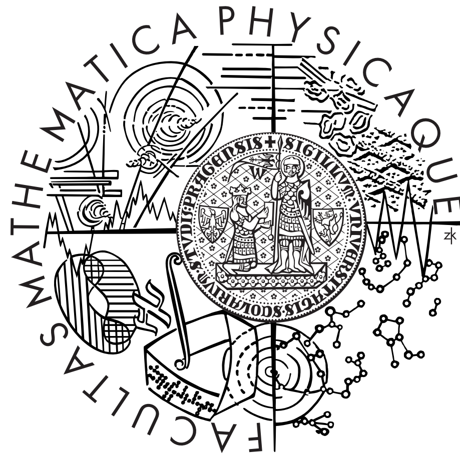
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Martin Všetička

School Timetabling

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: prof. RNDr. Roman Barták, Ph.D.

Study programme: Computer Science

Specialization: Theoretical Computer Science

Prague 2013

I would like to thank my supervisor prof. RNDr. Roman Barták, Ph.D. for his patient guidance and express my appreciation for his constructive suggestions. I'm very obliged to Jaroslav Reichl who helped me considerably with my effort to understand the whole scope of the timetabling problem at the Secondary School of Telecommunication and Broadcasting Technologies in Prague. I would also like to express my gratitude to my parents for their support when I was working on the thesis.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague on August 1, 2013 Martin Všetička

Název práce: School Timetabling

Autor: Martin Všetička

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: prof. RNDr. Roman Barták, Ph.D.

Abstrakt:

V práci se zabýváme těžkým rozvrhovacím problémem jedné české střední školy. V tomto problému je potřeba vyučovací hodiny rozvrhnout do učeben tak, aby byla respektována všemožná omezení, jako například učební plány a disponibilita učitelů. Studujeme existující přístupy, které se používají v problémech školního rozvrhování a ukazujeme, jak reprezentovat uvedený problém v existujícím systému pro školní rozvrhování. Dále pak představujeme softwarový prototyp, který řeší uvedený problém pomocí programování s omezujícími podmínkami. Rozebíráme také problémy související s představenou úlohou, například reprezentaci a konverzi dat.

Klíčová slova: school timetabling, školní rozvrhování, programování s omezujícími podmínkami, reálný problém

Title: School Timetabling

Author: Martin Všetička

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Roman Barták, Ph.D.

Abstract: The thesis deals with a hard real-life school timetabling problem of a Czech secondary school. In this problem, lessons are to be allocated to classrooms while respecting various constraints such as curricula and teacher availability. We study existing approaches used for school timetabling problems and we show how to represent introduced problem in existing school timetabling software. We then present a software prototype that solves introduced problem using constraint logic programming. Related problems, such as data representation and data conversion, are discussed.

Keywords: school timetabling, educational timetabling, constraint programming, real-world problem

Contents

Introduction **5**

1 School timetabling **7**
1.1 Terminology . . . 7
1.2 Problem definition . . . 8
1.3 Time complexity . . . 9
1.3.1 Simple (polynomial) problem . . . 9
1.3.2 Basic Search Problem . . . 10
1.4 XHSTT format . . . 11

2 Problem specification **13**
2.1 School timetabling problem at SSTBTP . . . 13
2.1.1 Introduction . . . 13
2.1.2 Study programs . . . 13
2.1.3 Classes . . . 14
2.1.4 Periods . . . 15
2.1.5 Classrooms . . . 16
2.1.6 Teachers . . . 16
2.1.7 Electives and optional subjects . . . 17
2.1.8 Fortnight lessons . . . 17
2.2 The current approach to timetabling . . . 17
2.3 Constraints . . . 18
2.3.1 Basic constraints . . . 18
2.3.2 Student constraints . . . 18
2.3.3 Classroom constraints . . . 18
2.3.4 Teacher constraints . . . 19
2.3.5 Educational constraints . . . 19
2.3.6 Travel time constraints . . . 19
2.3.7 Constraint satisfaction . . . 19
2.4 Formal model . . . 19

3 Existing approaches **25**
3.1 Sequential methods . . . 25
3.2 Evolutionary algorithms . . . 26
3.3 Metaheuristics . . . 27
3.3.1 Simulated annealing . . . 27
3.3.2 Tabu search . . . 27
3.4 Integer programming . . . 28
3.5 Constraint programming . . . 28
3.6 Comparison of approaches . . . 29

