different levels are not independent. Several teachers teach lessons in at least two levels. Therefore, some clusters of different levels cannot be assigned to the same time slot because they have a teacher in common. Consequently, no feasible time slot assignment can be found even if one exists.

Nowadays, computers can do a lot of bookkeeping and evaluate a huge number of alternatives fast. So we can think of tackling problems with more constraints and more decision variables than we could in the seventies. Therefore, we adopt a different solution approach. We discuss this new approach in the following chapter.

5

Solution approach

In the previous chapter, we defined the TIME SLOT AND SUBJECT GROUP ASSIGNMENT PROBLEM (TSSGAP). This problem is NP-hard, i.e., it is very unlikely that there exists a polynomial-time algorithm that solves TSSGAP. Therefore, we develop a heuristic that tries to find solutions for real-life instances of TSSGAP.

This chapter deals with the solution approach we developed for TSSGAP. The solution approach was introduced in [Willemen & Ten Eikelder, 2002]. Our approach updates the time slot assignment and subject group assignment decisions iteratively. The main idea is that we extend the subject group assignment step by step by assigning one (student group, subject) pair at a time, and maintaining a feasible complete timetable in each step.

In Section 5.1, we describe the idea of our approach by means of an example. Section 5.2 introduces the reader to the pseudocode in which we describe the procedures. In Section 5.3, we describe the basic step of the solution approach. Section 5.4 extends the subject group assignment by means of a tree search algorithm. The other main component of the algorithm, maintaining a feasible complete timetable, is discussed in Section 5.5. In Section 5.6, we discuss how we improve the performance of our algorithms by using problem specific knowledge. We conclude this chapter with an outline of the tree search algorithm, the parameters we have to set, and the range of values we can select from.

5.1 Core of the algorithm

Let us assume that we are given a feasible partial subject group assignment k_P and a feasible complete time slot assignment t_L. Student group g has already been assigned to two subject groups. Consequently, there are two lessons, l_1 and l_2, that cannot be taught simultaneously because g has to attend both. Now, suppose we want to assign student group g for a subject with two subject groups, say, A and B. Each subject group has exactly one lesson, so we can identify the lesson by its subject group.

In Figure 5.1, we show the steps of this example for the graph coloring representation we described in Subsection 4.3.2. In graph coloring terms, we are given a graph, following from k_P, and a feasible proper coloring of all vertices corresponding to t_L. Thus, the four lessons, l_1, l_2, A, and B, are represented by vertices. Student group g has to follow both l_1 and l_2, the two vertices that are surrounded by the dashed box. The edge between the two vertices corresponds to the resource constraint between these two lessons. There is a one-to-one correspondence between time slots and colors.

We try both possible assignments. If we assign student group g to A, then we introduce new resource constraints between the two lessons of g, i.e., l_1 and l_2, and the lesson of A. Now, in the current timetable, there is a resource conflict between l_2 and the lesson of A. Thus, timetable t_L is infeasible. Therefore, we need to repair it if we want to keep g assigned to A. We can do so by assigning a lesson that causes the resource conflict, e.g., the lesson of A, to a different time slot. We assign the lesson of A to the same time slot as the lesson of B. Now, the new timetable is feasible since all resource conflicts are resolved.

On the other hand, if we assign g to B, then we have no resource conflicts. In that case, the current timetable t_L remains feasible. So, for both assignments, assigning student group g to subject group A or to subject group B, there exists a feasible timetable.

5.2 Toward a formal description of the algorithms

In the remainder of this chapter, we give a detailed description of the algorithms that implement our solution approach. We will frequently use the concepts of completeness and feasibility of a solution. Therefore, we define the predicates $\mathsf{Complete}(k_P, t_L)$ and $\mathsf{Feasible}(k_P, t_L)$ that are used throughout the text. An assignment is complete if every element from its domain has been assigned. So, $\mathsf{Complete}(k_P)$ is by definition equivalent to $\mathrm{dom}(k_P) = P$, and $\mathsf{Complete}(t_L) :\Leftrightarrow \mathrm{dom}(t_L) = L$. A solution $\langle k_P, t_L \rangle$ of TSSGAP is complete if $\mathsf{Complete}(k_P) \wedge \mathsf{Complete}(t_L)$ holds.

A solution is feasible if all constraints, defined in Figure 4.2, are satisfied. $\mathsf{Satisfies}(k_P, t_L, \mathcal{C})$ is the abbreviation for "the assignments k_P and t_L satisfy all constraints in $\mathcal{C}$". Some constraints do not depend on one of the assignments; in that

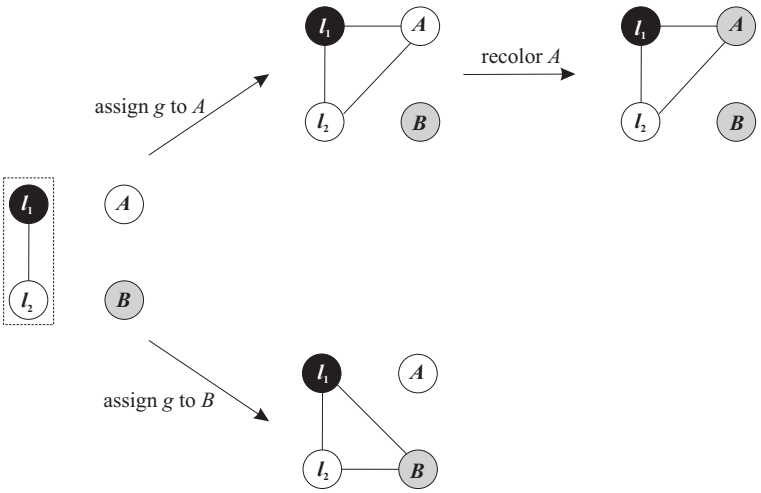


Figure 5.1. The basic step of the approach in graph coloring terms. We want to assign student group g for a subject with two subject groups, A and B, each having one lesson. Student group g has already been assigned to two lessons, l_1 and l_2. The corresponding vertices are surrounded by the dotted box. For each possible assignment, we determine a feasible proper coloring, corresponding to a feasible timetable.

case, we leave the unnecessary assignment out of the parameter list. For example, $\mathsf{Satisfies}(t_\mathrm{L}, ed)$ is short for the expression "the time slot assignment t_L satisfies all educational constraints (ed)". Now, we can define feasibility of a solution $\langle k_\mathrm{P}, t_\mathrm{L}\rangle$ as

$$\mathsf{Feasible}(k_\mathrm{P}, t_\mathrm{L}) :\Leftrightarrow \mathsf{Satisfies}(k_\mathrm{P}, t_\mathrm{L}, ed \wedge ra \wedge st \wedge su \wedge ss \wedge ta \wedge te).$$

We describe the main procedures of our solution approach by means of pseudocode. Comments in the pseudocode are preceded by // marks. Most procedures change the current solution $\langle k_\mathrm{P}, t_\mathrm{L}\rangle$. Therefore, the variable representing the current solution is declared global. The sets $K_\mathrm{P}(p)$ of possible subject groups for pairs $p \in P$ are also declared global. The initial value of a global variable is denoted by a hat, e.g., $\hat{k}_\mathrm{P}$ is the initial value of k_P. The begin and end of **while** loops and **if-then** selection statements are marked by increased and decreased indentation, respectively.

5.3 The basic step: Assigning one pair

The basic step of our algorithm is as follows. First, we extend the current subject group assignment k_P by the assignment of an unassigned pair $p \notin \mathrm{dom}(k_\mathrm{P})$ to one of its possible subject groups $k \in K_\mathrm{P}(p)$. Then, we determine a feasible timetable t_L. In Figure 5.2, we give the pseudocode for this step.

Let $\langle k_\mathrm{P}, t_\mathrm{L}\rangle$ be a feasible partial solution of TSSGAP for which $\mathsf{Complete}(t_\mathrm{L})$

TRY TO ASSIGN(pair p)
// global variables: solution $\langle k_P, t_L \rangle$, set $K_P(p)$ of possible subject groups for pair p

// $p \notin \mathrm{dom}(k_P)$
// $\wedge\ k_P = \hat{k_P} \wedge K_P(p) = \hat{K}$
// $\wedge\ \mathsf{Feasible}(k_P, t_L) \wedge \mathsf{Complete}(t_L) \wedge K_P(p) \subseteq K_V(v_P(p))$

$(k_0, \ldots, k_{m-1})$:= "the ordered sequence of all m subject groups in $K_P(p)$,
determined by subject group ordering rule *sgo*";
$i := 0$;
assigned := false;
while (**not** *assigned*) **and** $i < m$ **do**
  $k'_P := k_P \cup \{(p, k_i)\}$;
  // $\mathsf{Satisfies}(k'_P, t_L, ed \wedge ra \wedge ta)$
  if $\mathsf{Satisfies}(k'_P, ss)$ **and** $|F(k'_P, t_L)| > 0$ **then**
    // try to repair t_L
    t'_L := TIMETABLE WITH LEAST RESOURCE CONFLICTS($\langle k'_P, t_L \rangle$);
    if $|F(k'_P, t'_L)| = 0$ **then** $t_L := t'_L$;
  if $\mathsf{Feasible}(k'_P, t_L)$ **then**
    $k_P := k'_P$;
    assigned := true;
  else
    $K_P(p) := K_P(p) - \{k_i\}$;
    $i := i + 1$;

// $\mathsf{Feasible}(k_P, t_L) \wedge \mathsf{Complete}(t_L) \wedge K_P(p) \subseteq \hat{K}$
// $\wedge\ assigned \Rightarrow \mathrm{dom}(k_P) = \mathrm{dom}(\hat{k_P}) \cup \{p\} \wedge k_P \supseteq \hat{k_P}$
// $\wedge\ \neg assigned \Rightarrow k_P = \hat{k_P}$

return *assigned*;

Figure 5.2. Try to extend the current subject group assignment k_P by assigning an unassigned pair p to one of its possible subject groups $K_P(p)$. Subject groups are ordered following ordering rule *sgo*. If necessary, repair the current time slot assignment t_L. This procedure is defined in Figure 5.7. We remove all subject groups from $K_P(p)$ that were rejected.

holds. The set of possible subject groups for $p = (g, v)$ is a subset of $K_V(v)$. We try the possible subject groups in a predetermined order, defined by the subject group ordering rule *sgo*, and stop as soon as one subject group leads to a feasible extension of k_P.

Now, suppose we have assigned pair p to subject group k. We determine the feasibility of the new solution $\langle k_P \cup \{(p, k)\}, t_L \rangle$ by means of the following theorem.

Theorem 5.1. *Let $\langle k_P, t_L \rangle$ be a feasible partial solution. Furthermore, let $p = (g,v) \notin dom(k_P)$ be an unassigned pair and $k \in K_P(g,v) \subseteq K_V(v)$ be a subject group for (g,v). Determining whether* Feasible$(k_P \cup \{(p,k)\}, t_L)$ *holds can be done in polynomial time.*

Proof. Let $L_G(g)$ be the set of lessons that student group g has to attend. Moreover, let $G_K(k)$ be the set of student groups assigned to subject group k following from subject group assignment k_P. Then Feasible$(k_P \cup \{(p,k)\}, t_L)$ is implied by

$$\sum_{g' \in G_K(k)} z_G(g') + z_G(g) \leq \overline{S_V}(v) \ \wedge \ \forall_{l_1 \in L_G(g)} \forall_{l_2 \in L_K(k)} \ t_L(l_1) \neq t_L(l_2).$$

Satisfaction of this constraint can be verified in polynomial time. □

If the new subject group assignment $k'_P := k_P \cup \{(p,k)\}$ satisfies all subject group size constraints, then we determine whether the current timetable t_L is feasible for k'_P. In case t_L is infeasible, then at least one lesson $l_1 \in L_G(g)$ of g conflicts with a lesson $l_2 \in L_K(k)$ of k. That is, $\{l_1, l_2\} \in F(k'_P, t_L)$, where $F(k'_P, t_L)$ denotes the set of resource conflicts of solution $\langle k'_P, t_L \rangle$, see the definition on page 60. Possibly, a different timetable t'_L is feasible for k'_P. Instead of constructing such a feasible complete time slot assignment t'_L from scratch, we try to repair the current timetable t_L by assigning a few lessons to different time slots. We describe the implementation of this step in Subsection 5.5.2.

5.4 Tree search algorithm: Assigning all pairs

In this section, we show how we use the procedure of Figure 5.2 in the tree search algorithm that implements our approach. We start with a constructive heuristic that assigns pairs in a specific order and stops as soon as a dead end is detected. Next, we give an algorithm that performs multiple independent runs of the constructive heuristic. We continue our presentation of the tree search algorithm with extensions of the constructive heuristic. The two extensions deal with different ways to tackle dead ends encountered by the constructive heuristic. Moreover, we exploit a relationship between pairs to improve the performance of the tree search algorithm. Finally, we give the pseudocode of the full tree search algorithm for solving TSSGAP using the procedures we have described before.

5.4.1 Constructive heuristic

A simple implementation of the solution approach is a constructive heuristic for the subject group assignment. All unassigned pairs are ordered following the pair ordering rule *po*. The constructive heuristic selects an unassigned variable $p = (g,v)$, determined by the pair selection rule *ps*, and tries to extend the current solution $\langle k_P, t_L \rangle$ by assigning p to one of the possible values $K_V(v)$. If all possible subject groups for p have been tried, and no feasible timetable was found for any subject group, then the constructive heuristic stops in a dead end.

Instead of stopping the algorithm and returning the partial solution $\langle k_P, t_L \rangle$ found at the dead end, we can continue with the next unassigned pair. In that case, we give up the requirement of finding a feasible *complete* solution and try to assign as many pairs as possible instead. We can also restart the heuristic a given number of times for different pair orderings and stop when the maximal number of restarts is reached or a feasible complete solution is found.

We chose to perform multiple runs of the constructive heuristic. In Figure 5.3, we give the pseudocode of this algorithm. Since we do not use data collected during previous runs to guide the search in the following runs, all runs are independent of each other. By randomizing the variable and value selection, and performing more than one run, we hope to explore different parts of the solution space. We restart the constructive heuristic at most *pr* times.

5.4.2 Dead end handling

A dead end occurs when the current feasible partial solution $\langle k_P, t_L \rangle$ cannot be extended by assigning an unassigned pair p to one of the possible subject groups $K_P(p)$. That is, we were unable to find a complete timetable t'_L for which $\mathsf{Feasible}(k_P \cup \{(p,k)\}, t'_L)$ holds for every subject group $k \in K_V(v)$ of pair $p = (g,v)$. One possible reason is that no feasible complete solution has the current subject group assignment as a partial solution:

$$\left(\exists_{k'_P : P \to K} \exists_{t'_L : L \to T}\, k_P \subsetneq k'_P \wedge \mathsf{Complete}(k'_P, t'_L) \wedge \mathsf{Feasible}(k'_P, t'_L)\right) \Leftrightarrow \text{false}.$$

In that case, at least one of the previously made assignments has to be revised. We can restart from scratch and try to construct a different solution by randomizing the variable and value selection. This is implemented in the algorithm of Figure 5.3. We can also see the state of the algorithm at the dead end, denoted by $\delta = \langle \langle k_P, t_L \rangle, p, K_P \rangle$, as information about failure of the algorithm and use this information in the next steps of the algorithm.

We implement two ways to revise subject group assignments made in the past. First, we reconsider pairs in the same order we have assigned them in the previous steps. That is, we return to the previous subject group assignment and try a subject group that has not been considered before. Second, we remove all assignments and restart with the pair at the dead end being the first pair in the ordering. In this situation, we move the *bottleneck pair* to the beginning of the ordered pair sequence.

In Figure 5.4, we present backtracking and bottleneck identification in the context of our tree search algorithm. We can switch both ways of dead end handling "on" or "off" by parameters *bt* and *bip*. By having both parameters switched "off", the algorithm in Figure 5.4 performs multiple runs of the constructive heuristic, i.e., the algorithm is similar to the one in Figure 5.3.

Backtracking. If we apply backtracking, then we undo the *last* subject group assignment when a dead end occurs. Let p be the pair for which the dead end

```
MULTIPLE INDEPENDENT RUNS OF CONSTRUCTIVE HEURISTIC()
// global variable: solution ⟨k_P, t_L⟩

//  k_P = k̂_P ∧ t_L = t̂_L ∧ Feasible(k_P, t_L) ∧ Complete(t_L)

i := 0;
repeat
    P^not := P − dom(k_P);
    "order all unassigned pairs P^not following pair ordering rule po";
    forall (g, v) ∈ P^not do K_P(g, v) := K_V(v);
    restart := false;
    while P^not ≠ ∅ and (not restart) do
        p := "the next pair in P^not following pair selection rule ps";
        assigned := TRY TO ASSIGN PAIR(p);
        if assigned then
            P^not := P^not − {p};
        else
            // dead end
            restart := true;
            "⟨k_P, t_L⟩ becomes the initial solution, i.e., ⟨k̂_P, t̂_L⟩";
            i := i + 1;
until Complete(k_P) or i > pr;

//  k_P ⊇ k̂_P ∧ Feasible(k_P, t_L) ∧ Complete(t_L)

return Complete(k_P);
```

Figure 5.3. Pseudocode of the algorithm that implements multiple independent runs of a constructive heuristic. We try to assign the next unassigned pair $p \in P^{\text{not}}$ by means of the algorithm in Figure 5.2. The next unassigned pair is determined by the pair ordering rule *po* and the pair selection rule *ps*. If the pair p cannot be assigned, then a dead end is detected. In that case, we restart the outer loop. The outer loop is restarted at most *pr* times.

```
TRY TO ASSIGN ALL PAIRS(set of pairs P')
// global variables: solution ⟨k_P, t_L⟩, sets K_P of possible subject groups for all pairs

//  k_P = k̂_P ∧ t_L = t̂_L ∧ Feasible(k_P, t_L) ∧ Complete(t_L) ∧ P' ∩ dom(k_P) = ∅

i := 0;
repeat
   P^not := P';
   "order all pairs P^not following pair ordering rule po";
   forall (g, v) ∈ P^not do K_P(g, v) := K_V(v);
   restart := false;
   while P^not ≠ ∅ and (not restart) do
      p := "the next pair in P^not following pair selection rule ps, respecting
           the fixed order of bottleneck pairs";
      assigned := TRY TO ASSIGN PAIR(p);
      if assigned then
         P^not := P^not − {p};
      else
         // dead end
         if "do backtracking" then
            // bt = true ∧ p = (g, v) ∧ {(pred(p), k)} ∈ k_P
            k_P := k_P − {(pred(p), k)};
            K_P(pred(p)) := K_P(pred(p)) − {k};
            K_P(p) := K_V(v);
            P^not := P^not ∪ {pred(p)};
         else
            restart := true;
            if "restart with bottleneck pair first" then
               // bip = true
               "move bottleneck pair to front and fix position in ordering";
            "⟨k_P, t_L⟩ becomes the initial solution, i.e., ⟨k̂_P, t̂_L⟩";
            i := i + 1;
until P' ⊆ dom(k_P) or i > pr;

//  k_P ⊇ k̂_P ∧ dom(k_P) ⊆ dom(k̂_P) ∪ P' ∧ Feasible(k_P, t_L) ∧ Complete(t_L)

return P' ⊆ dom(k_P);
```

Figure 5.4. The extension of the algorithm in Figure 5.3 that includes backtracking and bottleneck identification. We try to assign every pair in the set $P' \subseteq P$ by means of the algorithm defined in Figure 5.2. The algorithm has five parameters. Pair selection follows rules *po* and *ps*. The two switches *bt* and *bip* determine whether we apply backtracking or restart with the bottleneck pair first. Finally, we restart the outer loop at most *pr* times.

$\langle\langle k_P, t_L\rangle, p, K_P\rangle$ occurs, $\text{pred}(p)$ be the pair that was assigned just before we tried to assign p, and $k = k_P(\text{pred}(p))$ be the subject group to which $\text{pred}(p)$ has been assigned. Then we remove $\{(\text{pred}(p), k)\}$ from the current subject group assignment k_P. Because the current timetable t_L remains feasible, we continue the search with the feasible solution $\langle k_P - \{(\text{pred}(p), k)\}, t_L\rangle$. Therefore, we do not have to maintain a history of timetables.