4 Timetabling software **31**
4.1 Open-source timetabling software packages . . . 31
4.1.1 UniTime Timetabler . . . 31
4.1.2 KHE . . . 32

4.1.3 FET 33

5 **Data collection and representation** **35**
5.1 Data collection 35
5.1.1 Data format selection 35
5.1.2 JSON data format 37
5.2 Solver properties 38
5.2.1 Fortnight lessons 38
5.2.2 Timetable compactness 39

6 **FET solver** **41**
6.1 Introduction 41
6.2 Algorithm 41
6.3 Data format 42
6.4 Representation of SSTBTP problem 42
6.4.1 Time grid 42
6.4.2 Students 42
6.4.3 Teachers, subjects, classrooms, buildings and activities 44
6.4.4 Constraints 45
6.5 Missing constraints 46

7 **CLP solver** **47**
7.1 Introduction 47
7.2 Choice of a constraint programming system 47
7.3 Global constraints 48
7.4 Constraint model 49
7.4.1 Representation of time slots 49
7.4.2 Variables and initial domains 49
7.4.3 Lesson decomposition 50
7.4.4 Students and a dual time model 51
7.4.5 Constraints based on the dual time model 52
7.4.6 Bounds on daily workload 52
7.4.7 Constraints for classrooms 53
7.5 Search procedure 53
7.5.1 Heuristic for lesson selection 54
7.5.2 Heuristic for lesson scheduling 54
7.6 Score function 55

8 **Experimental results** **57**
8.1 Data 57
8.2 FET 58
8.2.1 Experiments 58
8.2.2 Conclusions 59
8.3 CLP 59
8.3.1 Experiment 1 – Basic properties of the CLP solver 59
8.3.2 Experiment 2 – Dead end limit 60
8.3.3 Experiment 3 – Adjusted lesson scheduling strategy 61
8.3.4 Experiment 4 – Comparison with the official timetable 62

Conclusions **65**

Further work **67**

List of Abbreviations **75**

Appendix A **77**
A.4 buildings.json 77
A.5 classes.json 77
A.6 macros.json 79
A.7 study.programs.json 80
A.8 subjects.json 82
A.9 teachers.json 82
A.10 timetable.json 83

Appendix B **85**
B.1 FET XML data format in a nutshell 85
B.2 Students 89
B.3 Constraints 89
B.3.1 Basic constraints 89
B.3.2 Student constraints 90
B.3.3 Classroom constraints 91
B.3.4 Teacher constraints 94
B.3.5 Educational constraints 94
B.3.6 Travel time constraints 95

Appendix: CD contents **97**

Introduction

The thesis addresses the problem of constructing timetables for primary and secondary schools. These schools construct timetables every year and the process is typically very lenghty and tedious if users do not employ an automated tool. Unfortunately most of the primary and secondary schools in the Czech Republic (and many other countries) create their timetables still by hand or in a semi-automated way using various programs providing basic graphical tools for a human scheduler [1].

The status quo is caused by several reasons. School timetabling (ST) is computationally complex and it is proven to belong to the set of NP-complete problems [2]. Moreover, a significant obstacle in implementing a solver is the diversity of the problem at different schools. This may be one of the reasons why up until recent years the researchers did not study the problem as extensively as university timetabling and other educational timetabling problems [3][4].

More formally, school timetabling deals with the problem of allocating lessons to rooms and to teachers at primary and secondary schools while respecting constraints such as curricula and teachers availability. The problem belongs to the class of educational timetabling problems. Educational timetabling encompasses university course timetabling, examination timetabling and school timetabling.

A reserved attitude of schools towards existing automated school timetabling solvers is rational. Schools store their data in school management systems. Authors of the solvers typically do not support an import feature because there are simply too many different school management systems. Writing helper programs for exporting schools' data to the solvers' input formats is a reasonable way to tackle the problem. However, it is necessary to inspect the final data for errors. In the end, after several days of work one may find out that the endeavor was futile because there is an aspect of the problem the solver cannot cope with or that the solution returned by the solver is not as good as expected. The whole process of employing the solver instead of a manual scheduler is very time consuming and prospects are uncertain. Therefore, we are convinced that the thesis addresses an important problem because it presents a case study of a real-life school timetabling problem including all the necessary work from data collection to running an automatic solver. A comparison of problem-solving techniques was made and the chosen method was implemented in a software prototype.