The ordered sets K_P and the order in which the previous pairs have been assigned, i.e., the function pred, represent the tree of partial subject group assignments we have evaluated. This data must be updated whenever we perform a backtracking.

Trying to assign $\text{pred}(p)$ to k again is not very useful. Therefore, we should remove k from $K_P(\text{pred}(p))$, the set of possible subject groups for $\text{pred}(p)$. Note that at the dead end the set $K_P(p)$ for pair $p = (g, v)$ is empty. Since we will reconsider the assignment of p for a subject group assignment that differs from k_P, we must initialize $K_P(p)$. Recall that the initial value of $K_P(g, v)$ is the set of all subject groups for subject v.

Restarting with bottleneck decision first. The pair p that could not be assigned in the dead end $\delta = \langle\langle k_P, t_L\rangle, p, K_P\rangle$ is considered a bottleneck in finding a feasible complete solution. In any *complete* solution this pair *must* be assigned to a subject group, and in the dead end δ we are unable to assign it. By moving the bottleneck pair p to the beginning of the ordering, and restarting the heuristic with this new ordering, we try to assign p first in the next run. If p is assigned in this run, then we construct a solution that is different from $\langle k_P, t_L\rangle$, the partial solution in the dead end we try to escape from.

If we apply backtracking and bottleneck identification at the same time, we move the bottleneck pair only after we have decided not to perform backtracking anymore. As a consequence, we may have a set Δ of dead ends from which we can select the bottleneck pair. We choose the pair from the dead end that occured the farthest in our search, i.e., we choose a dead end $\langle\langle k_P, t_L\rangle, p, K_P\rangle \in \Delta$ for which $|\, k_P \,|$ is maximal, and consider p as the bottleneck pair that must be moved to the front of the ordered list.

5.4.3 Considering pairs of the same subject as one group

In practice, generic methods like tree search perform better if domain specific knowledge is used. For example, chronological backtracking is more effective if consecutive variables are somehow related. In our case, there is a direct relationship between pairs if they correspond to the same student group or subject.

We exploit this relationship in our tree search algorithm. All pairs of the same subject are considered as one group. Thus, we partition the set of unassigned pairs $P - \text{dom}(k_P)$ into subsets P_j of pairs for the same subject. The subjects are ordered following the subject ordering rule *so*. We try to complete the assignment of one subject before we continue with the next one. That is, for each subset P_j, we apply the algorithm presented in Figure 5.4.

It is very likely that the tree search algorithm arrives at a stage where not all pairs of the current subject v_j can be assigned. We try to escape from these dead ends by applying techniques similar to the ones introduced in Subsection 5.4.2. More specifically, we reconsider the subject group assignments of the previous subject (*backjumping*) or restart with the bottleneck subject first.

There is one difference between backtracking, as defined in the previous section, and backjumping. Backtracking reconsiders the assignment of the direct predecessor $\text{pred}(p)$ of the current pair p, because these pairs are related: both pairs correspond to the same subject. Two consecutive subjects $\text{pred}(v)$ and v are not necessarily related. For example, they belong to different levels and, therefore, to different groups of students. Hence, we only jump back to subjects of the same level.

Above, we have described the major components of our tree search algorithm. The full tree search algorithm is given in Figure 5.5.

5.5 Maintaining a feasible complete timetable

For the solution approach to be effective, two conjectures must be true. First, to start the process, it must be easy to determine a feasible timetable if only a few subject group assignments have been carried out. More precisely, we must be able to construct a feasible complete timetable if student groups are assigned for subjects with one subject group only.

Conjecture 5.2. *Suppose* $dom(k_P) = \{(g,v) \in P \mid v \in V^{\text{sin}}\}$, *where* $V^{\text{sin}} = \{v \in V \mid |K_V(v)| = 1\}$. *Although determining a complete timetable* t_L *for which* $\mathsf{Feasible}(k_P, t_L)$ *holds is NP-hard, we can find a feasible complete timetable quickly for our real-life instances of* TSSGAP. □

Second, by reassigning a few lessons only, we must be able to repair the timetable quickly. That is, trying to repair an almost feasible timetable, by assigning several lessons to different time slots, is an effective and efficient way to find a feasible timetable for an extended subject group assignment $k_P \cup \{(p,k)\}$.

Conjecture 5.3. *Let* $\langle k_P, t_L \rangle$ *be a feasible partial solution of* TSSGAP. *Moreover, let* $p \notin dom(k_P)$ *be an unassigned pair* (g,v) *and* $k \in K_V(v)$ *be a possible subject group. Furthermore,* $k'_P = k_P \cup \{(p,k)\}$. *If* $\mathsf{Satisfies}(k'_P, t_L, ed \wedge ra \wedge ta \wedge ss) \wedge \neg\mathsf{Satisfies}(k'_P, t_L, st \wedge su \wedge te)$ *holds, then finding a complete timetable* t'_L *for which* $\mathsf{Feasible}(k'_P, t'_L)$ *holds is NP-hard. In many cases the following holds for complete timetables* t'_L*:*

$$\exists_{t'_L : L \to T} \mathsf{Feasible}(k'_P, t'_L) \;\Rightarrow\; \exists_{t'_L \in \mathcal{N}(t_L)} \mathsf{Feasible}(k'_P, t'_L),$$

where $\mathcal{N}(t_L)$ *denotes the set of complete timetables that are close to* t_L*, i.e., all complete time slot assignments* t'_L *for which* $|t'_L \,\Delta\, t_L|$*, the number of differences between* t_L *and* t'_L*, is small.* □

```
TREE SEARCH ALGORITHM()
// global variables: solution ⟨k_P, t_L⟩, sets K_P of possible subject groups for all pairs

//  k_P = k̂_P ∧ t_L = t̂_L ∧ Feasible(k_P, t_L) ∧ Complete(t_L)

i := 0;
repeat
   P^not := P − dom(k_P);
   (v_0, ..., v_{m−1}) := "the ordered sequence of all m subjects in P^not
      determined by subject ordering rule so";
   // ∀_{i≤0<m} P_i = { p ∈ P^not | v_P(p) = v_i }
   j := 0;
   restart := false;
   while j < m and (not restart) do
      // ∀_{p∈P_j} v_P(p) = v_j ∧ p ∉ dom(k_P)
      assigned := TRY TO ASSIGN ALL PAIRS(P_j);
      if assigned then
         P^not := P^not − P_j;
         j := j + 1;
      else
         // dead end
         if "do backjumping" then
            // bj = true
            "backjump to preceding subject of same level as v_j";
         else
            restart := true;
            if "restart with bottleneck subject first" then
               // bis = true
               "move bottleneck subject to front and fix position in ordering";
            "⟨k_P, t_L⟩ becomes the initial solution, i.e., ⟨k̂_P, t̂_L⟩";
            i := i + 1;
until Complete(k_P) or i > sr;

//  k_P ⊇ k̂_P ∧ Feasible(k_P, t_L) ∧ Complete(t_L)
```

Figure 5.5. Pseudocode for the tree search algorithm. For each subject, we try to assign all pairs by means of the algorithm described in Figure 5.4. The tree search algorithm is guided by four parameters. All subjects are ordered following rule *so*. Depending on the switches for backjumping (*bj*) and restarting with the bottleneck subject first (*bis*), we either undo the assignment of the preceding subject of the same level or restart from scratch with the bottleneck subject first. We restart the outer loop at most *sr* times.

We try to verify these conjectures by our experiments. Before we do that, we give the implementation of the heuristics used for constructing the initial feasible complete timetable and repairing the current complete timetable with resource conflicts.

5.5.1 Constructing an initial timetable

The initial timetable is a complete timetable that is feasible for the trivial subject group assignments. Several subjects have one single subject group, so the pairs corresponding to these subjects can be assigned in only one way. We denoted this subset of subjects by V^{sin}. Before we apply our tree search algorithm, we carry out the assignment of all pairs (g, v) for subjects $v \in V^{\text{sin}}$.

Now, assume $\text{dom}(k_{\text{P}}) = \{ (g, v) \in P \mid v \in V^{\text{sin}} \}$ and $\mathsf{Feasible}(k_{\text{P}})$ holds, i.e., the trivial assignments have been made and the corresponding subject group assignment is feasible. The resulting graph contains cliques corresponding to the teacher and subject group constraints (*te*) and (*su*), and the edges that represent the student group constraints (*st*) implied by the trivial assignments. In practice, there are only a few subjects with one subject group. Therefore, finding a feasible complete timetable for the trivial subject group assignments reduces to finding a feasible proper coloring in a graph with the largest cliques being the cliques induced by the teacher constraints. It is very likely that a simple graph coloring heuristic, e.g., one that orders vertices *highest degree first* [Welsh & Powell, 1967] or *most constrained first* [Brélaz, 1979], will find a feasible proper coloring.

We implement a variant of the DSATUR graph coloring heuristic defined in [Brélaz, 1979]. This heuristic gradually extends a feasible partial time slot assignment t'_{L} by assigning an unassigned lesson $l \in L^{\text{not}} := L - \text{dom}(t_{\text{L}})$ with minimum number of feasible time slots $| T^{\text{pos}}_{\text{L}}(l) |$ in each step. Ties are broken randomly. The chosen lesson l is given a random feasible color $t \in T^{\text{pos}}_{\text{L}}(l)$. In Figure 5.6, we give the pseudocode for this heuristic.

Recall that the set of possible time slots for a lesson is initially equal to the set of time slots at which the lesson's teacher is available, i.e., $T^{\text{pos}}_{\text{L}}(l) = T_{\text{D}}(d_{\text{L}}(l))$ for every lesson $l \in L$. But, as soon as lessons are assigned to time slots, these sets are reduced due to the constraints that must be satisfied. Whenever one set $T^{\text{pos}}_{\text{L}}(l)$ becomes empty for an unassigned lesson $l \in L^{\text{not}}$, the assignment of l becomes inconsistent with the current partial time slot assignment t'_{L}. In that case, we restart the algorithm from scratch. Since we randomize the choice of the most urgent lesson in case of ties and the time slot we assign to it, we construct a different time slot assignment in the next run.

5.5.2 Repairing a timetable with resource conflicts

What remains to be discussed is the way we maintain a feasible complete timetable after assigning an unassigned pair p to a subject group k. Remember that if $\mathsf{Feasible}(k_{\text{P}}, t_{\text{L}})$ and $\mathsf{Satisfies}(k_{\text{P}} \cup \{(p, k)\}, ss)$ hold, then $\mathsf{Satisfies}(k_{\text{P}} \cup$

CONSTRUCT INITIAL TIMETABLE()
// global variables: solution $\langle k_{\mathrm{P}}, t_{\mathrm{L}} \rangle$

// $k_{\mathrm{P}} = \hat{k_{\mathrm{P}}} \wedge \mathrm{dom}(k_{\mathrm{P}}) = V^{\mathrm{sin}} \wedge \mathsf{Satisfies}(k_{\mathrm{P}}, ss)$

repeat
  $t_{\mathrm{L}} := \emptyset$;
  $L^{\mathrm{not}} := L$;
  inconsistent := false;
  while $L^{\mathrm{not}} \neq \emptyset$ **and** (**not** *inconsistent*) **do**
    forall $l \in L^{\mathrm{not}}$ **do** $T_{\mathrm{L}}^{\mathrm{pos}}(l) := \{ t \in T \mid \mathsf{Feasible}(k_{\mathrm{P}}, t'_{\mathrm{L}} \cup \{(l,t)\}) \}$;
    if $\exists_{l \in L^{\mathrm{not}}}\ T_{\mathrm{L}}^{\mathrm{pos}}(l) = \emptyset$ **then**
      inconsistent := true;
    else
      $l := \mathrm{argmin}_{l' \in L^{\mathrm{not}}} \ \ | T_{\mathrm{L}}^{\mathrm{pos}}(l') |$;
      $t :\in T_{\mathrm{L}}^{\mathrm{pos}}(l)$;
      $t_{\mathrm{L}} := t_{\mathrm{L}} \cup \{(l,t)\}$;
      $L^{\mathrm{not}} := L^{\mathrm{not}} - \{l\}$;
until $\mathsf{Complete}(t_{\mathrm{L}})$;

// $k_{\mathrm{P}} = \hat{k_{\mathrm{P}}} \wedge \mathsf{Feasible}(k_{\mathrm{P}}, t_{\mathrm{L}}) \wedge \mathsf{Complete}(t_{\mathrm{L}})$

Figure 5.6. Heuristic for constructing the initial feasible complete timetable. All unassigned lessons L^{not} are ordered randomly. Moreover, the algorithm selects a feasible time slot randomly.

$\{(p,k)\}, t_L, ed \wedge ra \wedge ss \wedge ta)$ holds. That is, given a feasible solution $\langle k_P, t_L \rangle$, the only constraints that can be violated after assigning p to k are the resource constraints (*st*), (*su*), and (*te*).

Local search is a frequently used method is to find a timetable t_L^* with minimal number of resource conflicts $|F(k_P, t_L^*)|$. If a timetable without resource conflicts is found, then we have the desired feasible timetable. In our case, the local search heuristic replaces the current timetable t_L with a timetable t_L' that follows from reassigning one or more lessons in t_L. Such a change in the current timetable is called a *move*. To guide the search to a timetable with minimal resource conflicts, we select a next timetable t_L' that maximizes $|F(k_P, t_L)| - |F(k_P, t_L')|$, the decrease, or gain, in resource conflicts. Since we want that all timetables satisfy the educational, room availability, and teacher availability constrants, only moves that lead to those timetables are feasible.

A simple local search heuristic like best improvement [Aarts & Lenstra, 1997] stops when the gain for all feasible moves is non-positive. But if the resulting timetable t_L' still has resource conflicts, we may want to continue the search for timetables without resource conflicts. Tabu search algorithms [Aarts & Lenstra, 1997] choose a feasible move with maximum gain even if the gain for every feasible move is non-positive. Permitting deteriorations of the timetable may result in cycling, i.e., arriving at a timetable that was visited in a previous step. We reduce the risk of cycling by rendering the reverse move tabu during the next few iterations. We overrule the tabu status of a move if application of the tabu move leads to a timetable that improves the current best timetable.

We pick a best feasible non-tabu move m randomly and apply this move to the current timetable t_L'. The resulting timetable is denoted as $t_L' \oplus m$. The reverse move is put on a tabu list. These moves stay on the tabu list during a number of iterations. The tabu search algorithm stops when either the number of iterations has reached a given maximum *ni*, all neighbors are either infeasible or tabu, or a timetable without resource conflicts is found.

5.6 Improving the performance of generic methods

In practice, generic methods only perform well if problem specific knowledge is used. The techniques we introduced, like backtracking and bottleneck identification, can be applied to tree search algorithms for other problems than TSSGAP. So far, the only addition specific for TSSGAP has been the grouping of related pairs, i.e., pairs of the same subject, to improve upon standard chronological backtracking.

In this section, we describe how we try to increase the performance of tree search and tabu search in our approach. We do so by making problem specific choices for variable and value selection rules and the neighborhood of the tabu search procedure.

Note that we try to solve several NP-hard sub-problems in our procedures. Some-

TIMETABLE WITH LEAST RESOURCE CONFLICTS(solution $\langle k_P, t_L \rangle$)

// Complete(t_L) $\wedge$ Satisfies($k_P, t_L, ed \wedge ra \wedge ss \wedge ta$) $\wedge$ $|F(k_P, t_L)| > 0$

$t'_L := t_L$;
$t^*_L := t_L$;
repeat
  if "there exists a feasible non-tabu move" **then**
    move := "a best feasible non-tabu move";
    $t'_L := t'_L \oplus move$;
    if $|F(k_P, t'_L)| < |F(k_P, t^*_L)|$ **then**
      $t^*_L := t'_L$;
    "update tabu list";
until $|F(k_P, t^*_L)| = 0$ **or** "there exists no feasible non-tabu move"

// Complete(t^*_L) $\wedge$ Satisfies($k_P, t^*_L, ed \wedge ra \wedge ss \wedge ta$) $\wedge$ $|F(k_P, t^*_L)| \leq |F(k_P, t_L)|$

return t^*_L;

Figure 5.7. The tabu search algorithm that is used to find a timetable t^*_L with zero resource constraints for a given subject group assignment k_P. We start with the current complete timetable t_L that satisfies all but the resource constraints. In each step, we choose a best feasible non-tabu move randomly.

times it is necessary to trade off spending more time on a particular partial solution or continuing with a different or new solution. We do that by introducing additional parameters that bound the number of iterations, and experimenting with these parameters.

Variable and value selection. The performance of tree search algorithms clearly depends on the way one selects the next variable and value. Often, one picks the *most constrained* variable first. This is called the *fail first* principle, because if we are not able to assign this variable, it would be of no use to assign the other variables: the algorithm fails to find a feasible complete solution anyway. For the value selection, the *most promising* value is usually tried first. In our tree search algorithm, the variable and value selection are determined by the subject ordering rule *so*, the pair ordering and selection rules *po* and *ps*, and the subject group ordering rule *sgo*.

Three possible subject orderings are considered. The *most students first* (*MSF*) ordering considers subjects in order of nonincreasing number of students $\sum_{g \in G_V(v)} z_G(g)$. The opposite ordering, *least students first* (*LSF*), starts with subjects with the least students. Finally, we compare MSF and LSF with a random subject ordering (*RSO*).

Note that if the MSF ordering rule is chosen, then mandatory subjects are considered before optional subjects of the same level. On the other side, the LSF ordering starts with optional subjects, such that the subject group assignment for the mandatory subjects is affected by the way students are assigned for optional subjects. The LSF rule resembles the way planners at Dutch secondary schools determine the subject groups in the clustering approach; see Section 4.4.

We can set the pair ordering rule *po* and the pair selection rule *ps* such that our tree search algorithm follows the most constrained variable first principle. We always start by ordering P^{not}, the set of all unassigned pairs. We consider two possible orderings of pairs: a *largest student group first* ordering, which emphasizes that student groups have a size and is therefore denoted *siz*, and a random pair ordering for which we use the abbreviation *rnd* in the remainder.

Pairs are selected either statically or dynamically. If we choose the static selection rule, *sta*, then we pick the next pair according to the initial ordering; in this case the ordering is fixed. For the dynamic selection, *dyn*, we determine in each step the unassigned pair p that maximizes the number of new resource constraints

$$\min_{k \in K_P(p)} |E(k_P \cup \{(p,k)\})| - |E(k_P)|.$$

Recall that $K_P(p)$ equals the set of possible subject groups for pair p. Moreover, $E(k_P)$, as defined on page 59, is equal to the set of lessons $\{l_1, l_2\}$ with a resource constraint due to subject group assignment k_P. Ties are broken by the initial pair ordering.

Note that the backtracking and the bottleneck pair restarting mechanisms may conflict with the dynamic pair selection. Therefore, we choose to fix the position of

pairs that have been considered before or have been moved to the front. Due to this choice, the dynamic pair selection only applies to the set of pairs that have not been fixed in a previous step of the algorithm. That is, we first consider all fixed pairs in the given order. Then, if all fixed pairs have been assigned, we choose the next unassigned pair following the dynamic pair selection rule.

We evaluate subject groups according to an ordering that is determined per pair. We consider three possibilities. Apart from a random ordering, we have two rules that follow the *most promising first* principle. One rule selects the subject group that introduces the least new resource constraints first. In this case, we only pay attention to the number of new edges that are added to the graph, i.e., the subject groups in $K_{\mathrm{P}}(p)$ are ordered such that $|E(k_{\mathrm{P}} \cup \{(p,k)\})|$ is nondecreasing. Since this rule only considers resource constraints, we call this rule *cns*. The other rule takes resource conflicts, and therefore the current coloring, into account. This rule orders subject groups such that the number of monochromatic edges $|F(k_{\mathrm{P}} \cup \{(p,k)\}, t_{\mathrm{L}})|$ is nondecreasing. In the remainder, we denote this rule by *cnf*.

Backtracking and backjumping. Usually, it is only efficient to apply backtracking to pairs that have been assigned recently. Therefore, we restrict the total number of backtracks by a given maximum *bt*. The number of backjumps is limited by two upper bounds. First, the total number of backjumps may not exceed a given maximum *bj*. Second, the algorithm may not spend too many backjumps between the same subjects v_1 and v_2. If the upper bound on the number of backjumps from v_1 to v_2, *bjft*, is reached, then the algorithm jumps back to the predecessor of subject v_2.

Tabu search. A clever choice of the neighborhood structure, i.e., the set of moves to be evaluated, is a key decision in implementing local search algorithms. The structure must be such that it enables fast computation of the gain and feasibility of each move and the number of moves to be evaluated is not too large. Often, only reassignments of one single lesson, *reassign one* moves, are considered, see e.g., the papers by Alvarez-Valdes, Martin & Tamarit [1996], Costa [1994], Hertz [1992], Hertz & Robert [1998], and Hertz & De Werra [1987]. We denote such a *reassign one* move by $m = (l,t,t')$, where l is a lesson, $t = t_{\mathrm{L}}(l)$ is the current time slot of l, and t' is the new time slot to which l will be assigned. We reduce the set of moves $M(k_{\mathrm{P}}, t_{\mathrm{L}})$ for the current solution $\langle k_{\mathrm{P}}, t_{\mathrm{L}} \rangle$ by considering only lessons that currently cause a resource conflict:

$$M(k_{\mathrm{P}}, t_{\mathrm{L}}) = \{\, (l, t_{\mathrm{L}}(l), t') \mid l \in L \;\wedge\; \exists_{l' \in L}\, \{l, l'\} \in F(k_{\mathrm{P}}, t_{\mathrm{L}}) \;\wedge\; t' \in T - t_{\mathrm{L}}(l) \,\}.$$

Recall that the best move we select must satisfy all educational, room availability, and teacher availability constraints. Since these constraints decreases the number of moves, we consider an extension of the *reassign one* neighborhood. Apart from all feasible *reassign one* moves, we also evaluate a subset of all feasible *reassign two* moves. We restrict ourselves to those feasible *reassign two* moves that consist of one infeasible *reassign one* move with positive gain and a complementary *reassign one*

move that repairs the infeasibility. More specifically, we also consider all combinations of two *reassign one* moves $m_1 = (l_1, t_1, t_1')$ and $m_2 = (l_2, t_2, t_2')$ for which

$$m_1 \in M(k_P, t_L) \wedge \neg\mathsf{Satisfies}(t_L \oplus m_1, ed \wedge ra)$$
$$\wedge \quad |F(k_P, t_L)| > |F(k_P, t_L \oplus m_1)| \wedge \mathsf{Satisfies}(t_L \oplus m_1 \oplus m_2, ed \wedge ra \wedge ta)$$

holds.

Before we apply the tabu search algorithm, we construct a large clique in the graph represented by the edge set $E(k_P)$. The heuristic greedily extends a clique of vertices $\mathcal{Q}$ by adding a vertex outside $\mathcal{Q}$ with maximum degree that is connected to every vertex in $\mathcal{Q}$. If the size of this clique is larger than the number of colors, i.e., the number of time slots $|T|$, then no feasible proper coloring exists. Consequently, there is no feasible complete time slot assignment for the given subject group assignment k_P. In this case, we skip the tabu search algorithm and immediately conclude that the current solution $\langle k_P, t_L \rangle$ is infeasible.

Apart from the maximal number of iterations *ni*, the tabu search algorithm has two additional parameters. The first one, *nit*, gives the number of iterations that a move stays on the tabu list. That is, each reverse move is only removed after a fixed number of iterations. We restrict the number of iterations of the tabu search algorithm further by aborting the search when the number of resource conflicts has not decreased during the last *nni* iterations.

5.7 Conclusion

Our heuristic consists of two components, i.e., a tree search algorithm that extends the subject group assignment step by step, and a tabu search algorithm that repairs the current timetable if the subject group assignment introduces resource conflicts. The tabu search method is integrated in the tree search algorithm. That is, the main step of our algorithm first changes the subject group assignment and then updates the time slot assignment. In this way, we maintain a feasible complete timetable in each step.

First, the algorithm carries out all trivial subject group assignments and constructs a feasible complete timetable. We expect that such a timetable can easily be found. Then, it partitions all unassigned pairs into subsets of pairs for the same subject.

For each subject, we try to assign all pairs by considering the assignment for one student group at a time. If the current timetable remains feasible, we continue with the next unassigned pair for the same subject. In the other case, we try to repair the current timetable by assigning several lessons to different time slots. We conjecture that if a feasible complete timetable exists, then such a timetable can often be found by reassigning several lessons.

In case we are unable to assign a pair, we say we arrive in a dead end. Then we either undo the assignment of the previous pair (backtracking), or restart from the beginning with the pair we could not assign, i.e., the bottleneck pair. Only a

limited number of restarts are carried out. If we are unable to assign all pairs of the same subject, then we apply the same kind of techniques. That is, we undo the assignments for the previous subject (backjumping) or restart from scratch with the bottleneck subject as the first subject.

We restrict ourselves to four variants of our tree search algorithm. These four variants differ in the way the backjumping and backtracking switches *bj* and *bt* and the bottleneck switches *bis* and *bip* are set. First, we have a constructive variant *constr*, for which all four switches are "off". For this variant, each dead end immediately incurs a restart. Second, we have a variant for which only the backjumping and backtracking switches are "on". This variant is called *back*. Third, we also consider an algorithm variant called *bottle* for which only the bottleneck switches are "on". Finally, for the *full* variant all switches are "on". For this variant, we combine the two different ways of tackling dead ends.

We characterize an execution of the tree search algorithm, based on the tree search template we defined in this chapter, by means of the following sets of parameters: the algorithm variant *av*, the initial timetable, the tabu search parameters (*ni*, *nni*, and *nit*), the variable and value selection rules (*so*, *po*, *ps*, and *sgo*), the maximal number of restarts for the two main loops (*sr* and *pr*), and the upper bounds for backjumping and backtracking (*bj*, *bjft* and *bt*). We summarize these parameters and their possible values in Table 5.1. Several choices in the procedures of the tree search algorithm are random. Therefore, a deterministic run of the tree search algorithm is completely specified by the parameter settings and the value of the random seed.

category	parameter	meaning	range
backjumping and backtracking			
subject	*bj*	max # backjumps	$\mathbb{N}$
	bjft	max # backjumps between two subjects	$\mathbb{N}$
pair	*bt*	max # backtracks	$\mathbb{N}$
dead end handling technique			
variant	*av*	way to handle dead ends	{ back, bottle, constr, full }
maintaining a feasible complete timetable			
construction		initial time slot assignment	
repair	*ni*	max # iterations	$\mathbb{N}$
	nni	max # iterations w/o improvement	$\mathbb{N}$
	nit	# iterations on tabu list	$\mathbb{N}$
restarts			
subject	*sr*	max # restarts subject loop	$\mathbb{N}$
pair	*pr*	max # restarts pair loop	$\mathbb{N}$
variable and value selection			
subject	*so*	initial subject ordering	{ LSF, MSF, RSO }
pair	*po*	initial pair ordering	{ rnd, siz }
	ps	selection rule	{ dyn, sta }
subject group	*sgo*	initial subject group ordering	{ cnf, cns, rnd }

Table 5.1. Parameters of tree search template and the range of parameter values. A deterministic run of the tree search algorithm is completely specified by the settings of all parameters and a value of the random seed.

6

Experimental results

In this chapter, we report on the experiments that we carried out with the algorithm we presented in the previous chapter. By carrying out a large number of experiments, we try to solve a number of real-life instances. We also perform a sensitivity analysis on the algorithm's parameters. Part of the results already appeared in [Willemen & Ten Eikelder, 2002].

We focus on two main themes. First, we try to find a suitable parameter setting for the tree search algorithm. That is, we look for a setting that solves the most instances. Here, we also try to make a distinction between easy and tough instances. Second, we want to know how well we can approximate a feasible complete solution for tough instances of TSSGAP. That is, we consider relaxations and sub-instances of TSSGAP and find out how close we can get to feasible complete solutions for the tough instances of TSSGAP.

The organization of this chapter is as follows. In Section 6.1, we describe the setup of our experiments. We present the instances we try to solve and the main questions we like to answer. We also describe the experiments and the order in which we carry these out. The report on the experiments is split into two parts. In Section 6.2, we take the instances as a constant factor and give results of the tree search algorithm for different parameter settings. Then, in Section 6.3, we fix the parameter setting and give the results for relaxations and sub-instances of TSSGAP. We conclude with a summary of all results in Section 6.4.

6.1 Experimental setup

This section gives the introduction to the experimental results. First, we present the instances that we use in our study. We show how we model several real-life timetabling problems at two Dutch secondary schools as school data, student group binding, and constraints for TSSGAP. Next, we give an overview of the experiments we want to carry out. That is, we present the key questions we like to answer by means of the experiments. Since one main topic is to find a suitable parameter setting for the tree search algorithm, we discuss how we compare the performance of the algorithm for different parameter settings.

6.1.1 Instances

School data. We consider two Dutch secondary schools: the Cambreurcollege in Dongen (*CD*) and the Pleincollege Sint-Joris in Eindhoven (*JE*). These two schools offer education in the school types MAVO, HAVO, and VWO. We focus on the upper years, i.e., those years in which students have freedom to compose their curriculum. Our data originates from 1998. The Cambreurcollege is a merge of several schools, among others one MAVO school in Rijen, which is a village nearby Dongen, a MAVO school in Dongen, and a combined HAVO/VWO school in Dongen. In 1998, all MAVO students from Rijen were still taught in Rijen and about ten teachers had to travel between the two villages. Therefore, we have two different *mavo4* levels, *dmavo4* and *rmavo4*, corresponding to the Dongen and Rijen branch of *mavo4*.

At the Pleincollege Sint-Joris, there are lessons in which students of different levels are taught simultaneously. First, the number of VWO students who have chosen Latin (*la*) is very small, such that school management had combined lessons in this language for the three VWO levels *vwo4*, *vwo5*, and *vwo6*. We model this by introducing an artificial level *vwo*, and a subject *la vwo* with only one subject group and five lessons. Students of the VWO levels who had chosen Latin have the subject *la vwo* in their curricula. Second, fourth-year students of MAVO and HAVO follow the same lessons of the two Arts subjects *tf* and *mf*. Therefore, we introduce a dummy level *maha4* for these two subjects.

Both schools have eight time slots per working day, resulting in forty time slots for one week. The Cambreurcollege has approximately eighty rooms, of which fifteen are in Rijen, whereas the Pleincollege Sint-Joris has fifty rooms in total.

Student group binding. We achieve student group binding, as defined on page 6, in two different ways. First, we model students with exactly the same curriculum as one student group. Second, for each level, we section all student groups in the same way for all mandatory subjects with the same number of subject groups. As a consequence, a student has exactly the same lessons as other students with the same program and sees the same group of fellow students during lessons in mandatory subjects.

For some levels, viz. *CDath4*, *JEmavo3*, and *JEvwo4*, the number of different

Cambreurcollege						Pleincollege Sint-Joris					
level	$\|L\|$	$\|S\|$	$\|V\|$	$\|K\|$	$\|D\|$	level	$\|L\|$	$\|S\|$	$\|V\|$	$\|K\|$	$\|D\|$
dmavo4	94	74	13	25	18	*mavo3*	99	69	16	42	27
rmavo4	72	52	13	19	13	*mavo4*	111	95	14	32	24
havo4	142	107	17	42	25	*havo4*	136	115	19	42	32
havo5	100	86	15	28	22	*havo5*	90	91	16	26	24
ath4	77	60	15	31	19	*vwo4*	67	53	18	29	24
ath5	100	67	17	33	21	*vwo5*	78	55	17	25	21
ath6	85	69	15	28	21	*vwo6*	66	39	15	19	18
						maha4	2	-	1	1	1
						vwo	5	-	1	1	1
ALL	670	515	105	206	64	*ALL*	654	517	117	217	66

Table 6.1. School data per level and for all levels. Given are the number of lessons $|L|$, students $|S|$, subjects $|V|$, subject groups $|K|$, and teachers $|D|$. The levels *maha4* and *vwo* are artificially created levels.

curricula is very small. While constructing the student group binding for these levels, we should take care that a student group g is not too large to satisfy the subject group size constraints, i.e., $z_{\mathrm{G}}(g) \leq \min_{v \in V_{\mathrm{G}}(g)} \overline{S_{\mathrm{V}}}(v)$ must hold. We decide to split a student group that is too large into several equal-sized student groups. To increase flexibility for the subject group assignment with regard to the subject group size constraints, we split every student group g for which

$$z_{\mathrm{G}}(g) > \left\lfloor \frac{1}{2} \min_{v \in V_{\mathrm{G}}(g)} \overline{S_{\mathrm{V}}}(v) \right\rfloor =: z$$

holds into $n = \lceil z_{\mathrm{G}}(g)/z \rceil$ groups: $nz - z_{\mathrm{G}}(g)$ student groups of size $z-1$ and $n - (nz - z_{\mathrm{G}}(g))$ student groups of size z.

In Table 6.2, we show the effect of student group binding on the number of decision variables. We distinguish variables for trivial assignments, i.e., pairs for optional subjects with a single subject group, from those for non-trivial decisions. For our instances, all mandatory subjects have more than one subject group. The largest reduction is achieved for *CDath4* and *JEvwo4*. Here the number of non-trivial decisions is decreased from 759 to 67 (-91%) and from 579 to 22 (-96%) respectively. These levels have relatively more mandatory subjects and students are more restricted in composing their program. Note that we obtain a large reduction in decision variables for the largest instances as well: 62% less variables for *CDALL* and 77% for *JEALL*. Note that for the artificial levels *JEvwo* and *JEmaha4*, all subjects have only one subject group. Consequently, for these levels all pairs are assigned in the initialization step. Therefore, we do not consider these instances in the remainder of the thesis.

Constraints. Three constraint types contain a parameter denoting the upper bound that must be met. When the parameter is set at low values, the constraints

	without group binding					with group binding				
instance	$\|S\|$	$\|P^{\text{man}}\|$	$\|P^{\text{opt}}\|$	$\|P^{\text{sin}}\|$	$\|P^{\text{tot}}\|$	$\|G\|$	$\|P^{\text{man}}\|$	$\|P^{\text{opt}}\|$	$\|P^{\text{sin}}\|$	$\|P^{\text{tot}}\|$
CDath4	60	540	219	21	780	12	24	43	5	72
CDath5	67	335	261	99	695	48	96	176	82	354
CDath6	69	207	225	141	573	48	96	148	106	350
CDdmavo4	74	148	300	96	544	43	43	165	64	272
CDhavo4	107	535	324	134	993	61	122	177	85	384
CDhavo5	86	258	229	138	625	58	116	147	105	368
CDrmavo4	52	104	174	100	378	30	30	96	63	189
CDALL	515	2127	1732	729	4588	300	527	952	510	1989
JEhavo4	115	575	320	160	1055	73	73	197	112	382
JEhavo5	91	273	181	184	638	64	64	112	144	320
JEmavo3	69	759	160	46	965	12	12	26	9	47
JEmavo4	95	380	290	94	764	49	98	141	58	297
JEmaha4	0	0	0	18	18	0	0	0	15	15
JEvwo	0	0	0	3	3	0	0	0	3	3
JEvwo4	53	530	49	131	710	13	13	9	34	56
JEvwo5	55	275	80	207	562	37	74	53	142	269
JEvwo6	39	78	46	190	314	30	30	38	144	212
JEALL	517	2870	1126	1033	5029	278	364	576	661	1601

Table 6.2. Effect of group binding on the number of decision variables, i.e., pairs. Students with exactly the same curriculum and mandatory subjects with the same number of subject groups are combined. We give the number of students $|S|$ and student groups $|G|$. The set of all pairs P^{tot} is divided into three subsets: those for mandatory subjects (P^{man}), optional subjects with more than one subject group (P^{opt}), and optional subjects with one subject group (P^{sin}).

are tight, possibly excluding solutions that are considered acceptable for planners at schools. We do not have information on how planners would set these parameters. Moreover, for our instances, we do not have the subject group assignments obtained by the planners. If we did have this data, then we could have determined values for $\overline{S_{\mathrm{V}}}(v)$ for which the planners' solution would be feasible. Therefore, we choose the parameter values such that at least every individual constraint of TSSGAP is satisfiable for every instance.

In principle, the maximum number of lessons per day for a subject group $k \in K$, $\overline{L_{\mathrm{K}}}(k)$, is one. Some subject groups *must* have at least two lessons at the same day. For example, those subject groups k for which

$$|\, L_{\mathrm{K}}(k) \,| > |\, \cup_{d \in D_{\mathrm{K}}(k)} W_{\mathrm{D}}(d) \,|$$

holds. Here, $D_{\mathrm{K}}(k) := \cup_{l \in L_{\mathrm{K}}(k)} \{d_{\mathrm{L}}(l)\}$ is the set of teachers for subject group k, and $W_{\mathrm{D}}(d) := \cup_{t \in T_{\mathrm{D}}(d)} \{w_{\mathrm{T}}(t)\}$ is the set of days at which teacher d is available. Thus, the parameter $\overline{L_{\mathrm{K}}}(k)$ must be set at larger values for every subject group with more lessons than days available for teaching these lessons. We want to have no more lessons on the same day than strictly necessary. Therefore, we give $\overline{L_{\mathrm{K}}}(k)$ the smallest possible value. For all our instances, it is sufficient to set it equal to two.

The room constraints depend on the parameter that denotes the number of rooms available at each time slot. Again, we can choose values such that the room availability constraints are tight. For each instance, we can divide the number of lessons $|\, L \,|$ by the number of time slots $|\, T \,|$, and round it to the nearest integer greater than this value. On the other side, if the instance we want to solve represents the first part of the final timetable under construction, then every room is still available. This is usually the case when planners schedule the upper years. As this is also our objective, we set the number of rooms equal to the number of rooms available at the school under consideration.

All upper bounds on the subject group size are given by

$$\overline{S_{\mathrm{V}}}(v) := \left\lceil \frac{\sum_{g \in G_{\mathrm{V}}(v)} z_{\mathrm{G}}(g)}{|\, K_{\mathrm{V}}(v) \,|} \right\rceil + sstol,$$

where *sstol* is a new constraint parameter: the subject group size tolerance. Thus, we replace $|\, V \,|$ parameters by only one and require that every subject group of a subject $v \in V$ may exceed the optimal subject group size for v by the given tolerance only. If $sstol = 0$, then all subject groups must be perfectly balanced. We expect that this choice is far too restrictive. Therefore, we set $sstol = 3$ for all instances.

6.1.2 Experiments and performance characteristics

An *experiment* is completely specified by a problem type, a parameter setting, a set of instances, the number of independent runs of the algorithm on each instance, and a set of performance characteristics. The problem type is in general TSSGAP, but we also consider variants of TSSGAP for which one or more constraint types have

been removed. In Section 5.7, we presented the parameters of the algorithm and the ranges for these parameters. Since the tree search algorithm makes random choices, we perform more than one run with different random seeds for the same parameter setting. Finally, we consider the following *performance characteristics*.