The thesis is organized as follows: Chapter 1 presents the school timetabling in detail. In Chapter 2 we introduce a real-life school timetabling problem we solved. Chapter 3 is concerned with the existing approaches for solving school timetabling problems. In Chapter 4 we present school timetabling software packages and we discuss their capabilities to solve the introduced school timetabling problem. In Chapter 5 we propose a data format for the problem introduced in Chapter 2. In Chapter 6 we propose a solution based on an existing school timetabling solver. Chapter 7 proposes a solver based on constraint programming. and in Chapter 8 we discuss how satisfactory are our solutions.

1. School timetabling

School timetabling deals with the problem of allocating lessons to rooms and to teachers at primary and secondary schools while respecting various constraints. The problem belongs to the class of educational timetabling problems.

1.1 Terminology

Since terminology used in school timetabling problems is not well established, this section introduces definitions of all important concepts. The definitions are loosely based on paper [5].

Subject refers to a branch of human knowledge such as physics, mathematics, English language, chemistry, etc.

Optional subject is a subject that a student has to choose from a predefined set of subjects. For example, a student is supposed to take a foreign language and it may be Spanish, French, or Chinese.

Elective subject is a subject that a student is free to take or not.

Class is a group of students that meets with teachers at certain times to learn particular subjects. The division of students into classes is responsibility of the school administration.

Subgroup refers to a particular group of students from one class.

Grade refers to a stage of study. A grade corresponds to one year of school tuition. For a student the grade represents progress towards graduation.

Lesson is a scheduled meeting of a set of students from one class and a teacher in a classroom. All data that are not predefined for lessons need to be assigned during timetabling process[1]. An example of a fully specified lesson is: class 1A meets with John Doe to study mathematics in classroom 1 on Monday at 9:00.

Lecture refers to an educational talk to an audience, especially one of students in a university. Lectures involve much less interaction than lessons. The term *lecture* will not be used further in this work.

Period refers to a time slot in a timetable. School administrations divide days into periods and lessons are to be assigned to these periods.

Timetable is defined by Marte [6] as *a rectangular time grid that divides the planning period into disjoint time intervals of equal duration which are called periods*. For a typical example see Table 1.1.

Block lessons refer to lessons that are taught consecutively to the same class and all the lessons take place in the same classroom. Blocks are useful for lessons such as chemistry for which students need to change clothes, set up and clean up laboratory equipment, etc. Blocks minimize these routine activities. We will use the term even for lessons that take only one period.

Idle period for a student (a teacher) is a period in a timetable for which the student (the teacher) does not have any lesson scheduled, and moreover, there is at least one lesson before and after the mentioned period to which the student

[1] In this thesis, a starting time and a classroom has to be assigned to each lesson during the timetabling process since the other data for lessons are provided. However, school timetabling problems differ. For example, teachers need to be assigned to lessons in some school timetabling problems.

Day / Period	1	2	3	4	5	6	7	8
Monday								
Tuesday								
Wednesday								
Thursday								
Friday								
Saturday								
Sunday								

Table 1.1: An example of a timetable

(teacher) has to attend. Idle periods may be used for lunch breaks but having too many idle periods in a timetable is not desirable as it is hard for students to use the time effectively.

Compact timetable is a such timetable that contains low number of idle periods. We say that timetable A is *more compact* that timetable B if A has less idle periods than B.

Resource is either a class or a teacher or a classroom.

Constraint is a limitation of a resource usage determined by its capabilities or by an educational system. For example, a classroom can accommodate up to 16 students.

Soft constraint (also *desire*) refers to a constraint that should be respected in a timetable. For example, a teacher wishes not to teach on Friday. Violations of the soft constraint are penalized but the violation itself does not lead to an invalid timetable. Nevertheless, in many real-life problems it often happens that soft constraints cannot be all satisfied.

Hard constraint (also *requirement*) is a constraint that must be respected in a timetable in order for the timetable to be valid. For example, a teacher cannot teach more than one class at once.

Feasible timetable refers to a timetable that satisfies all hard constraints.

Willemen's definition [7]: *Educational timetabling is the sub-class of timetabling for which the events take place at educational institutions.* If the events are examinations (lessons or lectures) then we talk about *examination timetabling* (*school timetabling* or *university course timetabling*).

Scheduling cycle refers to a period for which a given timetable was devised. Typically, school timetabling uses a one-week scheduling cycle. However, there are also timetables that employ a two-week scheduling cycle and in that case a student needs to know if it is an odd week or an even one.