- The number of instances solved.
- The number of successful runs for an instance, i.e., the number of runs that lead to a feasible complete solution of the instance.
- The average CPU-time of all runs, of all successful runs, and of all unsuccessful runs for an instance.
- The minimum number of unassigned pairs for an instance, which is zero if at least one of the runs was successful.
- The estimated time to success ETS for an instance, defined as

$$ETS := \frac{1-p}{p} I\!E\left\{\tau^{\mathrm{f}}\right\} + I\!E\left\{\tau^{\mathrm{s}}\right\},$$

  where p is the probability that a run is successful and $I\!E\left\{\tau^{\mathrm{f}}\right\}$ and $I\!E\left\{\tau^{\mathrm{s}}\right\}$ are the estimated CPU-times of a unsuccessful run and a successful run, respectively. Since we do not know the exact values of p, $I\!E\left\{\tau^{\mathrm{f}}\right\}$, and $I\!E\left\{\tau^{\mathrm{s}}\right\}$, we have to estimate these values. We do that as follows:

$$\begin{aligned} \hat{p} &:= \#\text{ successful runs divided by the total \# runs,} \\ \hat{\tau}^{\mathrm{f}} &:= \text{average CPU time of all unsuccessful runs, and} \\ \hat{\tau}^{\mathrm{s}} &:= \text{average CPU time of all successful runs.} \end{aligned}$$

  This gives the following approximation for the expected time to success:

$$ETS \approx \frac{\text{total CPU time for all runs}}{\#\text{ successful runs}}.$$

All runs were performed on a Linux-PC with a Pentium III-EB, 533 MHz, processor and 256 MB internal memory. The algorithm's executable was obtained by `g++`.

6.1.3 Outline of experiments

We are interested in solving instances of TSSGAP and approximating a feasible complete solution for tough instances by means of the full tree search algorithm. The first theme introduces the following key questions.

1. Are all our instances feasible, i.e., does there exist a feasible complete solution for all our instances?
2. What is a suitable parameter setting for the full tree search algorithm? Does there exist a parameter setting for which all our instances are solved? How sensitive is the tree search algorithm with regard to the parameter setting?
3. Are all instances routinely solved? Or are there also tough instances for which only a few runs are successful?

4. Do we need both kinds of dead end handling? Or can we already obtain good results without backjumping, backtracking, or bottleneck identification?

For the instances that we cannot solve routinely, we want to know how close we can get to a feasible complete solution. We approximate a feasible complete solution by (1) giving up the requirement that all pairs must be assigned, (2) removing all constraints of one constraint type, (3) relaxing constraints, and (4) trying to find solutions for sub-instances.

Below, we describe the details of the experiments and the order in which we carry out these experiments.

Find suitable parameter setting for full algorithm. We distinguish four kinds of parameters. First, the parameter *av* determines the way dead ends are tackled. Second, we have parameters that give upper bounds on the number of iterations within the pair and subject loops of the algorithm: *sr*, *pr*, *bj*, *bjft*, and *bt*. We expect that increasing the value of these parameters leads to higher average CPU-times. Moreover, if we are able to solve an instance for given values of the upper bounds, we expect to solve it for higher values as well. Third, we have the variable and value selection rules *so*, *po*, *ps*, and *sgo*. The effect of these parameters on the CPU-time and solution rate is not clear. Fourth, there are parameters that deal with maintaining a feasible complete timetable in each step. We expect that increasing the upper bounds on the number of iterations *ni* and *nni* increases the CPU-time but does not deteriorate the solution rate. The effect of the number of iterations *nit* on the tabu list is less straightforward. Usually, this parameter is set neither too low nor too high.

We start with choosing good values for the tabu search parameters and the variable and value selection rules. This choice is based on earlier experiments with the constructive heuristic *constr*. We let the number of instances solved determine the best parameter choices.

Next, we concentrate on the full algorithm and set all other parameter values at their lowest nonnegative values. Based on the results obtained from this experiment, we make a distinction between instances that are routinely solved and tough instances.

We continue with carrying out experiments for the tough instances in several steps. In each step, we perform a number of experiments in which each upper bound is increased independently to a higher value. Based on a comparison of the number of instances solved and, in case every run was unsuccessful, the number of unassigned pairs, we choose a best parameter setting and continue with the next step. We continue until the experiments are too much time-consuming or no further improvement is made. An experiment is considered to be too much time-consuming if all runs for the tough instances take more than one day of CPU-time on our network of 16 independent PCs.

In this way, we end with a parameter setting that is considered suitable. Apart from the performance characteristics for this parameter setting, we also show the

performance for settings which have one parameter set to a slightly smaller or larger value.

We finish with some experiments in which we vary the variable and value selection rules and tabu search parameters. Recall that we started by fixing these parameters. Now, we fix the values for the upper bounds. In this way, we can perform a sensitivity analysis for the variable and value selection rules and the tabu search parameters.

Compare dead end handling techniques. We ask ourselves which way of dead end handling pays off. Therefore, we compare the four variants *constr*, *back*, *bottle*, and *full* for the parameter values we found in the previous experiments.

The variants without backjumping and backtracking, i.e., *constr* and *bottle*, immediately restart the pair or subject loop at dead ends. We expect that these variants have lower average CPU-times but are less effective in solving the instances.

We compare the variants on the number of instances solved, the number of successful runs, the average CPU-time and the estimated time to success. The latter characteristic is an indication of the efficieny of the variant.

Approximate a feasible complete solution of TSSGAP. It is interesting to see how well we can approximate a feasible complete solution for tough instances, i.e., instances for which at least one run was unsuccessful. We discuss several ways to approximate such a solution. For example, we give up the curriculum constraints (*cu*), i.e., accept incomplete but feasible subject group assignments, and try to assign as many pairs as possible. If only a few pairs are left, a planner could analyze the solution, identify one or more causes for failure, and make manual decisions to overcome the immediate problems. Also, if the pairs correspond to optional subjects, then school management may decide to reject the student choices for these optional subjects and require the affected students to change their curricula.

We also remove or relax constraints of some type and try to solve these variants. In this way, we may be able to identify which constraints of TSSGAP are hard to satisfy by our tree search algorithm. We can relax all constraints of one type by increasing the upper bound parameters $\overline{L_{\mathrm{K}}}(k)$, $|R|$, or $\overline{S_{\mathrm{V}}}(v)$.

Finally, we try to find solutions for sub-instances of the tough instances. Note that the two largest instances, *CDALL* and *JEALL*, comprise all smaller instances of the same school as sub-instances. We restrict ourselves to sub-instances that follow after removal of one or more levels and all related decisions and constraints. In this way, we try to identify bottleneck levels that should be tackled before the other levels.

For all these experiments, we execute the *full* algorithm and use the parameter setting we found in the previous experiments.

6.2 Solving TSSGAP

Determine values of tabu search parameters. We first determine appropriate values for the tabu search parameters. In Table 6.3, we give the performance of ten runs of a variant of the constructive heuristic for different values of the tabu search parameters. Instead of restarting at a dead end, we skip the pair that cannot be assigned and continue the algorithm. We do not group all pairs of the same subject. Moreover, we order pairs and subject groups randomly. All runs stopped within a few minutes of CPU-time. We arrive at the following conclusions.

- It is useful to repair the timetable, since no instance is solved when we skip the repair step ($ni = nni = 0$).
- The setting $ni = 1000$, $nni = 100$, and $nit = 10$ seems to be a reasonable choice since little improvement of performance is observed for higher values.
- The instances with the least pairs, *JEvwo4*, *JEmavo3*, *CDath4*, and *JEvwo6* can already be solved and the majority of the runs are successful.

Therefore, we set the tabu search parameters as follows: $ni = 1000$, $nni = 100$, and $nit = 10$.

Determine variable and value selection rules. Since there are too many combinations of the variable and value selection rules *so*, *po*, *ps*, and *sgo*, we also first determine good values for these parameters. Again, the choice is based on experiments with the simplest algorithm, i.e., the constructive heuristic *constr*. We run the algorithm ten times for 48 different parameter settings. These settings comprise all $3 \times 2 \times 2 \times 3 = 36$ possible settings of the variable and value selection rules. For comparison, we also added a column reporting on the number of successful runs when we do not group all pairs for the same subject (*so* = none). The total number of successful runs for each value is given in Table 6.4. We draw the following conclusions.

- Considering the number of instances solved, a random subject and pair ordering (*so* = RSO, *po* = rnd), dynamic pair selection (*ps* = dyn), and ordering subject groups according to nonincreasing number of resource conflicts (*sgo* = cnf) is the best parameter choice.
- Based on the number of successful runs, MSF dominates RSO for *so*, siz dominates rnd for *po*, sta dominates dyn for *ps*, and cns dominates cnf for *sgo*.
- Again, the instances with the smallest number of pairs (*JEvwo4*, *JEmavo3*, *CDath4*, and *JEvwo6*) are easily solved.
- *JEvwo5* seems to be a small but tough instance, because instances with more pairs were solved.

Therefore, we set the variable and value selection rules equal to *so* = RSO, *po* = rnd, *ps* = dyn, and *sgo* = cnf.

Determine values of upper bound parameters. We continue our experiments with the *full* tree search algorithm, i.e., the variant that includes backjumping, back-

		ni	0	10	100	1000	10000	100000
		nni	0	10	10	100	1000	10000
		nit	0	10	10	10	10	10
instance	pairs	groups						
JEvwo4	22	2	(13)	**9**	**9**	**9**	**9**	**9**
JEmavo3	38	4	(13)	**2**	**5**	**7**	**8**	**8**
CDath4	67	7	(15)	(1)	**2**	**6**	**6**	**6**
JEvwo6	68	3	(51)	**2**	**1**	**3**	**5**	**5**
CDrmavo4	126	5	(51)	(3)	(1)	(3)	(4)	(3)
JEvwo5	127	4	(66)	(19)	(15)	(10)	(3)	(17)
JEhavo5	176	5	(85)	(4)	(3)	(3)	(4)	(4)
CDdmavo4	208	7	(72)	(12)	(9)	(6)	(4)	(3)
JEmavo4	239	7	(82)	(11)	(3)	(7)	(3)	(3)
CDath6	244	8	(67)	(7)	(8)	(4)	(6)	(6)
CDhavo5	263	7	(91)	(9)	(9)	(8)	(3)	(6)
JEhavo4	270	7	(120)	(32)	(21)	(22)	(18)	(18)
CDath5	272	9	(108)	(16)	(22)	(14)	(12)	(10)
CDhavo4	299	8	(122)	(39)	(42)	(36)	(36)	(32)
JEALL	940	32	(435)	(91)	(90)	(65)	(50)	(54)
CDALL	1479	51	(532)	(150)	(148)	(114)	(83)	(93)

Table 6.3. Performance of ten runs of a variant of the constructive heuristic on all sixteen instances of TSSGAP for different values of tabu search parameters *ni*, *nni*, and *nit*. The variable and value selection rules are (none,rnd,sta,rnd), there are no restarts, backtracks, or backjumps. Instead of restarting at a dead end, we skip the unassigned pair and continue. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number of unassigned pairs between parentheses.

			so				*po*		*ps*		*sgo*			
instance	pairs	groups	none (120)	LSF (120)	MSF (120)	RSO (120)	rnd (240)	siz (240)	dyn (240)	sta (240)	cnf (160)	cns (160)	rnd (160)	total
JEvwo4	22	2	119	119	120	120	238	240	240	238	160	160	158	478
JEmavo3	38	4	109	114	112	107	211	231	220	222	155	149	138	442
CDath4	67	7	87	86	76	69	144	174	140	178	124	98	96	318
JEvwo6	68	3	59	18	86	48	110	101	108	103	100	42	69	211
CDrmavo4	126	5	1	1	14	6	6	16	8	14	7	13	2	22
JEvwo5	127	4												0
JEhavo5	176	5	1			4	2	3	3	2	4		1	5
CDdmavo4	208	7		1		2	3		1	2	3			3
JEmavo4	239	7	2	1	6	6	12	3	9	6	12	1	2	15
CDath6	244	8												0
CDhavo5	263	7			1		1		1		1			1
JEhavo4	270	7												0
CDath5	272	9												0
CDhavo4	299	8												0
JEALL	940	32												0
CDALL	1479	51												0
		total	378	340	415	362	727	768	730	765	566	463	466	

Table 6.4. Number of successful runs of *constr* algorithm for all sixteen instances of TSSGAP and different variable and value selection rules. For comparison, we added a column for the results without grouping (*so* = none). We do not restart the subject or pair loops and do not perform backjumps or backtracks. The tabu search parameters are: $ni = 1000$, $nni = 100$, and $nit = 10$. If all runs were unsuccessful, we left the cell void. We perform ten runs for each setting, so there are 480 runs for each instance. We give the total number of runs for each parameter value between parentheses.

solved instances			
instance	pairs	groups	success
JEvwo4	22	2	10
JEmavo3	38	4	10
CDath4	67	7	10
JEvwo6	68	3	10
CDrmavo4	126	5	10
JEvwo5	127	4	6
JEhavo5	176	5	10
CDdmavo4	208	7	10
JEmavo4	239	7	10
CDath6	244	8	10
CDhavo5	263	7	10
JEhavo4	270	7	1

not solved instances			
instance	pairs	groups	unassigned
CDath5	272	9	13
CDhavo4	299	8	25
JEALL	940	32	65
CDALL	1479	51	245

Table 6.5. Performance of ten runs of the *full* algorithm on all sixteen instances of TSSGAP for the parameter setting with smallest nonnegative upper bounds, i.e., $sr = pr = bj = 5, bjft = 1, bt = 25$. Tabu search parameters and variable and value selection rules are at their fixed values. Given are the number of successful runs for solved instances and the minimum number of unassigned pairs for unsolved instances.

tracking, and restarting with the bottleneck subjects and pairs first. Given the fixed settings for the tabu search parameters and variable and value selection rules, we try to find good values for the maximal number of restarts of the subject and pair loops (*sr* and *pr*), the maximal number of backjumps in total and between two subjects of the same level (*bj* and *bjft*), and the maximal number of backtracks *bt*. First, we set all parameters at the the smallest values ($sr = pr = bj = 5$, $bjft = 1$, $bt = 25$). We run the *full* algorithm ten times. Only four instances remain unsolved: *CDath5*, *CDhavo4*, *JEALL* and *CDALL*. Except for *JEvwo5* and *JEhavo4*, all other instances are solved each of the ten runs. We summarize the results in Table 6.5.

Now, we focus our attention on the tough instances. That is, in the remainder, we only perform the experiments for the four instances *CDath5*, *CDhavo4*, *JEALL*, and *CDALL* that we did not solve yet and the instances *JEvwo5* and *JEhavo4* for which at least one run was unsuccessful.

We carry out a large number of experiments with increasing values for the upper bounds and stop when we cannot improve upon the setting without consuming too much CPU-time for an experiment. This leads to the parameter setting $sr = 15, pr = 10, bj = 5, bjft = 2, bt = 25$, for which we can solve all instances but *CDALL*.

In Table 6.6, we give the performance characteristics for this parameter setting. For the sake of completeness, we also include the results for the easy instances. As expected, the easy instances are always solved, and for the tough instances we only

				not assigned		CPU time (min)			
instance	pairs	groups	succ	pairs	subjects	total	succ	fail	ETS
JEvwo4	22	2	**10**			<1	<1	-	<1
JEmavo3	38	4	**10**			<1	<1	-	<1
CDath4	67	7	**10**			<1	<1	-	<1
JEvwo6	68	3	**10**			4	4	-	4
CDrmavo4	126	5	**10**			7	7	-	7
JEvwo5	127	4	**7**	0.8	0.3	28	24	39	41
JEhavo5	176	5	**10**			30	30	-	30
CDdmavo4	208	7	**10**			44	44	-	44
JEmavo4	239	7	**10**			4	4	-	4
CDath6	244	8	**10**			92	92	-	92
CDhavo5	263	7	**10**			130	130	-	130
JEhavo4	270	7	**4**	9.4	0.6	269	109	376	673
CDath5	272	9	**1**	23.8	1.4	216	154	224	2166
CDhavo4	299	8	**1**	37.6	1.4	388	200	409	3881
JEALL	940	32	**2**	93.3	3.9	531	362	573	2654
CDALL	1479	51	(149)	417.8	15.1	705	-	705	$+\infty$

Table 6.6. Performance of ten runs of the *full* algorithm on all sixteen instances of TSSGAP for the suitable parameter setting (variable and value selection rules: (RSO,rnd,dyn,cnf), restarts: (15,10), backtrack and backjump parameters: (5,2,25), and tabu search parameters: (1000,100,10)). We give the number of successful runs in boldface or the minimum number of unassigned pairs between parentheses, the average number of unassigned pairs, the average number of not completely assigned subjects, the average CPU-times in minutes for all runs, all successful runs, and all unsuccessful runs, and, finally, the estimated time to success in minutes.

have a few successful runs. Almost all successful runs for tough instances take less than six hours, whereas unsuccessful runs usually take longer, sometimes up to 12 hours of CPU-time. Performing more than one run is necessary to solve the tough instances. Considering the estimated time to success, an algorithm that solves an instance by performing as many runs as necessary may take up to as much as 65 hours! Interestingly, *JEvwo5* is tougher than might be expected if we consider its size. On contrary, *JEmavo4* is much easier than expected.

Try slightly smaller or larger values for upper bounds. We show that the chosen parameter setting is locally optimal by presenting the results of experiments with smaller and larger values for the upper bounds in Table 6.7. In these tables, the first entry is either the number of successful runs or the smallest number of not assigned pairs are given. The average CPU-times in hours for each parameter set is given as the second entry. Finally, the third value in each cell is the estimated time to success in hours. Note that the parameters *bj* and *bt* are at the lowest value already. We draw the following conclusions.

- If we only focus on the number of instances we solved, the parameter setting

$sr = 15$, $pr = 10$, $bj = 5$, $bjft = 2$, and $bt = 25$ seems to be suitable, since for this setting we were able to solve instances that are not solved for smaller or larger values.

- For *CDath5* and *CDhavo4*, it seems that we have only one lucky run that solves the instances, which we cannot repeat by setting at larger values.
- *CDath5* and *CDhavo4* seem to be extremely tough instances.
- For *JEhavo4* and *JEALL*, larger values of the upper bound parameters lead to more successful runs and a shorter estimated time to success.

Try different variable and value selection rules. We perform a sensitivity analysis on the parameters we set at the very beginning. First, we vary the variable and value selection rules. Instead of carrying out experiments for all possible combinations of variable and value selection rules, we restrict ourselves to the two best values per parameter. That is, for each parameter we select the two values from Table 6.4 for which most instances were solved. We present the results in Table 6.8. We conclude that our choice for *so* = RSO, *po* = rnd, *ps* = dyn, and *so* = cnf is reasonable, considering the number of instances solved.

Try different initial timetable and tabu search parameters. We reconsidered the parameters for maintaining a feasible complete timetable in each step. First, we tried different initial timetables and run the algorithm with the suitable parameter setting. We did not observe a large difference in performance of the algorithm for a different initial timetable. Second, we set the tabu search parameters at smaller and larger values. In Table 6.9, we give the performance of the algorithm for these experiments. Again, we conclude that the original setting is a reasonable choice. Interestingly, decreasing the maximal number of iterations *ni* leads to an *increase* in the average CPU-time, which is the opposite of what we expected. Moreover, the CPU-time strongly depends on the maximal number of non-improving moves *nni*. This gives an indication that the tabu search routine often stops when this bound is reached.