School timetabling solver (ST solver) refers to a software program that attempts to solve school timetabling problems represented in an input format that the program supports.

1.2 Problem definition

Formal definition of timetabling was given by A. Wren in paper [8]:

Definition 1. *Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly*

as possible a set of desirable objectives.

We can define *school timetabling* by restricting Definition 1 as follows:

Definition 2. *School timetabling refers to timetabling specific for primary and secondary schools.*

It remains to explain what is meant by ***resource***, ***object*** and ***objective*** in the previous two definitions. In school timetabling the ***resource*** is identical to our definition of *resource*, i.e. a class or a teacher or a classroom. ***Object*** represents a *lesson*. The ***objective*** of school timetabling is a *feasible timetable* that satisfies soft constraints as much as possible.

1.3 Time complexity

The following two sections show the threshold where a school timetabling problem becomes intractable. They are based on the paper [9] by A. Schaerf where a more in-depth discussion of the time complexity of ST is presented.

1.3.1 Simple (polynomial) problem

Simple (polynomial) problem was defined by de Werra [10]. It is a formal model representing a common core of the most school timetabling problems, i.e. *assign lessons to periods in such a way that no teacher or class is involved in more than one lesson at a time*[9]. It is defined by the following constraints:

$$\sum_{k=1}^{p} x_{ijk} = r_{ij} \quad (i = 1..m; j = 1..n) \tag{1.1}$$

$$\sum_{j=1}^{n} x_{ijk} \leq 1 \quad (i = 1..m; k = 1..p) \tag{1.2}$$

$$\sum_{i=1}^{m} x_{ijk} \leq 1 \quad (j = 1..n; k = 1..p) \tag{1.3}$$

$$x_{ijk} = 0 \text{ or } 1 \quad (i = 1..m; j = 1..n; k = 1..p) \tag{1.4}$$

where the meaning of the variables is as follows:

- $x_{ijk} = \begin{cases} 1 & \text{if class } c_i \text{ and teacher } t_j \text{ meet at period } k \\ 0 & \text{otherwise} \end{cases}$
- $c_1, \ldots, c_m$ represents m classes,
- $t_1, \ldots, t_n$ represents n teachers,
- $R_{m\times n}$ is so called *requirement matrix* and $r_{ij} \in R_{m\times n}$ is the number of lessons given by teacher t_j to class c_i, and
- $1, \ldots, p$ represents p periods.

Equation 1.1 specifies that class c_i has to meet with teacher t_j for r_{ij} times to fulfill requirement requests. Each teacher can be scheduled for at most one lesson per period. This requirement is expressed by equation 1.2. Equation 1.3 is analogous case to equation 1.2 and it constrains scheduling of classes. The objective is then to find x_{ijk} for $i = 1..m; j = 1..n; k = 1..p$ such that all the constraints are satisfied.

Hopcroft and Karp [11] proved that the mentioned problem can be solved in polynomial time with respect to n, m and p. The solution was based on a bipartite multigraph and on finding a sequence of maximal matchings.

This Simple (polynomial) problem can be solved in polynomial time but unfortunately the model is too restricted for most real-life problems including our task.

1.3.2 Basic Search Problem

Junginger [12] formulated the following formal model that extends the simple (polynomial) model by adding two binary matrices $T_{m \times p}$ and $C_{n \times p}$ that express teacher and class (un)availability. The constraints are:

$$\sum_{k=1}^{p} x_{ijk} = r_{ij} \quad (i = 1..m; j = 1..n) \tag{1.5}$$

$$\sum_{j=1}^{n} x_{ijk} \leq t_{ik} \quad (i = 1..m; k = 1..p) \tag{1.6}$$

$$\sum_{i=1}^{m} x_{ijk} \leq c_{jk} \quad (j = 1..n; k = 1..p) \tag{1.7}$$

$$x_{ijk} = 0 \text{ or } 1 \quad (i = 1..m; j = 1..n; k = 1..p)$$

where the meaning of new variables is as follows:

- $t_{ik} = \begin{cases} 1 & \text{if teacher } t_i \text{ is available at period } k \\ 0 & \text{otherwise} \end{cases}$

- $c_{jk} = \begin{cases} 1 & \text{if class } c_j \text{ is available at period } k \\ 0 & \text{otherwise.} \end{cases}$

Equation 1.5 is identical to equation 1.1. Equation 1.6 (1.7) specifies that each teacher (class) can be scheduled for at most one lesson per period if the teacher (the class) is available.

Even et al. proved in paper [13] that the Junginger's problem is NP-complete through reduction from 3-SAT.

The result is important for the purpose of the thesis because teachers do specify when they cannot teach in our problem instance and also because the equations formally describe basic constraints (cf. Section 2.3.2) that are imposed on our problem instance.

1.4 XHSTT format

A school timetabling problem is a well-known problem but the problem definitely cannot be considered solved. The research in the area of school timetabling is still active. However, up until recently, there was no standardized data format in order to exchange ST problem instances. As a consequence, a comparison or a benchmarking of researchers' solvers was problematic. Therefore, it was reasonable to try using several solving techniques on one specific problem instance, as opposed to attempting to solve several problem instances with one solving method.

XHSTT is a joint effort of Gerald Post et al. to remedy the situation. It is an XML based data format for high schools that attempts to become a standard for school timetabling. In fact, the contribution of Post et al. is not only the data format itself. The researchers specified the common core of the school timetabling problems such that the data format can be widely used. The data format evolved from its first publication in paper [14] in 2008 to the current specification [15] that is available online. There is also an online XHSTT archive [16] that currently provides 39 problem instances, real and artificial ones, of school timetabling problems from many countries. The archive and the standard format simplify benchmarking. Unfortunately, there are still very few solvers that support the XHSTT data format. We were able to obtain and try a few solvers from the International Timetabling Competition 2011 [17] that support the XHSTT data format. However, the solvers are proof of concept programs rather than production quality solvers.

2. Problem specification

The thesis is concerned with the school timetabling problem at the Secondary School of Telecommunication and Broadcasting Technologies at Panská 3, Prague (SSTBTP). More precisely, the aim of the work is to propose and implement a solving method for the problem in order to create a timetable for each teacher and for each class such that the hard constraints are satisfied and such that the best effort is put to satisfy the soft constraints.

The school was chosen because we believe that school timetabling at SSTBTP is a hard problem in comparison with other secondary schools in the Czech Republic. The other secondary schools in the Czech Republic typically do not own more than one school building and therefore they do not need to minimize movements between buildings. Moreover, SSTBTP is a technical school and it has many specialized classrooms. Since capacities of the specialized classrooms are limited, classes need to be divided into subgroups. It is hard to preserve compactness of a timetable when there are many of these divisions.

As a consequence, a successful solving method for SSTBTP may prove useful for the other Czech schools as well.

2.1 School timetabling problem at SSTBTP

This section characterizes SSTBTP from the perspective of timetabling and it describes how the school operates.

2.1.1 Introduction

SSTBTP is a technical secondary school that provides education in broadcasting and telecommunication technologies. The school offers four years of education that is finished with a school leaving exam called *maturita*. The school is unique, from the timetabling perspective, by the fact that teaching takes place at two sites simultaneously. This is not common for secondary schools in the Czech Republic.

We will refer to the locations by the name of the street the site is located at as *Panská* and *Štupartská*. Both sites are located in Prague and the walking distance between them is 10 minutes.

2.1.2 Study programs

There are four study programs that are provided by the school, namely *Technical Lyceum*, *Global Network Technologies*, *Communication and Multimedia* and *Film and Television Production*. A study program represents a curriculum for entire study (i.e. four years). A student enrolls in a study program by submission of a secondary school application. The study program specifies how many lessons have to be taught per week for each particular subject[1].

[1] The numbers are not necessarily integers. For example, civics is supposed to be taught 1.5 times per week and the meaning is that one week students take just one lesson of civics and the other week they take 2 lessons of civics.

2.1.3 Classes

A *class* is always a group of students who study the same study program and who are in the same grade[2]. The number of classes changes from year to year because it depends on the number of admitted students. The school administration is in charge of sectioning students into classes. The maximum and the most common number of students in a class is 32 students. The limit represents a maximum seating capacity of a regular classroom in the school. The number of students in a class may change every year. There are, for example, some students who need to repeat grades or students who leave school before graduation.

Typically, a class meets with a teacher in a classroom to attend a lesson. However, it is not always possible or desirable for all students of the class to take the lesson at one time. For this reason students of the class can be divided by the school administration into groups called *subgroups* and the original lesson has to be taught as many times as the number of subgroups is. Moreover, all these new lessons can have a different teacher, they do no need to take place at the same time and the new lessons require empty classrooms.