Compare different dead end handling techniques. In Table 6.10, we compare the performance of the four variants *constr*, *back*, *bottle*, and *full* on all instances 16 instances of TSSGAP. The *full* algorithm clearly dominates all others on the number of instances solved and the number of successful runs per instances. Hence, *full* is more robust than the other three with regard to the random choices. Adding bottleneck identification to *back* clearly pays off. Tough instances like *CDath5*, *CDhavo4*, and *JEALL* are only solved by *full*, and *full* is more efficient in solving *JEvwo5* and *JEhavo4*. For easier instances, *constr* and *bottle* are more efficient than *full*. In these situations, *full* probably spends too much time in exploring branches that do not lead to feasible complete solutions.

		sr	***10***	15	15	15	***20***	15	15	15	15
		pr	10	***5***	10	10	10	***15***	10	10	10
		bj	5	5	5	5	5	5	***10***	10	10
		bjft	2	2	***1***	2	2	2	2	***3***	2
		bt	25	25	25	25	25	25	25	25	***50***
instance	pairs	groups									
JEvwo5	127	4	**7**	**4**	**5**	**7**	**7**	**5**	**3**	**4**	**5**
			29m	22m	33m	29m	29m	52m	59m	42m	47m
			0h41	0h56	1h05	0h41	0h42	1h44	3h15	1h45	1h33
JEhavo4	270	7	**4**	**4**	**7**	**4**	**4**	**7**	**7**	**5**	**6**
			266m	147m	250m	269m	271m	364m	409m	272m	490m
			11h05	6h07	5h57	11h13	11h18	8h39	9h44	9h04	13h36
CDath5	272	9	**1**	(13)	(13)	**1**	**1**	(11)	(6)	(16)	(17)
			217m	114m	222m	217m	221m	330m	420m	235m	341m
			36h06	$+\infty$	$+\infty$	36h06	36h52	$+\infty$	$+\infty$	$+\infty$	$+\infty$
CDhavo4	299	8	**1**	(15)	(14)	**1**	**1**	(17)	**1**	(15)	(17)
			389m	213m	427m	388m	396m	634m	746m	429m	711m
			64h49	$+\infty$	$+\infty$	64h41	65h57	$+\infty$	124h20	$+\infty$	$+\infty$
JEALL	940	32	(20)	**1**	(18)	**2**	**2**	**3**	**2**	**2**	**3**
			356m	269m	552m	531m	659m	750m	982m	582m	1056m
			$+\infty$	44h46	$+\infty$	44h14	54h54	41h41	81h52	48h29	58h39
CDALL	1479	51	(170)	(177)	(173)	(149)	(149)	(185)	(113)	(179)	(169)
			474m	342m	709m	705m	959m	992m	1314m	690m	1162m
			$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Table 6.7. Performance of ten runs of *full* algorithm on the six tough instances of TSSGAP for smaller (left part) and larger (right part) upper bounds. We vary the restart parameters *sr* and *pr*, the maximal number of backjumps in total and between two subjects of the same level (*bj* and *bjft*), and the maximal number of backtracks *bt*. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number of unassigned pairs between parentheses. We also give the average CPU-time in minutes and the estimated time to success in hours.

so	RSO								MSF							
po	rnd				siz				rnd				siz			
ps	dyn		sta		dyn		sta		dyn		sta		dyn		sta	
sgo	cnf	rnd	cnf	rnd	cnf	rnd	cnf	rnd	cnf	rnd	cnf	rnd	cnf	rnd	cnf	rnd
JEvwo5	**7**	**3**	**4**	**5**	**4**	**7**	**1**	**1**	**4**	**4**	**2**	**3**	**1**	**1**	(4)	**1**
	29m	45m	40m	53m	39m	37m	60m	67m	33m	31m	49m	51m	42m	41m	51m	60m
	0h41	2h30	1h40	1h46	1h38	0h52	9h58	11h06	1h23	1h17	4h07	2h51	6h58	6h51	$+\infty$	9h57
JEhavo4	**4**	**1**	**4**	**3**	**5**	**3**	**4**	**2**	**5**	**3**	**4**	**5**	(7)	**4**	**1**	**4**
	269m	467m	313m	467m	290m	394m	269m	443m	254m	396m	288m	412m	329m	415m	274m	398m
	11h13	77h49	13h03	25h55	9h40	21h53	11h12	36h57	8h29	22h01	11h59	13h44	$+\infty$	17h18	45h36	16h34
CDath5	**1**	(13)	(9)	(13)	(12)	(15)	**1**	(13)	(8)	(13)	(3)	(6)	(13)	**1**	(6)	(4)
	217m	232m	234m	264m	221m	237m	212m	265m	204m	213m	227m	253m	206m	220m	206m	248m
	36h06	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	35h15	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	36h37	$+\infty$	$+\infty$
CDhavo4	**1**	(23)	(18)	(23)	(17)	(33)	(14)	(24)	(16)	(37)	(15)	(35)	(36)	(37)	(11)	(34)
	388m	482m	436m	552m	398m	487m	340m	530m	340m	431m	375m	509m	349m	435m	300m	506m
	64h41	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
JEALL	**2**	(42)	**1**	(58)	(37)	**1**	(47)	(22)	**1**	(74)	(1)	(1)	(101)	**2**	(101)	(13)
	531m	675m	458m	712m	427m	623m	402m	733m	477m	523m	398m	679m	331m	661m	379m	760m
	44h14	$+\infty$	76h24	$+\infty$	$+\infty$	103h45	$+\infty$	$+\infty$	79h30	$+\infty$	$+\infty$	$+\infty$	$+\infty$	55h04	$+\infty$	$+\infty$
CDALL	(149)	(170)	(169)	(187)	(119)	(176)	(311)	(250)	(160)	(209)	(104)	(156)	(161)	(162)	(157)	(468)
	705m	859m	725m	955m	735m	944m	655m	1016m	789m	1000m	835m	1102m	787m	999m	666m	1037m
	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Table 6.8. Performance of ten runs of *full* algorithm on the six tough instances for TSSGAP for different variable and value selection rules. We vary the subject ordering rule *so*, the pair ordering and selection rules *po* and *ps*, and the subject group ordering rule *sgo*. The second column (RSO,rnd,dyn,cnf) represents the choices in our suitable parameter setting. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number of unassigned pairs between parentheses. We also give the average CPU-time in minutes and the estimated time to success in hours.

		ni	***500***	1000	1000	1000	***2000***	1000	1000
		nni	100	***50***	100	100	100	***200***	100
		nit	10	10	***5***	10	10	10	***20***
instance	pairs	groups							
JEvwo5	127	4	**7**	**5**	**4**	**7**	**7**	**4**	**6**
			29m	25m	36m	29m	29m	63m	38m
			0h42	0h51	1h30	0h41	0h42	2h37	1h04
JEhavo4	270	7	**4**	**6**	**4**	**4**	**4**	**6**	**5**
			272m	160m	318m	269m	271m	434m	200m
			11h19	4h27	13h15	11h13	11h17	12h03	9h20
CDath5	272	9	**1**	(14)	(15)	**1**	**1**	(9)	(10)
			221m	143m	225m	217m	221m	389m	229m
			36h57	$+\infty$	$+\infty$	36h06	36h52	$+\infty$	$+\infty$
CDhavo4	299	8	**1**	**1**	(17)	**1**	**1**	(17)	**1**
			397m	248m	413m	388m	396m	719m	411m
			66h04	41h17	$+\infty$	64h41	65h57	$+\infty$	68h29
JEALL	940	32	**2**	**2**	**1**	**2**	**2**	**3**	**2**
			540m	331m	521m	531m	543m	794m	673m
			45h01	27h37	86h49	44h14	45h14	44h05	56h05
CDALL	1479	51	(149)	(164)	(172)	(149)	(149)	(131)	(165)
			714m	471m	676m	705m	712m	1235m	704m
			$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Table 6.9. Performance of ten runs of *full* algorithm on the six tough instances of TSSGAP for different tabu search parameters. We vary the maximal number of iterations in total and without improvement *ni* and *nni* and the number of iterations on the tabu list *nit*. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number of unassigned pairs between parentheses. We also give the average CPU-time in minutes and the estimated time to success in hours.

instance	pairs	groups	constr	back	bottle	full	instance	pairs	groups	constr	back	bottle	full
JEvwo4	22	2	**10**	**10**	**10**	**10**	*JEmavo4*	239	7	**10**	**10**	**10**	**10**
			<1m	<1m	<1m	<1m				1m	1m	2m	4m
			<0h01	<0h01	<0h01	<0h01				0h01	0h01	0h02	0h04
JEmavo3	38	4	**10**	**10**	**10**	**10**	*CDath6*	244	8	**5**	**10**	**10**	**10**
			<1m	<1m	<1m	<1m				5m	42m	3m	92m
			<0h01	<0h01	<0h01	<0h01				0h09	0h42	0h03	1h32
CDath4	67	7	**10**	**10**	**10**	**10**	*CDhavo5*	263	7	(10)	**6**	**4**	**10**
			<1m	<1m	<1m	<1m				7m	166m	5m	130m
			<0h01	<0h01	<0h01	<0h01				$+\infty$	4h36	0h13	2h10
JEvwo6	68	3	**10**	**10**	**10**	**10**	*JEhavo4*	270	7	(28)	**3**	(3)	**4**
			<1m	1m	<1m	4m				12m	388m	11m	269m
			<0h01	0h01	<0h01	0h04				$+\infty$	21h33m	$+\infty$	11h13
CDrmavo4	126	5	**8**	**8**	**10**	**10**	*CDath5*	272	9	(39)	(21)	(14)	**1**
			1m	14m	<1m	7m				8m	340m	8m	217m
			0h01	0h17	<0h01	0h07				$+\infty$	$+\infty$	$+\infty$	36h06
JEvwo5	127	4	(22)	**1**	**1**	**7**	*CDhavo4*	299	8	(51)	(47)	(19)	**1**
			2m	45m	2m	29m				14m	671m	11m	388m
			$+\infty$	7h33	0h23	0h41				$+\infty$	$+\infty$	$+\infty$	64h41
JEhavo5	176	5	**10**	**10**	**10**	**10**	*JEALL*	940	32	(210)	(149)	(177)	**2**
			2m	32m	1m	30m				24m	378m	18m	531m
			0h02	0h32	0h01	0h30				$+\infty$	$+\infty$	$+\infty$	44h14
CDdmavo4	208	7	**7**	**10**	**10**	**10**	*CDALL*	1479	51	(291)	(217)	(335)	(149)
			3m	9m	2m	44m				26m	809m	26m	705m
			0h04	0h09	0h02	0h44				$+\infty$	$+\infty$	$+\infty$	$+\infty$

Table 6.10. Performance of ten runs of the four variants *constr*, *back*, *bottle*, and *full* on all sixteen instances of TSSGAP for suitable parameter setting. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number of unassigned pairs between parentheses. We also give the average CPU-time in minutes and the estimated time to success in hours.

6.3 Approximating TSSGAP

Relax the curriculum constraint. In a preliminary study, we ran a variant of the constructive heuristic on the largest instances, *CDALL* and *JEALL*. In this variant, we did not restart but skipped all unassigned pairs at dead ends and continued with a pair we have not considered before. With this heuristic, we could find solutions with only 83 (*CDALL*) and 18 (*JEALL*) unassigned pairs. Thus, a very simple implementation of the solution approach did not solve the largest instances but stopped with partial solutions for which only a few percent of the decision variables remained unassigned. Although this may look very promising, it does not necessarily mean that we are very close to a complete solution. The remaining decisions may be extremely difficult, such that we have to revise a large part of the partial subject group assignment we had constructed.

The *full* tree search algorithm with the chosen parameter setting returned solutions with substantially more unassigned pairs for *CDALL*, see Table 6.6. All 149 pairs correspond to six optional subjects. An experiment with one hundred, instead of ten, runs for *CDALL* leads to a minimum of 89 unassigned pairs, for four different optional subjects. Due to randomness in the algorithm, the variation in the number of unassigned pairs is very large. One run finishes with 801 pairs of 27 different subjects unassigned, while 356 of these pairs are for mandatory subjects and therefore *must be* assigned in any acceptable solution.

Solve sub-instances. In the previous section, we have discovered that *CDhavo4*, *JEhavo4*, *CDath5*, and *JEvwo5* are tough instances of TSSGAP. The presence of these instances as sub-instances may be a possible explanation for not being able to solve the upper years instances *CDALL* and *JEALL* in all runs. Therefore, we run our *full* algorithm for sub-instances of *CDALL* and *JEALL* in which one or both levels are excluded. In Table 6.11, we show that we can easily find solutions of TSSGAP if both levels are removed. It is still impossible to solve the remaining instance of *CDALL* if only one of the levels is removed. But we are able to solve *CDALL-ath5-havo4*, an instance that has more pairs than *JEALL*, in all ten runs!

Remove constraints. We want to find out whether the performance of the *full* algorithm differs for variants of TSSGAP with some constraints removed. We carry out a number of experiments for relaxations of TSSGAP. We assume that the resource constraints must always be satisfied. Moreover, since we set the number of rooms $|R|$ at a large value, the room constraints are not very tight. Therefore, we do not consider relaxing this constraint type in the remainder.

In Table 6.12, we give the results for variants of TSSGAP without all educational constraints (*ed*), teacher availability constraints (*ta*), or subject group size constraints (*ss*). The variants *with* subject group size constraints have in general fewer successful runs, or are not solved at all. We also see that in some cases removing all constraints of a certain type does not imply a better performance. It seems that in these cases solving a more restricted problem is easier.

instance	pairs	groups	performance
CDALL	1479	51	(149)
CDALL-ath5	1277	42	(19)
CDALL-havo4	1180	43	(81)
CDALL-ath5-havo4	978	34	**10**
JEALL	940	32	**2**
JEALL-vwo5	813	28	**10**
JEALL-havo4	670	25	**7**
JEALL-vwo5-havo4	543	21	**10**

Table 6.11. Performance of ten runs of the *full* algorithm on sub-instances of *CDALL* and *JEALL* of TSSGAP for the suitable parameter setting. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number of unassigned pairs between parentheses.

ed	.	•	.	.	.	•	•	•
ta	.	.	•	.	•	.	•	•
ss	.	.	.	•	•	•	.	•
instance								
JEvwo5	**9**	**6**	**10**	**7**	**5**	**4**	**6**	**7**
JEhavo4	**10**	**10**	**10**	**7**	**7**	**7**	**10**	**4**
CDath5	**2**	**5**	**5**	(9)	(11)	(18)	**2**	**1**
CDhavo4	**10**	**10**	**10**	(12)	**1**	**3**	**9**	**1**
JEALL	**10**	**10**	**10**	**6**	**1**	**5**	**10**	**2**
CDALL	**2**	**4**	**1**	(172)	(164)	(165)	**2**	(149)

Table 6.12. Effectiveness of ten runs of the *full* algorithm on the six tough instances and several relaxations of TSSGAP for the suitable parameter setting. We remove all constraints of one or more of the following types: educational (*ed*), teacher availability (*ta*), or subject group size constraints (*ss*). The resource and room availability constraints are always hard. The hard constraints are marked "•", the constraint types marked "." have been excluded. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number unassigned pairs between parentheses.

	subject group size tolerance *sstol*							
instance	3	4	5	6	7	8	9	10
JEvwo5	**7**	**6**	**5**	**8**	**8**	**7**	**9**	**9**
JEhavo4	**4**	**5**	**8**	**6**	**6**	**7**	**8**	**6**
CDath5	**1**	(12)	(9)	(11)	(13)	(9)	**1**	(11)
CDhavo4	**1**	**1**	**1**	**2**	**2**	(15)	**3**	**3**
JEALL	**2**	**4**	**2**	**6**	**6**	**8**	**8**	**9**
CDALL	(149)	(220)	(183)	(132)	(168)	(61)	(53)	(18)

Table 6.13. Effectiveness of ten runs of *full* algorithm on the six tough instances of TSSGAP for the suitable parameter setting. We increase the subject group size tolerance *sstol*. We give the number of successful runs in boldface or, if all runs were unsuccessful, the minimum number unassigned pairs between parentheses.

Relax subject group size constraints. In the previous paragraph, we concluded that the subject group size constraints may be a cause for weak performance of the *full* algorithm. Therefore, we also carry out some experiments in which these constraints are relaxed. We do that by increasing the parameter for this constraint type, the subject group size tolerance *sstol*, step by step. Recall that the initial value of *sstol* is equal to three. The results of these experiments are given in Table 6.13. From this table, we can conclude that, in general, relaxing the subject group size constraints leads to more successful runs and, hence, a better performance of the algorithm. This strengthens our belief that the subject group size constraints have considerable impact on the performance of the tree search algorithm.

6.4 Summary

We applied the tree search algorithm on sixteen instances derived from two real-life school data sets. For each school, we had instances for each level in the upper years and one large instance comprising all levels. All instances except *CDALL*, the largest instance with 1479 (student group, subject) pairs to be assigned, were solved by means of the tree search algorithm with backjumping, backtracking, and bottleneck identification.

We found a suitable parameter setting for which all but the *CDALL* instance were solved. For this setting, we showed that we could not solve the remaining instance by changing each parameter individually. Also, the chosen parameter setting was considered to be more efficient than other settings which values were only different for one parameter.

Since there are random choices in our algorithm, we applied the algorithm ten times for different random seeds. The size of the instance, i.e., the number of pairs, gives a good indication for the success of the algorithm. The smallest instances, i.e., the instances with the least pairs, were solved ten times in only a few minutes of

computation time. For the largest instances, the algorithm performed much worse. We identified six tough instances: the two upper years instances *CDALL* and *JEALL*, the instances for the fourth year of HAVO, *CDhavo4* and *JEhavo4*, and the fifth year of VWO, *CDath5* and *JEvwo5*. Solving these instances took several days of CPU-time, since the majority of the ten runs were unsuccessful. The inherent difficulty of these levels contributes to the complexity of the upper years' instances *CDALL* and *JEALL*.

Applying both types of dead end handling, backjumping and backtracking on one side and bottleneck identification on the other side, pays off. The toughest instances were only solved by the *full* algorithm that uses them both.

The algorithm stopped with solutions for which only a few percent of the decision variables remain unassigned. We also observed that the algorithm performed better when the subject group size constraints were less tight or even removed.

Recall that the justification of our solution approach is based on two conjectures, which were defined in Section 5.5. We conjectured that it would be easy to find an initial feasible complete timetable and that repairing the current timetable with resource conflicts would be an efficient way to maintain a feasible complete timetable in each step. The first conjecture turns out to be true. For all sixteen instances, we could easily find a feasible timetable, even with only one run of the constructive heuristic defined on page 83. We have found less overwhelming evidence to conclude that the second conjecture is true. We are able to solve almost all instances, but for a number of instances the number of successful runs is a minority. To validate this conjecture, it will be necessary to implement a more sophisticated tabu or local search algorithm than the one we used and see if the performance improves.

7

Conclusion

In this chapter, we conclude our discussion of TSSGAP and the algorithm we implemented for solving it. Our algorithm has several procedures for which we have chosen straightforward implementations. In the previous chapter, we have seen that these implementations already give satisfactory results for most instances. Therefore, we have evidence that our solution approach is successful. We have not been able to compare the current implementations with more sophisticated ones.

In Section 7.1, we discuss how we could possibly improve the efficiency of the tree search algorithm for TSSGAP. Section 7.2 deals with the practical relevance of TSSGAP. That is, we show how the tree search algorithm for TSSGAP could be incorporated into a decision support system and what need to be done to tackle other characteristics of real-life timetable construction problems.

7.1 Improving the performance of the algorithms

Although the algorithm for TSSGAP has performed reasonably well on sixteen instances, we still have difficulties with some instances and are even unable to solve the largest instance. We still see ways to improve upon the current algorithm while sticking to the solution approach we defined.

The algorithm we defined in Chapter 5 has two major components. The subject group assignment decisions are carried out by a tree search algorithm, whereas a tabu search algorithm tries to find a feasible complete timetable in each step of the tree

search algorithm. For both components we have chosen two simple implementations. Below, we show how each component could be improved.

7.1.1 Subject group assignment

In our tree search algorithm, we first determine the next unassigned pair $p = (g, v) \in P$ we want to assign. Next, we try the possible subject groups $K_P(g, v) \subseteq K_V(v)$ for this pair in a specific order. If we find a feasible extension of the current subject group assignment k_P, then we continue with the next unassigned pair. Otherwise, we have detected a dead end, and at least one of the following possibilities holds for every possible subject group $k \in K_P(p)$.

1. The extended subject group assignment $k_P \cup \{(p,k)\}$ violates the subject group size constraint for subject group k.
2. We cannot find a feasible timetable for $k_P \cup \{(p,k)\}$ because no feasible timetable exists.
3. There exists a feasible timetable for $k_P \cup \{(p,k)\}$ but our current tabu search heuristic fails to find one.

Each time we detect a dead end, we remove the assignments of one or more pairs we considered before, and try alternatives for them.

In Subsection 7.1.2, we consider how we can improve our algorithm with regard to the latter two cases. In this subsection, we focus on the subject group size constraints. We distinguish two possible directions of improvement. First, we can perform consistency checking with regard to the subject group size constraints and try to detect inconsistency earlier. Second, we can determine the urgency of unassigned pairs based on the number of not inconsistent subject groups for each pair. In that case, we may select a more constrained pair than we do with the current strategy.