We will use the term *partition* for a set of subgroups of a class so that each student of the class belongs to exactly one subgroup of the partition. If students of a class need to be divided for a lesson the division is always a partition. One partition can be used for several lessons of a class.

Partitions are necessary for two reasons. First, specialized classrooms' capacities are typically lower than the number of students in a class. For example, the capacity of gymnasiums is 16 students. If a class has more than 16 students then it is necessary to create a partition where each subgroup is composed of 16 students or less. The school administration specifies partitions for classes' lessons. A partition is created so that each subgroup of the partition has approximately the same number of students. Second, there are lessons where students need to interact with their teachers more intensely. For example, teachers of English lessons have to talk with students. Obviously, it would be hard to talk with 32 students in 45 minutes. Partitions help to alleviate this problem.

A partition is a tool used to solve classroom constraints and to improve quality of education for students. However, partitions complicate the timetabling process. Physical education can serve as an example. Typically, a partition with subgroups *boys* and *girls* is used for PE lessons. PE lessons for the two subgroups do not have to take place at the same time. Actually, if the school administration assign the same teacher to both subgroups of PE lessons then the lessons cannot take place at the same time.

All students of SSTBTP study English. English lessons of a class with more than 16 students are divided into two halves – traditionally named A1 and A2 – according to students' knowledge of English language. We call the partition *language1*. Obviously, the lesser the number of partitions, the easier it is to solve the timetabling problem. Therefore, a convention was devised: *Use language1 partition whenever a class needs to be divided into halves and when it does not matter which student is in which subgroup.*

The *language1* partition of a class typically covers many cases where the whole class cannot attend a lesson at one time. Yet, there are special cases, such as

[2] cf. Terminology section 1.1

photographic practice that require a class to be divided into quarters. Another special case is the physical education. Students are put to subgroups according to gender, and on top of that only students fit to exercise[3] are taken into account.

It is important to note that for the scheduling purposes it is not generally known which student attends which level of English because the timetabling process takes place in June and the assessment test for new students is taken in September. This means we are not able to work with students on a finer level of granularity than on a subgroup level. Moreover, the problem input does not contain an information how students are distributed to partitions' subgroups. In this regard, we only know the names of partitions and their subgroups.

Because the school administration ultimately decides about classes' partitions, generating partitions automatically for each class according to predefined rules, while partially possible, is not practical for such a low number of classes. Therefore, for the purposes of scheduling the partitions are considered to be a part of the problem input.

Since we do not know which students belong to which subgroups, it may be very well possible that lessons of some two subgroups from different partitions of a class could take place at the same time. The school administration at SSTBTP decided not to support this arrangement. Too much extra data would be required for the feature and there is no convincing justification for it.

An auxiliary term we will use is *study group*. Study groups of a class denotes students who were assigned to the same subgroups in all partitions of the class. Consequently, students of a class belonging to the same study group of the class follow the same timetable.

2.1.4 Periods

Teaching takes place from Monday to Friday. Teaching at Panská may take place from 7:00 to 17:40 (i.e. 12 periods). On the other hand, teaching at Štupartská takes place from 7:10 to 17:00 (i.e. 11 periods). The complete list of periods can be seen in Table 2.1.

Period	Panská	Štupartská
1	7:00 - 7:45	7:10 - 7:55
2	7:50 - 8:35	8:00 - 8:45
3	8:45 - 9:30	8:55 - 9:40
4	9:50 - 10:35	10:00 - 10:45
5	10:45 - 11:30	10:55 - 11:40
6	11:40 - 12:25	11:50 - 12:35
7	12:35 - 13:20	12:45 - 13:30
8	13:30 - 14:15	13:40 - 14:25
9	14:20 - 15:05	14:30 - 15:15
10	15:15 - 16:00	15:25 - 16:10
11	16:05 - 16:50	16:15 - 17:00
12	16:55 - 17:40	

Table 2.1: List of periods at Panská and Štupartská

[3]Some students do not exercise because of health issues.

As the reader can see from the table, the period times are shifted by ten minutes in Panská and Štupartská. The decision was made by the school administration in order to ease the problem of traveling between school buildings. This way it is possible for classes to move from Panská to Štupartská without the need of an idle period. It is not, however, possible in the opposite direction. If the period times were the same then it would not be possible to travel between buildings without an idle period in either way.

You can also notice that all the periods have the same duration. However, it is customary to group some consecutive periods to *blocks* to teach some subjects. Timetabling solvers typically do not support different period durations.

Idle periods are permitted but the timetable should be rather compact. The reason is that students are not able to use the free time effectively as teachers can.

2.1.5 Classrooms

Classrooms are divided into three types:

- regular classrooms for up to 32 students,
- specialized classrooms for up to 16 students (language classrooms, gymnasiums, etc.) and
- specialized classrooms for up to 8 students (photographic studio, music studio, etc.).

A specialized classroom may be required or preferred for a lesson. The lessons of physical education require a gymnasium. Conversely, the lessons of English language should be taught in a language classroom but it is not required. Note that the school possess several language classrooms and even several gymnasiums, an ST solver needs to take into account the fact.

If a regular classroom is sufficient for a lesson then a regular classroom should be preferred. It would be odd to take history lessons in an electrotechnical laboratory or in a gymnasium. The constraint also prevents damage of school property as specialized rooms are expensive to equip.

Each class is assigned a classroom called a *home classroom.* This classroom is used as a default option where lessons should take place whenever it is possible. The convention is useful for classes because it eliminates moves of classes between classrooms. Moreover, the assignments break symmetry of the ST problem and therefore make it easier to solve.

2.1.6 Teachers

The school administration negotiates assignment with each teacher. As the agreement of both parties takes place before the timetabling process, the agreement is still preliminary because it may turn out to be problematic. As a consequence some lessons are preassigned to certain time slots in the final timetable.

Formally, we can represent assignment as a 4-tuple *(teacher, lessons, class, subgroup).* So for example assignment *(John Doe, English lessons, 1A, A2)* means that the teacher John Doe is supposed to teach English lessons in the subgroup A2 of the class 1A.

2.1.7 Electives and optional subjects

The students at SSTBTP are quite restricted as for the choice of their subjects. The students can attend optional subjects in the 3rd and 4th year of their studies. Elective subjects are intended only for 4th year students. However, the ratio of the number of compulsory subjects to the number of elective and optional subjects is approximately 10:1. This model is rather similar to the Greek school timetabling problem [2] than to the Dutch model that allows a great amount of freedom for students [18].

2.1.8 Fortnight lessons

Most of the lessons are taught on a weekly basis. However, there are lessons, such as chemistry labs and physics practice, that are taught biweekly. The school currently uses a one-week scheduling cycle and teachers and students are briefed about which lessons are fortnight ones. Naturally this leads to the question if two-week scheduling cycle or one-week scheduling cycle should be employed. The former approach is impractical because many constraints need to be added twice, thus making the problem even more complex[4]. We decided to use the latter approach in the formal model of the timetabling problem because our problem does not contain many fortnight lessons. Moreover, timetables can be often improved by joining two fortnight lessons into one regular lesson in order to obtain a more compact timetable.

2.2 The current approach to timetabling

The timetabling is currently done manually at SSTBTP. In the past, the timetabling process involved a plan of school buildings and pieces of paper, representing lessons, that were placed on classrooms. Nowadays, the school uses the program called Bakaláři[5] with an integrated timetabling module. However, the program provides only a basic timetabling heuristic that is based on placing the most constrained lessons first. It is certainly a more sophisticated approach of solving the problem than the method with pieces of paper but this solver proved to be inadequate to create satisfactory timetable for SSTBTP. Yet, the program is still used because it can verify basic constraints such as classroom availability when the human scheduler creates a timetable manually.

The time required by a human scheduler to create a final timetable is about two weeks. The work results in a feasible timetable. On the other hand, the problem has typically many solutions and we would like to know how good the solution provided by the human scheduler is. This is difficult when we have only one solution at our disposal. Moreover, the human scheduler is not able to take all soft constraints into consideration, therefore some soft constraints are discarded at the beginning of the timetabling process.

It is worth mentioning that timetables from previous years cannot be reused. The numbers of students in study programs change because of many external factors like students' preferences, demographic changes, state subventions for

[4]Current software packages mostly do not directly support a two-week scheduling cycle.

[5]`http://www.bakalari.cz/` - website is only in the Czech language

schools, etc. Moreover, the staff is subject to change as a teacher may retire or leave for any other reason. The external teachers typically do have a contract for one year and if they quit then the subject may not be taught any more. All of these impediments make reuse of previous timetables impractical.

2.3 Constraints

This section introduces a complete list of constraints that are imposed on the timetabling problem of SSTBTP by the school administration.