Apply consistency checking. For each unassigned pair, we can keep track of the set of subject groups to which the pair can be assigned without violating the subject group size constraints. These sets depend on the current subject group assignment. Let k_P be the current subject group assignment, and $p = (g, v)$ be an unassigned pair for student group g and subject v. A subject group $k \in K_V(v)$ is inconsistent for (g, v) if $z_G(g)$, the size of student group g, is larger than $\overline{S_V}(v) - \sum_{g' \in G_K(k)} z_G(g')$, the remaining capacity of subject group k. In that case, we can remove k from the set of possible subject groups $K_P(p)$. Initially, when the subject group assignment is empty, none of the subject groups $K_V(v)$ is inconsistent.

Each time we assign a pair (g, v) for student group g to a subject group k, we must check whether k has become inconsistent for some unassigned pairs for the same subject. Dead ends can be detected in two ways. If for one pair all subject groups are inconsistent and, consequently, the set of possible subject groups is empty, then we cannot extend the current subject group assignment to a complete one. We can also propagate the subject group size constraints by trying to solve the remaining

bin packing problems for unassigned pairs $P^{\text{not}} \subseteq P$. That is, for every subject v for which not all pairs have been assigned, i.e., $v \in \{ v_{\text{P}}(p) \mid p \in P^{\text{not}} \}$, we can try to find a subject group assignment for the set of all remaining pairs P^{not}. In general, these bin packing problems have bins with unequal capacities. If we can prove that for some subject no feasible subject group assignment exists, then no feasible complete subject group assignment exists that extends k_{P}. In that case, we have also detected a dead end.

In our tree search algorithm, all constraints are checked *after* carrying out the subject group assignment. In this way, we can only prove inconsistency for the pair that we assigned in the last step. By applying consistency checking, we may be able to prove inconsistency for other unassigned pairs as well. With the additional computational effort, we can detect dead ends earlier and therefore reduce the number of unnecessary backtracks of the tree search algorithm. Consequently, we might try higher values of the backjump and backtrack parameters and hopefully obtain better results.

Use a different dynamic pair selection rule. The tree search algorithm uses heuristic rules to select the unassigned pair that will be assigned in the next step. By means of the pair ordering rule and pair selection rule parameters, we can experiment with different selection strategies. For example, we can choose for the dynamic pair selection rule. This rule gives a higher priority to pairs that introduce more resource constraints, see page 86. Thus, the underlying selection rule only focuses on the resource constraints and does not pay attention to the subject group size constraints at all.

An alternative dynamic selection rule determines the urgency on both the resource and the subject group size constraints. For example, we can choose as the next pair that we want to assign the one with

1. the least possible subject groups, and among those
2. the one that introduces the most new resource constraints.

In this case, we try to assign pairs with only one remaining subject group before other pairs. Again, the main benefit is that we can detect dead ends earlier in the search.

7.1.2 Time slot assignment

After each extension of the subject group assignment k_{P} by the assignment of pair $p = (g, v)$ to subject group k, the algorithm determines whether $k'_{\text{P}} = k_{\text{P}} \cup \{(p, k)\}$ satisfies all constraints. First, it checks whether the subject group size constraint for k is satisfied. If this is the case, it verifies feasibility of the current timetable t_{L} for the extended subject group assignment k'_{P}. The tree search algorithm continues with the next unassigned pair if $\mathsf{Feasible}(k'_{\text{P}}, t_{\text{L}})$ holds. Otherwise, some lessons of $L_{\text{K}}(k)$ conflict with lessons in $L_{\text{G}}(g)$ in t_{L}. Then, we try to detect infeasibility by determining a large clique heuristically and check the maximum clique condition

defined in Subsection 4.3.2. Whenever we cannot prove infeasibility, we try to repair t_L by means of a tabu search algorithm.

In the next paragraphs, we discuss the current implementation of these steps in the tree search algorithm and alternatives that may improve the performance. First, we look at the clique heuristic. Second, we present ways to enhance our tabu search algorithm. We conclude with a completely different way of determining a feasible timetable, i.e., by constructing one from scratch without using the timetable found so far.

Detect infeasibility of the maximum clique condition. Currently, we use a very simple heuristic to determine a large clique in the graph. Finding a maximum clique is NP-hard and using an optimization algorithm is impractical for our instances. We implemented a heuristic that gradually increases a clique by adding a vertex with maximum degree that is adjacent to all vertices currently in the clique. This greedy constructive heuristic is usually outperformed by more intelligent ones. The price that must then be paid is the increase in computation time.

Carter & Gendreau [1992] have developed a recursive algorithm that follows a different strategy. Their algorithm iteratively deletes vertices with minimum degree because these are the least likely to be member of a maximum clique. The algorithm stops when the remaining sub-graph is a clique. Several versions of the algorithm have been tested, including a heuristic without recursion and an optimization algorithm. Versions with a few levels of recursion already return optima within a few seconds of CPU-time, but without a guarantee of optimality.

Recall that this feasibility check is carried out at the deepest level of our code: after each extension of the subject group assignment. Therefore, even CPU-times of more than one second are impractical. However, we may choose to spend more time on computing a larger clique regularly but less often.

Enhance the tabu search algorithm. If the current timetable t_L is infeasible for the extended subject group assignment $k'_P := k_P \cup \{(p,k)\}$, then we try to find a different complete timetable t'_L that is feasible for k'_P. Note that t_L has at most $|L_K(k)|$ resource conflicts, see page 75. We relax the resource constraints and try to find t'_L by using a tabu search algorithm that repairs t_L. In each step of the tabu search algorithm, we move to a timetable with minimum resource conflicts that is slightly different from the current one and satisfies all other constraints. Because this procedure is, in the worst case, carried out after each extension of the subject group assignment, the tabu search algorithm is called a large number of times; for the fourteen single-level instances we considered, there are already thousands of calls. Therefore, the tabu search algorithm must preferrably stop within a very short amount of time.

Finding an effective and time-efficient implementation of the tabu search template is a major research topic in itself. Many effective implementations are tailor-made and follow from extensive experiments on the parameters of the template. We have started with a very simple implementation of the tabu search template and have

not been able to investigate different alternatives.

We have chosen for a single tabu list and a fixed tabu tenure. In practice, a dynamic tabu tenure usually has a better performance than a fixed one, see e.g., [Alvarez-Valdes, Crespo & Tamarit, 2002]. This alternative can be implemented quickly, so it would be reasonable to try this possibility first.

Our neighborhood structure is rather straightforward: at most two lessons may be reassigned and all educational, room availability, and teacher availability constraints must be satisfied. Many neighboring timetables have the same number of resource conflicts, so it is very likely that there is more than one timetable with maximum gain. Also, we must satisfy all constraints except the resource constraints, so we cannot move to every possible timetable. Since some of the neighboring timetables are marked tabu, the set of possible timetables is even further reduced. In the worst possible situation, there are no neighbors at all. An interesting question would be to find a reasonably sized neighborhood structure that overcomes these problems. A commonly used approach is to relax other constraints as well and weigh violations of relaxed constraints in the cost function. One could also choose to implement more than one neighborhood structure, e.g., by trying to reassign all lessons of a single subject group, student group, or teacher.

Construct a new timetable from scratch. If there exists a feasible timetable t_{L}' that differs only slightly from t_{L}, we expect to find t_{L}' by means of our tabu search algorithm. As the runs of the tabu search algorithm must be short, we cannot choose high values for the maximal number of iterations and therefore do not evaluate a huge number of timetables. It is reasonable to ask ourselves what we need to do if there does exist a feasible timetable that we cannot find because we spend our time in the wrong part of the solution space of possible timetables.

Of course, we could apply diversification strategies in our tabu search algorithm that guide us to different, yet unexplored parts of the solution space. We could also try to construct a feasible complete timetable from scratch when tabu search was unable to find one. We expect that constructing a feasible complete timetable always from scratch is too time-consuming. Coudert [1997] states that constructing optimal colorings for real-life graphs does not take superpolynomial time in practice. He implements a branch-and-bound algorithm to solve the graph coloring problem. His algorithm first determines a large clique by performing incomplete enumeration, then colors the vertices in this clique, and finally enumerates colorings, in worst case all, of the remaining vertices for the given coloring of the clique. Note that there always exists an optimal coloring with the given proper coloring of the clique. His algorithm is successful for graphs following from real-life applications, which are all 1-perfect, see page 61, because his clique heuristic always finds a maximum clique for this type of graph and, moreover, coloring the remaining vertices optimally turns out to be easy.

We wondered whether Coudert's positive results on graph coloring problems for

timetabling graphs would be applicable to instances of TSSGAP as well. De Louw [2002] reports on an algorithm based on Coudert's graph coloring approach that takes all constraints of TSSGAP into account. He ran his algorithm on a number of instances that occurred as instances for the tabu search algorithm in our tree search algorithm. Based on these experiments we conclude that there are instances that can be solved by our tabu search algorithm but cannot be solved by the Coudert-like heuristic. A possible explanation for this weak performance on instances of TSSGAP is that we may have to revise the assignments in the clique we constructed in the first step. Recall that this is unnecessary for plain graph coloring. Due to the additional constraints of TSSGAP, Coudert's approach excludes a large part of the solution space, which may contain feasible solutions.

7.2 Toward a practical timetable

TSSGAP is a sub-problem of timetable construction problems at most Dutch secondary schools. In fact, most planners consider TSSGAP a core problem that must be solved first, before considering room assignment and polishing the timetable given the desires of students and teachers. Therefore, one can use the tree search algorithm we developed as a sub-routine in an integrated approach for solving school timetable construction problems.

Decision support systems (*DSSs*) are frequently used for solving real-life planning and scheduling problems in general and timetable construction problems in particular, see e.g., [Alvarez-Valdes, Crespo & Tamarit, 2002] [Bardadym, 1996] [Ferland & Fleurent, 1992]. Both best-selling commercial software packages in the Netherlands, the packages Curs & gp-Untis of Deloitte & Touche ICS Adviseurs and Clusterfact & Roosterfact, which was sold by CMG, are examples of DSSs. The main components of a DSS are a graphical user interface, several views on the input data and solution, and algorithms for solving sub-problems. The user interface should enable a planner to add, change, or remove assignments and switch between the possible views easily. There should be views on the timetable for each resource type and on the composition of the subject groups. Moreover, a planner should have an overview of all imperfections of the current solution and suggestions on how to overcome them. The clustering algorithms of Curs and Clusterfact and our tree search algorithm are examples of algorithms that can be hidden behind a menu option of a DSS.

We did not compare the performance of our tree search algorithm for TSSGAP with the algorithms in the currently best-selling DSSs for timetable construction problems at Dutch secondary schools. If algorithms in these DSSs would also tackle TSSGAP directly, then we could compare these algorithms with our tree search algorithm. Unfortunately, this is not the case. However, we can find solutions for TSSGAP by means of the algorithms in the DSSs, but then user interaction is required.

In Section 4.4, we have shown that planners must split full instances into instances per level before determining a clustering. Therefore, we have to define sub-instances and solve the clustering problem with either Curs or Clusterfact first. Then, we have to choose an acceptable clustering. Finally, we can assign clusters to time slots by means of the algorithms in gp-Untis and Roosterfact. The need for user interaction makes a comparison of our tree search algorithm and the solution approaches in commercial software hard. In that case, we should evaluate the approaches instead of the algorithms and allow user interaction on our tree search algorithm too.

In the remainder of this section, we discuss how TSSGAP can be used in a DSS for solving real-life timetable construction problems. We split our discussion into two parts. First, we discuss how several characteristics, currently not modeled in TSSGAP, can be taken into account. Second, we show how user interaction in a DSS can help in solving instances of TSSGAP if our tree search algorithm does not return a feasible complete solution.

7.2.1 Dealing with other characteristics

Several characteristics of timetable construction problems at Dutch secondary schools are not captured in the TSSGAP model. First, we excluded room assignment, room types, and locations. As a consequence, we cannot guarantee that a feasible room assignment exists for the timetable we construct. Second, we do not take idle time into account. Therefore, teachers and students may have unacceptable idle time in our final timetable. Third, we cannot account for blocks. Below, we discuss which characteristics can easily be added to our TSSGAP model and approach, and which cannot. At the end of this subsection, we elaborate on modeling self-study during school hours, a new characteristic that must be dealt with at Dutch secondary schools since 1999.

Room types and locations. We can easily add room types to our model of TSSGAP. In that case, we would replace the room availability constraints by the room assignment constraint. That is, the time slot assignment t_{L} is feasible only if there exists a room assignment $r_{\mathrm{L}} : L \to R$ that satisfies the room constraints and room compatibility constraints, i.e., $l_1 \neq l_2 \;\Rightarrow\; r_{\mathrm{L}}(l_1) \neq r_{\mathrm{L}}(l_2)$ holds for every time slot t and any pair of lessons $l_1, l_2 \in L_{\mathrm{T}}(t)$ assigned to time slot t, and $y_{\mathrm{R}}(r_{\mathrm{L}}(l)) \in Y_{\mathrm{L}}(l)$ holds for every lesson $l \in L$. For a given timetable t_{L}, this constraint can be checked in polynomial time if there are no blocks. Recall that this boils down to determining a maximum matching in a bipartite graphs for every time slot. The problem becomes NP-hard in case of blocks, see Section 3.7.

Idle time. It is less straightforward to define which timetables have acceptable amounts of idle time for teachers and students and which do not. We can count the gaps and their sizes in a teacher's or student's timetable and put limits on the number of gaps of a given size. For teachers, we can easily add these constraints to our TSSGAP model, such that timetables that do not satisfy these constraints are

excluded. We cannot do the same for students, because we do not know all lessons of each student yet. Therefore, we would prefer to consider idle time for students in a post-processing step. That is, when we have constructed a feasible complete solution of TSSGAP, we can determine the idle time per student and teacher. If this is unacceptable, we could try to find an acceptable timetable by means of a local search algorithm.

Blocks and breaks. If we want to model blocks, then we must also take breaks into account, because planners do not want that blocks overlap breaks. In the TC model of Chapter 2, we model breaks by introducing block times and requiring that blocks are only assigned to these block times. We can extend TSSGAP by adding the same kind of constraints.

Usually, only two lessons of the same subject group can make a block. Therefore, the educational constraints should not conflict with the block time constraints. Moreover, we must change the tabu search algorithm slightly. If we reassign one lesson of a block, then we must also reassign its mate to a consecutive time slot.

If there are many blocks in practice, or if we want to model blocks of different sizes, then we may prefer a model with lessons of variable but prespecified length instead of unit time length.

Self-study during school hours. In 1999, the Dutch government introduced the concept of *studiehuis* into the upper years of secondary schools. The idea behind this *studiehuis*, or *places of study*, concept is that students must master subjects not only by attending the traditional class-teacher lessons with students of the same subject group, but also by individual work on a subject through self-study.

Consequently, several traditional lessons are replaced by self-study time during school hours. During these hours, some teachers are available for helping students with questions on subjects.

Now, planners must determine at which time each student will have self-study time. Moreover, they must assign teachers with prespecified capabilities to time slots. We must take into account that every student must be able to consult a teacher for every subject with self-study time at least once.

Some schools have decided that certain time slots are reserved for self-study of all students in upper years. This can easily be modeled in TSSGAP. In that case, we must mark the corresponding time slots as being unavailable. Moreover, we must reduce the number of lessons for each subject group by the amount of self-study time for that subject. Then, the planner must only determine which teachers will be available for consults at what times. It is clear that we can only implement the *studiehuis* in that way if we are able to house all upper year students at a school location outside of the classrooms.

We have also seen a different implementation of self-study time at the Pleincollege Sint-Joris. Here, self-study time and teachers are allocated after the timetable is constructed. The planner tries to assign teachers to time slots such that students' idle

time can be used for self-study time. That is, he first constructs a timetable without taking the self-study time into account. Next, he removes all lessons that must be replaced by self-study time. Finally, he determines which teachers must be available for consults at each time slot, based on the curricula of students with idle time. We have not been able to represent this situation in our TSSGAP model.

7.2.2 Solve TSSGAP by user interaction

In case of success, we can obtain a feasible complete solution of any instance of TSSGAP by running our tree search algorithm once on the instance under consideration. However, TSSGAP is NP-hard, so it is unlikely that there exists an algorithm that constructs such a solution in reasonable time for all instances. In practice, planners want to find any solution for their timetable construction problem and as quickly as possible. As a consequence, a tool *must* return a solution, which is hopefully feasible and complete.

Our experiments have shown that we are still unable to solve some instances. However, we believe that the current tree search algorithm is still useful for the instances that could not be solved. We can try to construct a solution of TSSGAP by allowing user interaction. In that case, we run our tree search algorithm more than once and on different instances. For example, we can try to solve *CDALL*, the full instance for Cambreurcollege in Dongen, by first constructing a solution for the *havo4* level, then for *ath5* level, and finally for the remaining levels. In Section 6.3, we have seen that we are able to solve these three instances individually.

To allow user interaction, we must be able to start the tree search algorithm with a feasible partial solution. Our current algorithm can deal with any feasible solution in which some pairs are assigned and others are not. Moreover, our tree search algorithm will never undo the subject group assignments in the initial solution, so we can consider these assignments as being fixed in the consecutive steps. The timetable in the initial solution is, of course, not fixed and will update itself in the next steps, if necessary.

Finally, we recall that a planner may relax one or more constraints for the instance under consideration. In that case, he would not solve the original problem but a related one that is less constrained. For this reason, a DSS should assist a planner by determining infeasibility of the remaining instance as soon as possible, e.g., by checking the conditions defined in Section 4.3.

Bibliography

AARTS, E.H.L., AND J.K. LENSTRA (eds.) [1997], *Local search in combinatorial optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Ltd.

ALVAREZ-VALDES, R., E. CRESPO, AND J.M. TAMARIT [2002], Design and implementation of a course scheduling system using tabu search, *European Journal of Operational Research* **137**, 512–523.

ALVAREZ-VALDES, R., G. MARTIN, AND J.M. TAMARIT [1996], Constructing good solutions for the Spanish school timetabling problem, *Journal of the Operational Research Society* **47**, 1203–1215.

ARKIN, E.M., AND E.B. SILVERBERG [1987], Scheduling jobs with fixed start and end times, *Discrete Applied Mathematics* **18**, 1–8.

ASRATIAN, A.S., T.M.J. DENLEY, AND R. HÄGGKVIST [1998], *Bipartite Graphs and their Applications*, Cambridge Tracts in Mathematics 131, Cambridge University Press.

ASRATIAN, A.S., AND D. DE WERRA [2002], A university timetable problem: Complexity and existence problems, to appear in *European Journal of Operational Research*.

AUBIN, J., AND J.A. FERLAND [1989], A large scale timetabling problem, *Computers and Operations Research* **16**, 67–77.

BANKS, D., P. VAN BEEK, AND A. MEISELS [1998], A heuristic incremental modeling approach to course timetabling, *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence*, 16–29.

BARDADYM, V.A. [1996], Computer-aided school and university timetabling: The new wave?, in: E.K. Burke and P. Ross (eds.), *Practice and Theory of Automated Timetabling, First International Conference, Selected papers*, Lecture Notes in Computer Science 1153, Springer, 22–45.

BERGHUIS, J., A.J. VAN DER HEIDEN, AND R. BAKKER [1964], The preparation of school timetables by electronic computer, *BIT* **4**, 106–114.

BONVANIE, M.C., AND J.A.M. SCHREUDER [1982], *Model lesroosters vakkenpakket VWO*, Technical Report 411, Universiteit Twente, Enschede, The Netherlands.

BRÉLAZ, D. [1979], New methods to color the vertices of a graph, *Communications of the ACM* **22**, 251–256.

BRUNETTA, L., AND A. DE POLI [1997], Integer programming for solving the

course timetabling problem, *Ricerca Operativa* **27**, 35–55.

BURKE, E.K., AND M.W. CARTER (eds.) [1997], *Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling*, University of Toronto, Canada.

BURKE, E.K., AND M.W. CARTER (eds.) [1998], *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT '97, Selected papers*, Lecture Notes in Computer Science 1408, Springer.

BURKE, E.K., AND W. ERBEN (eds.) [2000], *Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling*, Fachhochschule Konstanz, Germany.

BURKE, E.K., AND W. ERBEN (eds.) [2001], *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers*, Lecture Notes in Computer Science 2080, Springer.

BURKE, E.K., AND P. ROSS (eds.) [1996], *Practice and Theory of Automated Timetabling, First International Conference, Selected papers*, Lecture Notes in Computer Science 1153, Springer.

CARTER, M.W. [2001], A comprehensive course timetabling and student scheduling system at the University of Waterloo, in: E.K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers*, Lecture Notes in Computer Science 2079, Springer, 64–82.

CARTER, M.W., AND M. GENDREAU [1992], *A Practical Algorithm for Finding the Largest Clique in a Graph*, Technical Report 820, Centre de recherche sur les transports, Montreal, Canada.

CARTER, M.W., AND G. LAPORTE [1996], Recent developments in practical examination timetabling, in: E.K. Burke and P. Ross (eds.), *Practice and Theory of Automated Timetabling, First International Conference, Selected papers*, Lecture Notes in Computer Science 1153, Springer, 3–21.

CARTER, M.W., AND G. LAPORTE [1998], Recent developments in practical course timetabling, in: E.K. Burke and M.W. Carter (eds.), *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT '97, Selected papers*, Lecture Notes in Computer Science 1408, Springer, 3–19.

CARTER, M.W., AND C.A. TOVEY [1992], When is the classroom assignment problem hard?, *Operations Research* **40**, S28–S39.

COFFMAN, JR., E.G., M.R. GAREY, AND D.S. JOHNSON [1978], An application of bin-packing to multiprocessor scheduling, *SIAM Journal on Computing* **7**, 1–17.

COFFMAN, JR., E.G., M.R. GAREY, AND D.S. JOHNSON [1997], Approximation algorithms for bin packing: A survey, in: D.S. Hochbaum (ed.), *Approximation algorithms for NP-hard problems*, PWS Publishing Company, 46–97.

COLORNI, A., M. DORIGO, AND V. MANIEZZO [1998], Metaheuristics for high

school timetabling, *Computational Optimization and Applications* **9**, 275–298.

COOK, S.A. [1971], The complexity of theorem-proving procedures, *Proceedings of the 3rd. Annual ACM symposium on Theory of computing*, 151–158.

COOK, W.J., W.H. CUNNINGHAM, W.R. PULLEYBLANK, AND A. SCHRIJVER (eds.) [1998], *Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Ltd.

COOPER, T.B., AND J.H. KINGSTON [1996], The complexity of timetable construction problems, in: E.K. Burke and P. Ross (eds.), *Practice and Theory of Automated Timetabling, First International Conference, Selected papers*, Lecture Notes in Computer Science 1153, Springer.

COSTA, D. [1994], A tabu search algorithm for computing an operational timetable, *European Journal of Operational Research* **76**, 98–110.

COSTA, D., A. HERTZ, AND O. DUBUIS [1995], Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs, *Journal of Heuristics* **1**, 105–128.

COUDERT, O. [1997], Exact coloring of real-life graphs is easy, *Proceedings of the 34th Design Automation Conference ACM-IEEE*, 121–126.

DI GASPERO, L., AND A. SCHAERF [2001], Tabu search techniques for examination timetabling, in: E.K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers*, Lecture Notes in Computer Science 2079, Springer, 104–117.

EIKELDER, H.M.M. TEN, AND R.J. WILLEMEN [2001], Some complexity aspects of secondary school timetabling problems, in: E.K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers*, Lecture Notes in Computer Science 2079, Springer, 18–27.

EVEN, S., A. ITAI, AND A. SHAMIR [1976], On the complexity of timetable and multicommodity flow problems, *SIAM Journal on Computing* **5**, 691–703.

FELDMAN, R., AND M.C. GOLUMBIC [1990], Optimization algorithms for student scheduling via constraint satisfiability, *The Computer Journal* **33**, 356–364.

FERLAND, J.A. [1998], Generalised assignment-type problems: A powerful modeling scheme, in: E.K. Burke and M.W. Carter (eds.), *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT '97, Selected papers*, Lecture Notes in Computer Science 1408, Springer, 53–77.

FERLAND, J.A., AND C. FLEURENT [1992], *SAPHIR: A Decision Support System for Course Scheduling*, Technical Report 764, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada.

GALINIER, P., AND J-K. HAO [1999], Hybrid evolutionary algorithms for graph coloring, *Journal of Combinatorial Optimization* **3**, 379–397.

GANS, O.B. DE [1981], A computer timetabling system for secondary schools in the Netherlands, *European Journal of Operational Research* **7**, 175–182.

GAREY, M.R., AND D.S. JOHNSON [1979], *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company.

GIARO, K., M. KUBALE, AND D. SZYFELBEIN [2000], Consecutive graph coloring for school timetabling, in: E.K. Burke and W. Erben (eds.), *Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling*, Fachhochschule Konstanz, Germany, 212–221.

GOLUMBIC, M.C. (ed.) [1980], *Algorithmic Graph Theory and Perfect Graphs*, Academic Press.

GOTLIEB, C.C. [1963], The construction of class-teacher time-tables, *Information Processing, Proceedings of IFIP Congress 62*, 73–77.

GRAHAM, R.L. [1969], Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics* **17**, 263–269.

GRAVES, R.L., L. SCHRAGE, AND J. SANKARAN [1993], An auction method for course registration, *Interfaces* **23**, 81–92.

HERTZ, A. [1991], Tabu search for large scale timetabling problems, *European Journal of Operational Research* **54**, 39–47.

HERTZ, A. [1992], Finding a feasible course schedule using tabu search, *Discrete Applied Mathematics* **35**, 255–270.

HERTZ, A., AND V. ROBERT [1998], Constructing a course schedule by solving a series of assignment type problems, *European Journal of Operational Research* **108**, 585–603.

HERTZ, A., AND D. DE WERRA [1987], Using tabu search for graph coloring, *Computing* **39**, 345–351.

JOHNSON, D.S., C.R. ARAGON, L.A. MCGEOCH, AND C. SCHEVON [1991], Optimization by simulated annealing: An experimental evaluation; Part II, graph coloring and number partitioning, *Operations Research* **39**, 378–406.

JOHNSON, D.S., AND M.A. TRICK (eds.) [1996], *Cliques, Coloring, and Satisfiability. Second DIMACS Implementation Challenge, October 11-13, 1993*, DIMACS. Series in Discrete Mathematics and Theoretical Computer Science 26, American Mathematical Society.

KARP, R.M. [1972], Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85–103.

KESTEREN, B. VAN [1999], The clustering problem in Dutch high schools: Changing metrics in search space, Internal report, Leiden Institute for Advanced Computer Science, Leiden University, Leiden, The Netherlands, 99–06.

KINGSTON, J.H. [2001], Modelling timetabling problems with STTL, in: E.K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers*, Lecture Notes in Computer Science 2079, Springer, 309–321.

KUBALE, M., AND B. JACKOWSKI [1985], A generalized implicit enumeration al-

gorithm for graph coloring, *Communications of the ACM* **28**, 412–418.

LAGUNA, M., AND R. MARTÍ [2001], A GRASP for coloring sparse graphs, *Computational Optimization and Applications* **19**, 165–178.

LAPORTE, G., AND S. DESROCHES [1986], The problem of assigning students to course sections in a large engineering school, *Computers and Operations Research* **13**, 387–394.

LEIGHTON, F.T. [1979], A graph coloring algorithm for large scheduling problems, *Journal of Research of the National Bureau of Standards* **84**, 489–503.

LEWANDOWSKI, G. [1994], *Practical implementations and applications of graph coloring*, Ph.D. thesis, University of Wisconsin-Madison, U.S.A.

LOUW, P. DE [2002], Graph coloring in timetabling, Master's thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, Eindhoven, The Netherlands, to appear.

MARTELLO, S., AND P. TOTH [1990], *Knapsack problems: Algorithms and Computer Implementations*, John Wiley & Sons Ltd.

MIYAJI, I., K. OHNO, AND H. MINE [1987], Solution method for partitioning students into groups, *European Journal of Operational Research* **33**, 82–90.

MORGENSTERN, C.A. [1990], *Algorithms for general graph coloring*, Ph.D. thesis, University of New Mexico, Albuquerque, New Mexico, U.S.A.

NUIJTEN, W.P.M. [1994], *Time and resource constrained scheduling: A constraint satisfaction approach*, Ph.D. thesis, Technische Universiteit Eindhoven, The Netherlands.

REIS, L.P., AND E. OLIVEIRA [2001], A language for specifying complete timetabling problems, in: E.K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers*, Lecture Notes in Computer Science 2079, Springer, 322–341.

ROBERT, V., AND A. HERTZ [1996], How to decompose constrained course scheduling problems into easier assignment type subproblems, in: E.K. Burke and P. Ross (eds.), *Practice and Theory of Automated Timetabling, First International Conference, Selected papers*, Lecture Notes in Computer Science 1153, Springer, 364–373.

RUDOVÁ, H., AND L. MATYSKA [2000], Constraint-based timetabling with student schedules, in: E.K. Burke and W. Erben (eds.), *Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling*, Fachhochschule Konstanz, Germany, 109–123.

SAMPSON, S.E., J.R. FREELAND, AND E.N. WEISS [1995], Class scheduling to maximize participant satisfaction, *Interfaces* **25**, 30–41.

SCHAERF, A. [1995], *A survey of automated timetabling*, Technical Report CS-R9567, CWI, Amsterdam, The Netherlands.

SCHAERF, A. [1996], *Tabu search techniques for large high-school timetabling prob-*

lems, Technical Report CS-R9611, CWI, Amsterdam, The Netherlands.

SCHOLL, A., R. KLEIN, AND C. JÜRGENS [1997], BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem, *Computers and Operations Research* **24**, 627–645.

SEWELL, E.C. [1996], An improved algorithm for exact graph coloring, in: D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability. Second DIMACS Implementation Challenge, October 11-13, 1993*, DIMACS. Series in Discrete Mathematics and Theoretical Computer Science 26, American Mathematical Society, 359–373.

SIMONS, J.L. [1974], *ABC: Een programma dat automatisch blokken construeert bij de vakdifferentiatie binnen het algemeen voortgezet onderwijs*, Technical Report NLR TR 74107 U, NLR, Amsterdam, The Netherlands.

TRIPATHY, A. [1992], Computerised decision aid for timetabling: A case analysis, *Discrete Applied Mathematics* **35**, 313–323.

WELSH, D.J.A., AND M.B. POWELL [1967], An upper bound for the chromatic number of a graph and its application to timetabling problems, *The Computer Journal* **10**, 85–86.

WERRA, D. DE [1971], Construction of school timetables by flow methods, *INFOR* **9**, 12–22.

WERRA, D. DE [1985], An introduction to timetabling, *European Journal of Operational Research* **19**, 151–162.

WERRA, D. DE [1997], The combinatorics of timetabling, *European Journal of Operational Research* **96**, 504–513.

WERRA, D. DE [1999], On a multiconstrained model for chromatic scheduling, *Discrete Applied Mathematics* **94**, 171–180.

WERRA, D. DE, A.S. ASRATIAN, AND S. DURAND [2002], Complexity of some special types of timetabling problems, to appear in *Journal of Scheduling*.

WIJNHOVEN, L. [2000], Het voortgezet onderwijs: Gids voor ouders, verzorgers en leerlingen 2000-2001, Ministerie van Onderwijs, Cultuur en Wetenschappen, Zoetermeer, The Netherlands.

WILLEMEN, R.J., AND H.M.M. TEN EIKELDER [2002], A core problem in educational timetabling and its solution by tree search, manuscript, submitted to *European Journal of Operational Research*.

WOOD, J., AND D. WHITAKER [1998], Student centred school timetabling, *Journal of the Operational Research Society* **49**, 1146–1152.

Symbol Index

The numbers in the right column refer to the pages of first occurrence.

Basic sets

$\mathbb{B}$	set of booleans $\{\text{true}, \text{false}\}$	
$\mathbb{N}$	set of nonnegative integers $\{0, 1, 2, \dots\}$	
$\mathbb{Z}^+$	set of positive integers $\{1, 2, 3, \dots\}$	

Definitions

$a := b$	a is by definition equal to b	
$a :\Leftrightarrow b$	a is by definition equivalent to b	

Operations on sets

$A \cup B$	union of sets A and B	
$A \cap B$	intersection of sets A and B	
$A - B$	elements of set A that are not in B	
$A \triangle B$	symmetric difference between sets A and B	
$\mathcal{P}(X)$	power set of set X	

Blocks

$\mathsf{Block}(l_1, l_2)$	lessons l_1 and l_2 form a block	17
$\mathsf{BlockTime}(t_1, t_2)$	time slots t_1 and t_2 consecutive and not separated by a break	15

Constraint types

bt	all block time constraints	21
cu	the curriculum constraint	54
ed	all educational constraints	18
rb	all room assignment for blocks constraints	21
rc	all room compatibility constraints	20
ro	all room constraints	19
ss	all subject group size constraints	20
st	all student group constraints	19

Constraint symbols

Graph coloring representation

Input sets

Lesson attributes

Lesson sets

$L_{\mathrm{T}}(t)$	set of lessons taught at time slot t	54

Pair attributes

$g_{\mathrm{P}}(g,v)$	student group of pair (g,v)	17
$k_{\mathrm{P}}(g,v)$	subject group to which pair (g,v) has been assigned	18
$v_{\mathrm{P}}(g,v)$	subject of pair (g,v)	17

Room attributes

$y_{\mathrm{R}}(r)$	room type of room r	15

Sets

A	set of levels	16
C	set of clusters of lessons	66
D	set of teachers	15
G	set of student groups	16
K	set of subject groups	16
L	set of lessons	17
P	set of (student group, subject) pairs	17
Q	set of clusters of subject groups	63
R	set of rooms	15
T	set of time slots	15
V	set of subjects	16
W	set of days	15
Y	set of room types	15

Sizes

$z_{\mathrm{G}}(g)$	number of students in student group g	16
$z_{\mathrm{K}}(k)$	number of students assigned to subject group k	64

Student groups

$G_{\mathrm{K}}(k)$	set of student groups assigned to subject group k	64
$G_{\mathrm{V}}(v)$	set of student groups that have chosen subject v	64

Student group attributes

$a_{\mathrm{G}}(g)$	level of all students in student group g	16
$V_{\mathrm{G}}(g)$	set of subjects in curriculum of student group g	16

Subject attributes

$a_{\mathrm{V}}(v)$	level of subject v	16
$K_{\mathrm{V}}(v)$	set of subject groups for subject v	18

Subject group attributes

$v_{\mathrm{K}}(k)$	subject of subject group k	16

Teacher attributes

$T_{\mathrm{D}}(d)$	set of time slots at which teacher d is available	15

Time slot attributes

$w_{\mathrm{T}}(t)$	day of time slot t	15

Upper bounds

$\overline{L_{\mathrm{K}}}(k)$	upper bound on $\lvert L_{\mathrm{K,W}}(k,w)\rvert$	16
$\overline{S_{\mathrm{V}}}(v)$	upper bound on number of students in $k \in K_{\mathrm{V}}(v)$	16

Algorithm parameters

av	algorithm variant	89
bj	maximal number of backjumps	87
bjft	maximal number of backjumps between two subjects	87
bt	maximal number of backjumps	87
ni	maximal # iterations tabu search	88
nit	# iterations on tabu list	88
nni	maximal # iterations tabu search without improvement	88
po	pair ordering rule	86
pr	maximal # restarts pair loop	76
ps	pair selection rule	86
sgo	subject group ordering rule	87
so	subject ordering rule	86
sr	maximal # restarts subject loop	81

Parameter values

back	backtracking and backjumping only	89
bottle	bottleneck identification only	89

cnf	least resource conflicts first	87
cns	least resource constraints first	87
constr	constructive heuristic	89
dyn	dynamic pair selection	86
full	backtracking, backjumping, and bottleneck identification	89
LSF	least students first	86
MSF	most students first	86
rnd	random ordering	86
RSO	random subject ordering	86
siz	largest student group first	86
sta	static pair selection	86

Author Index

Samenvatting

In dit proefschrift bestuderen we het construeren van schoolroosters voor middelbare scholen. Een schoolrooster is een schema dat voor elke les aangeeft *wanneer* deze les gegeven zal worden, *welke docent* de les zal geven, *welke leerlingen* en *welk vak* de docent zal onderwijzen en in *welk lokaal* de les zal plaatsvinden. De nadruk van de studie ligt op middelbare scholen in Nederland waar, in tegenstelling tot verschillende andere landen, leerlingen in de bovenbouw hun vakkenpakket tot op zekere hoogte zelf mogen samenstellen. Het vakkenpakket van een leerling bestaat uit een aantal verplichte vakken, zoals Nederlands en Engels, en verschillende keuzevakken. Vaak is het aantal leerlingen dat een bepaald vak wil volgen te groot om door één docent in één klas onderwezen te worden. Voor die vakken worden verschillende klassen ingeroosterd, die we *vakgroepen* noemen. Elke leerling zal elk vak uit zijn vakkenpakket in één vakgroep volgen.

Het maken van een schoolrooster op een Nederlandse middelbare school vindt minstens eens per jaar plaats, namelijk in de zomervakantie voorafgaand aan het nieuwe schooljaar. Het construeren van een acceptabel schoolrooster is om een aantal redenen lastig. Allereerst moet een roosterplanner een keuze maken uit een zeer groot aantal mogelijke roosters. Daarnaast moet hij rekening houden met veel verschillende randvoorwaarden. In Hoofdstuk 1 geven we een opsomming van deze randvoorwaarden. Sommige van deze voorwaarden, de *eisen*, zijn hard en duidelijk te formuleren: elk schoolrooster *moet* hieraan voldoen. Denk bijvoorbeeld aan de eis dat leerlingen en docenten nooit twee lessen gelijktijdig mogen hebben. Andere voorwaarden, de *wensen*, zijn vaag en kunnen niet altijd precies omschreven worden. Zo moet bijvoorbeeld het aantal tussenuren acceptabel zijn en moet een zeker klassenverband bestaan. Daardoor kost het maken van een acceptabel schoolrooster vaak één of meer manweken.

Voordat we kunnen rekenen aan roosterconstructieproblemen moeten we eerst exact beschrijven wat we meenemen in onze analyse en wat we buiten beschouwing laten. Het resultaat is de definitie van een formeel model van ons roosterconstructieprobleem. Dit is het onderwerp van Hoofdstuk 2. In het formeel model geven we exact aan welke schooldata we gebruiken, welke beslissingen door ons genomen moeten worden en aan welke randvoorwaarden we moeten voldoen opdat het uiteindelijke schoolrooster acceptabel is. Ons generieke roosterconstructieprobleem omvat de toewijzingen die elke roosterplanner moet uitvoeren. Daarnaast nemen we in ons model die eisen mee die voor alle roosterplanners van belang zijn. We eisen boven-

dien dat de verschillende vakgroepen voor hetzelfde vak niet te groot mogen worden. Het reduceren van tussenuren laten we buiten beschouwing omdat het niet duidelijk is hoe deze wens eenduidig omschreven moet worden.

We hebben vanuit twee verschillende invalshoeken naar het door ons gedefinieerde roosterconstructieprobleem gekeken. Allereerst hebben we de theoretische complexiteit van ons roosterconstructieprobleem bepaald. Daarnaast hebben we voor twee middelbare scholen geprobeerd roosters voor de bovenbouw te construeren. Hiervoor hebben we een algoritme op een PC geïmplementeerd en een zeer groot aantal runs van het algoritme uitgevoerd.

In Hoofdstuk 3 laten we zien dat het roosterconstructieprobleem dat wij beschouwen formeel lastig is. Dit betekent dat het erg onwaarschijnlijk is dat een algoritme bestaat dat voor elke school snel een acceptabel rooster vindt. Wij tonen bovendien aan dat elke eis afzonderlijk het roosterconstructieprobleem formeel lastig maakt. Concreet bewijzen we dat verschillende theoretische roosterconstructieproblemen, waarin niet alle eisen opgenomen zijn, lastig worden als we een ontbrekende randvoorwaarde toevoegen. In Figuur 3.11 op pagina 52 geven we een overzicht van alle in dit proefschrift besproken complexiteitsresultaten.

In Hoofdstuk 4 beperken we ons tot de kern van ons roosterconstructieprobleem: het toewijzen van leerlingen aan vakgroepen en het construeren van een geschikt rooster. Om die reden laten we de klaslokalen en bloklessen uit het roosterconstructieprobleem buiten beschouwing. Dit roosterprobleem blijkt op zich al formeel lastig te zijn. Thans wordt het op vrijwel alle middelbare scholen opgesplitst in kleinere deelproblemen. In deze aanpak, het *clusteren*, probeert de roosterplanner eerst voor elke afdeling en elk leerjaar afzonderlijk een oplossing te vinden. Vervolgens worden de oplossingen van alle deelproblemen samengevoegd. Deze aanpak heeft als voordeel dat het aantal beslissingen en randvoorwaarden wordt teruggebracht, waardoor de problemen die de roosterplanner moet oplossen kleiner en overzichtelijker worden. Een belangrijk nadeel is dat de roosterplanner daardoor een behoorlijk aantal mogelijke roosters nooit zal beschouwen. Zeker als het aantal roosters dat acceptabel is, dat wil zeggen: aan alle eisen voldoet, klein is, loopt de roosterplanner het risico dat hij met de huidige aanpak geen acceptabel rooster vindt terwijl er wel een bestaat.

Wij hebben ervoor gekozen om het eerdergenoemde kernprobleem op een andere manier aan te pakken. In deze aanpak construeren we eerst een rooster zodanig dat er voor docenten geen conflicten zijn. Het blijkt dat we zo'n rooster in de praktijk snel kunnen construeren, omdat we nog geen rekening hoeven te houden met mogelijke conflicten voor leerlingen. Vervolgens bekijken we de vakken en de leerlingen die de vakken gekozen hebben, één voor één. In elke stap van onze aanpak wijzen we eerst één groep leerlingen met een identiek vakkenpakket aan een bepaalde vakgroep toe. Daarna bepalen we of door deze toewijzing conflicten in het huidige rooster ontstaan. In dat geval proberen we het rooster te repareren door een klein aantal lessen op

andere tijdstippen te geven. Indien nodig evalueren we alle mogelijke vakgroepen voor de groep leerlingen die we op dat moment beschouwen of herzien we eerder genomen beslissingen. We proberen zo elke leerling voor elk vak in het vakkenpakket aan een vakgroep toe te wijzen, en in de tussentijd een rooster zonder conflicten bij te houden.

We kunnen de mogelijke toewijzingen van leerlingen aan vakgroepen wiskundig representeren door middel van een boom. Elke knoop in de boom stelt een toewijzing van leerlingen aan vakgroepen voor, en de bladeren van de boom representeren de oplossingen van ons roosterprobleem. Een tak tussen twee knopen van een boom geeft aan dat twee toewijzingen vrijwel identiek zijn. De extra toewijzing van één groep leerlingen aan een vakgroep maakt het verschil.

Het door ons ontwikkelde algoritme is een *boomzoekalgoritme*. In elke knoop bepalen we of in het huidige rooster conflicten voor leerlingen optreden. In dat geval proberen we het rooster te repareren. Hiervoor gebruiken we een *tabu search* procedure. De details van het boomzoekalgoritme bespreken we in Hoofdstuk 5. Het algoritme heeft een tiental parameters die afgestemd moeten worden. In Tabel 5.1 geven we een opsomming van alle parameters.

We hebben de prestaties van ons algoritme bepaald voor een aantal praktijkproblemen. In Hoofdstuk 6 presenteren en analyseren we de resultaten van deze experimenten. We hebben het boomzoekalgoritme toegepast op zestien verschillende *probleeminstanties*: twee instanties die corresponderen met de volledige bovenbouw van het Cambreurcollege uit Dongen en het Pleincollege Sint-Joris uit Eindhoven, en één instantie voor elk van de bovenbouwniveaus afzonderlijk. Beide scholen zijn middelgrote middelbare scholen met ruim vijfhonderd bovenbouwleerlingen. We hebben geprobeerd alle zestien instanties op te lossen, in het bijzonder de twee bovenbouwinstanties, en hebben een geschikte parameterinstelling van het algoritme bepaald. De beste resultaten zijn verkregen door op een aantal plaatsen in het algoritme randomisatie toe te passen, dat wil zeggen: door willekeurige keuzes te maken, en te herstarten als er in redelijke tijd geen oplossing gevonden kon worden. Het toepassen van *backtracking*, dat wil zeggen: het herzien van recente vakgroeptoewijzingen, is effectief gebleken. De prestatie van het algoritme wordt verder verbeterd door te herstarten met de toewijzing van die leerlingen waarvoor het algoritme zojuist geen rooster zonder conflicten kon vinden.

Het is ons gelukt om een rooster en vakgroeptoewijzing te vinden voor de bovenbouw van het Sint-Joris en voor alle afzonderlijke bovenbouwniveaus van zowel het Cambreur als het Sint-Joris. Voor het Cambreur waren we in staat om 94% van de vakkenpakketkeuzes van alle bovenbouwleerlingen te honoreren. We hebben ondervonden dat HAVO 4 en VWO 5 voor ons algoritme lastige instanties zijn. Bovendien hebben we laten zien dat alle instanties oplosbaar zijn als we een onbegrensd aantal leerlingen aan een vakgroep mogen toewijzen.

We sluiten het proefschrift af met een beschouwing over mogelijke verbeteringen

van ons algoritme en de praktische toepasbaarheid ervan. Omdat we voor een eenvoudige implementatie van ons basisidee hebben gekozen, zijn er verschillende punten waarop het algoritme krachtiger gemaakt zou kunnen worden. Als een roosterplanner ons algoritme heeft toegepast, heeft hij een oplossing voor het kernprobleem. Hij moet daarna nog alle lessen aan lokalen toewijzen en het rooster aanpassen zodat het zo goed mogelijk aan alle wensen voldoet. Roosterplanners op middelbare scholen maken hiervoor gebruik van beslissingsondersteunende systemen, dat wil zeggen: commerciële software die de roosterplanner niet vervangt maar juist ondersteunt bij het construeren van een acceptabel schoolrooster. Ons boomzoekalgoritme kan worden gebruikt als een rekenprocedure in zo'n beslissingsondersteunend systeem.

Dankwoord

Verschillende mensen hebben in de afgelopen jaren een bijdrage geleverd aan mijn promotieonderzoek. Hun bijdrage heeft geleid tot het proefschrift dat nu voor u ligt. Ik wil bij deze van de gelegenheid gebruik maken om hen hiervoor te danken.

Mijn dagelijks begeleider en copromotor, Huub ten Eikelder, ben ik veel dank verschuldigd. In de afgelopen vijf jaar heb ik nooit tevergeefs bij hem aangeklopt. Ik heb veel van zijn manier van onderzoeken en zijn altijd constructieve commentaar op mijn werk geleerd. Een groot deel van de resultaten uit Hoofdstuk 3 zijn te danken aan zijn inspanningen en de vele langdurige discussies die wij over de complexiteitspuzzels hadden.

Daarnaast wil ik mijn beide promotoren heel hartelijk danken voor hun onvoorwaardelijke steun en vertrouwen in mijn werk. Door hun nauwgezette manier van lezen heb ik de eerdere versies van mijn proefschrift essentieel kunnen verbeteren. Dankzij mijn eerste promotor, Emile Aarts, heb ik al snel de door mij gewenste praktische component in mijn onderzoek kunnen introduceren. De keren dat wij samen in de diepte zijn gedoken heb ik als zeer waardevol ervaren. Hem wil ik bovendien danken voor het in contact brengen met mijn nieuwe werkgever, het Centre for Quantitative Methods CQM B.V. Mijn tweede promotor, Jan Karel Lenstra, heeft voor het onderzoek zeer waardevolle opmerkingen, referenties en suggesties gegeven. Ik wil hem in het bijzonder bedanken voor zijn inhoudelijke bijdrage aan het complexiteitshoofdstuk.

I would like to thank the other members of the core committee, professors Van Hee (Technische Universiteit Eindhoven) and de Werra (École Polytechnique Fédérale de Lausanne), for their comments on the draft of this thesis and suggestions for improvements.

Roosterplanning op middelbare scholen is bij uitstek een onderwerp dat je in de praktijk moet leren. In de afgelopen jaren heb ik veel kennis opgedaan tijdens gesprekken met roosterplanners op middelbare scholen. Op het Pleincollege Sint-Joris heb ik een aantal dagen een kijkje in de keuken mogen nemen. De roosterplanners aldaar, Jan van de Donk, Jacques Schoenmaekers en Herman Lembeck waren altijd bereid mijn vele vragen te beantwoorden. Ik dank hun heel hartelijk voor hun gastvrijheid. Hun roosterplanbord is bij deze vereeuwigd op de omslag van dit proefschrift. Ik wil bovendien de heren Philips (Stolberg te Eindhoven), Sprangers (Cambreurcollege te Dongen), Swinkels (Carolus Borromeuscollege te Helmond) en Van Dooremalen (Eckartcollege te Eindhoven) danken voor het met ons delen van hun

praktijkervaring en, indien van toepassing, het beschikbaar stellen van hun roostergegevens. Ik hoop dat mijn werk in de toekomst nog voor hen van nut zal zijn.

Ik heb bovendien met enkele ontwikkelaars van roosterplanningsoftware gesproken. Zij hebben hun software voor mijn onderzoek kosteloos beschikbaar gesteld. In dit kader wil ik Malmberg en de heren Jansen (CQM), Hentzen (ICS Adviseurs) en Boekesteijn (BEP Software) danken. Een bijzonder woord van dank wil ik richten aan Benjamin Jansen waarmee ik een aantal waardevolle discussies over schoolroosterplanning heb gehad.

Een aanzienlijk deel van mijn promotieonderzoek heeft bestaan uit het implementeren van het algoritme in computercode. Door RSI-klachten ben ik niet in staat geweest om alle code handmatig in te voeren. Omdat ik in die periode niet pijnloos kon schrijven leek het programmeren een probleem op zich te worden. Gelukkig bleek het mogelijk om de code met behulp van Dragon Dictate in te spreken. De macro's van lotgenoot Wieger Wesselink zijn daarbij onmisbaar gebleken. Ik wil hem heel hartelijk danken voor het mogen gebruiken van deze macro's.

Ik ben Ramon Clout veel dank verschuldigd. Hij heeft mij enorm geholpen met allerlei (computertechnische)zaken rondom het schrijven en opmaken van dit proefschrift. Daarnaast heeft Martijn Anthonissen mij een aardig eind op weg geholpen bij het maken van figuren.

Mijn collega's van de Technische Universiteit Eindhoven dank ik voor de plezierige werksfeer. Van mijn nieuwe werkgever heb ik alle vrijheid gekregen om de laatste zaken rondom mijn promotie af te handelen. Hiervoor wil ik Jan van Doremalen en Mynt Zijlstra hartelijk danken.

Het doen van promotieonderzoek is niet over rozen gegaan. Gelukkig heb ik de laatste jaren tijd kunnen vrijmaken voor leuke activiteiten buiten mijn werk. Ik wil graag mijn familie en vrienden, waarmee ik de afgelopen jaren het werk en het sociale leven zo goed mogelijk heb kunnen afwisselen, bedanken. Daarbij richt ik me in het bijzonder tot mijn ouders Ad en Ina, mijn broer Dolf en zijn vriendin Isabel, mijn kook-, bordspel-, zwem- en cabaretclubmaatjes Arjan, Martijn en Ramon, Hendriena, Ramiro en Rusmely, en last but not least Danielle. Het werk zit er op, jullie zullen binnenkort (eindelijk!) weer meer van me zien en horen!

Curriculum Vitae

Roy Willemen was born on 20 September 1972 in Breda, the Netherlands. He attended the Dr. Schaepmancollege, currently called Cambreurcollege, in Dongen and obtained his VWO diploma in 1990.

After he finished his secondary education, he studied Mathematics at the Technische Universiteit Eindhoven. During his studies, he helped to organize a study tour to Russia for the study society of mathematics and computer science students. He also was an active member of the editorial board of the study society's journal Supremum and the educational committee of the Mathematics section.

In 1994, he studied Marketing and Business Management at Cornell University in the United States of America. This study was financially supported by a grant of the Stichting Gerrit-Jan Heijn Fellowship. He was placed on the Dean's List of the College of Agriculture and Life Sciences for the Fall semester, 1994.

In 1996, he graduated at the Technische Universiteit Eindhoven. He carried out his graduation project for Speciaalzaken Ahold, a sub-division of Koninklijke Ahold N.V. The project was supervised by prof.dr. J.K. Lenstra and dr. J.A. Hoogeveen of Technische Universiteit Eindhoven en J.J. Bartoo, Logistics Manager of Etos B.V. His Master's thesis was concerned with synergetic benefits in logistics and transportation for Speciaalzaken Ahold. The thesis was nominated for the Ahold Scriptieprijs 1996.

After his graduation, he worked as a Ph.D. student at the Department of Mathematics and Computer Science of the Technische Universiteit Eindhoven. The project was supervised by prof.dr. E.H.L. Aarts, prof.dr. J.K. Lenstra, and dr.ir. Huub ten Eikelder of the same university.

Since 1 January 2002, he works as a logistics consultant for the Centre of Quantitative Methods CQM B.V. in Eindhoven.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-1

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-2

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-5

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-6

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-7

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-9

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty

of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

Stellingen

behorende bij het proefschrift

School timetable construction: Algorithms and complexity

van

Roy Willemen

I

Het kernprobleem waarmee roosterplanners op Nederlandse middelbare scholen geconfronteerd worden bestaat uit twee typen beslissingen. Ten eerste moet elke leerling voor elk vak in het vakkenpakket aan een vakgroep worden toegewezen. Ten tweede moet een acceptabel rooster bepaald worden. De lokaaltoewijzingsbeslissing maakt geen onderdeel uit van het kernprobleem. Als in het rooster op elk tijdstip voldoende vakspecifieke lokalen zijn, kan de lokaaltoewijzing namelijk vaak plaatsvinden *nadat* een acceptabel rooster bepaald is.

II

Op veel middelbare scholen wordt het kernprobleem op eenzelfde manier aangepakt. In deze aanpak, clusteren genaamd, wordt voor elk bovenbouwniveau afzonderlijk bepaald welke vakgroepen gelijktijdig onderwezen zullen worden. Hierdoor sluiten roosterplanners onnodig veel mogelijke roosters uit. Een integrale aanpak, zoals beschreven in Hoofdstuk 5 van dit proefschrift, verdient de voorkeur.

III

Beschouw het bipartiete koppelingsprobleem met relaties. Gegeven zijn een bipartiete graaf $G = (L, T, E)$ met puntverzamelingen L en T en kantverzameling E, een deelverzameling B van L en twee relaties, gegeven door de functies $b : B \to B$ en $bt : T \to T$. Een koppeling in G is een deelverzameling van E waarin elk punt uit L en T hoogstens één keer voorkomt. Gevraagd wordt een koppeling M te vinden waarin elk punt in L voorkomt en die bovendien voldoet aan de voorwaarde

$$\forall_{l \in B} \forall_{t \in T} \quad (l,t) \in M \Rightarrow (b(l), bt(t)) \in M.$$

Dit probleem modelleert het vinden van een toegelaten rooster in het geval van bloklessen en parttimedocenten en is NP-volledig.

TEN EIKELDER, H.M.M. AND WILLEMEN, R.J. [2001], Some complexity aspects of secondary school timetabling problems, in: E.K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers,* Lecture Notes in Computer Science 2079, Springer, 18–27.

IV

Beschouw het meervoudig bipartiete koppelingsprobleem met maxima. Gegeven zijn een bipartiete graaf $G = (V, T, E)$ met puntenverzamelingen V en T en kantverzameling E, een positief geheel getal n, n deelverzamelingen $V_1, \ldots, V_n$ van V en een positief geheel getal u_v voor elke $v \in V$. We identificeren n bipartiete grafen $G_i = (V_i, T, E_i)$ met $1 \leq i \leq n$ en $E_i = (V_i \times T) \cap E$. Een koppeling in G is een deelverzameling van E waarin elk punt uit V en T hoogstens één keer voorkomt. Gevraagd wordt om n koppelingen te vinden in de grafen $G_1, \ldots, G_n$, zodanig dat elk punt in V_i gekoppeld is en dat bovendien elke kant $(v,t) \in E$ in hoogstens u_v koppelingen voorkomt. Dit probleem modelleert het vinden van een gebalanceerde vakgroeptoewijzing voor een gegeven rooster en is NP-volledig.

> TEN EIKELDER, H.M.M. AND WILLEMEN, R.J. [2001], Some complexity aspects of secondary school timetabling problems, in: E.K. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Selected papers*, Lecture Notes in Computer Science 2079, Springer, 18–27.

V

In de literatuur beschreven roosterplanningsproblemen op onderwijsinstellingen worden vaak gerubriceerd aan de hand van drie categorieën: schoolroosterplanning, universiteitsroosterplanning en examenroosterplanning. Daardoor beperken onderzoekers zichzelf in de bespreking van relevante literatuur ten onrechte tot slechts die roosterplanningproblemen die tot dezelfde categorie behoren. Om een juiste selectie te kunnen maken uit de in de literatuur besproken roosterplanningsproblemen is het wenselijk dat de overeenkomsten en verschillen tussen roosterplanningsproblemen duidelijker worden. Dit kan bijvoorbeeld door te rubriceren op basis van de te nemen beslissingen en randvoorwaarden waarmee rekening gehouden moet worden.

VI

De werkmaatschappijen van Speciaalzaken Ahold (SpA) worden bevoorraad vanuit decentrale distributiecentra. SpA kan echter een grote besparing op de transportkosten realiseren door alle winkels van deze werkmaatschappijen vanuit één centraal distributiecentrum te leveren. Zelfs als zo'n distributiecentrum ver verwijderd is van het zwaartepunt van de winkels, is een synergetisch voordeel haalbaar.

> WILLEMEN, R.J. [1996], *Synergieën in de logistiek en distributie, in het bijzonder in het transport, binnen Speciaalzaken Ahold*, Afstudeerverslag, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven.

VII

In de detailhandel wordt nog regelmatig een transportschema op basis van vuistregels bepaald. De transportkosten van deze organisaties kunnen behoorlijk verlaagd worden door toepassing van wiskundige modellen zoals het periodiek voertuigrouteringsprobleem met tijdsvensters en benaderingsmethoden zoals *tabu search*. Een noodzakelijke voorwaarde voor het realiseren van de genoemde kostenbesparing is het beschikbaar hebben van de exacte restricties op de laad- en lostijden bij winkels.

> WILLEMEN, R.J. [1996], *Synergieën in de logistiek en distributie, in het bijzonder in het transport, binnen Speciaalzaken Ahold*, Afstudeerverslag, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven.

VIII

Het leren accepteren van en omgaan met een lichamelijke handicap heeft een grotere positieve invloed op het genezingsproces van mensen met RSI-klachten dan het vervangen van de hulpmiddelen op de werkplek door allerlei ergonomisch verantwoorde apparaten.

IX

Promoveren is topsport. Om tot topprestaties te komen is het belangrijk dat een promovendus, net als elke andere topsporter, intensief begeleid wordt tijdens zijn sportbeoefening.

X

Zelfs kleine instanties van het bipartiete koppelingsprobleem met maxima blijken formeel lastig.

> Toestemming voor huwelijk kroonprins en Máxima, *de Volkskrant*, 4 juli 2001.

XI

Het is wenselijk om, in navolging van de introductie van begrippen als kritische massa en kritiek gebied, het begrip kritische leeftijd voor vrouwen te introduceren. Deze leeftijd scheidt twee intervallen waarbinnen vrouwen liever jonger danwel ouder geschat willen worden.