All constraints are denoted by one of the following designation: *Cz*, *HCx* or *SCy*. The last two stand for *hard constraint* and *soft constraint*, respectively. *C* stands for *constraint* and it may be either a soft constraint or a hard one, depending on the decision of the school administration in each particular case. The x, y and z are counters and they serve only for reference purposes.

2.3.1 Basic constraints

HC1 At most one lesson per period must be scheduled for each teacher.

HC2 At most one lesson per period must be scheduled for each classroom.

HC3 At most one lesson per period must be scheduled for each student.

2.3.2 Student constraints

HC4 The maximum student workload is 9 lessons per day.

HC5 If a student attends 8 or more lessons on a certain day she has to have a break. It means that a student can take at most 7 lessons consecutively.

SC1 A final timetable should be compact for students.

2.3.3 Classroom constraints

HC6 Each classroom is specified by its seating capacity and the capacity cannot be exceeded.

C1 Specialized rooms are required or desired for some lessons.

HC7 Some specialized classrooms are inappropriate[6] for some lessons and thus they are not allowed for the lessons.

HC8 Classrooms are available according to opening hours of their respective buildings (cf. Table 2.1).

SC2 Home classroom should be taken into account and lessons should be scheduled to the home classroom. The importance of this constraint is low.

[6]For example, a gymnasium is an inappropriate classroom for any subject other than physical education.

2.3.4 Teacher constraints

HC9 The maximum teacher workload per day is 8 lessons.

HC10 A teacher must not be scheduled for times when he or she is unavailable.

2.3.5 Educational constraints

HC11 Each class must be taught in conformity with its study program. Especially, the number of lessons for each particular subject in the timetable must be in accordance with the study program specification.

HC12 Block constraints must be respected, i.e. some lessons of certain subjects must be taught in a *block*.

2.3.6 Travel time constraints

SC3 The number of moves between the school buildings should be as low as possible for a student.

HC13 When a student needs to move from *Štupartská* to *Panská*, there has to be an idle period for that purpose.

2.3.7 Constraint satisfaction

The definition of a *hard constraint* dictates that the constraint has to be satisfied in a feasible timetable. On the other hand, soft constraints does not affect timetable's feasibility. Yet, an importance of particular soft constraints differs. For this reason many timetabling solvers come with the concept of *numeric weights* that are assigned to soft constraints. The weights express then how important the satisfaction of a soft constraint is with respect to other constraints.

The school administration does not follow a methodology specifying the importance of particular soft constraints. We are therefore left with a wide variety of options in this matter.

2.4 Formal model

We denote by

- $\mathcal{D} = \{1, 2, \ldots, m\}$ a set of days,
- $\mathcal{P} = \{1, 2, \ldots, n\}$ a set of periods,
- $\mathcal{T}$ a set of teachers,
- $\mathcal{S}$ a set of students,
- $\mathcal{C}$ a set of classes,
- $\mathcal{L}$ a set of block lessons (as defined in Section 1.1),
- $\mathcal{R}$ a set of classrooms,

- $\mathcal{B}$ a set of buildings,
- $roomCap : \mathcal{R} \rightarrow \mathbb{N}$ a function specifying capacities of classrooms,
- $homeRoom : \mathcal{C} \rightarrow \mathcal{R}$ a function specifying home classrooms for classes,
- $tAvail : \mathcal{T} \times \mathcal{D} \times \mathcal{P} \rightarrow \{0, 1\}$ a function specifying for each teacher if he or she is available for teaching at a given day and period,
- $bld : \mathcal{R} \rightarrow \mathcal{B}$ a function specifying for each classroom building the classroom is located in, and
- $O : \mathcal{B} \rightarrow 2^{|\mathcal{P}|}$ a function specifying for each building when its classrooms are available.

In the following text we assume that $\mathcal{B} = \{\text{Panská}, \text{Štupartská}\}$.

Definition 3. *We define* class *$C \in \mathcal{C}$ as a group of students $C \subseteq \mathcal{S}$ such that all students of class C are in the same grade. Moreover, each student $s \in \mathcal{S}$ belongs to exactly one class.*

Definition 4. *Subgroup S of class $C \in \mathcal{C}$ is defined as subset $S \subseteq C$.*

Notation 1. *We denote C_S a set of all subgroups defined for class $C \in \mathcal{C}$ in the problem input by the school administration.*

Definition 5. *Partition P of class $C \in \mathcal{C}$ is a set of subgroups $\{S_1, S_2, \ldots, S_n\} \neq \emptyset$ of class C such that $\forall s \in C\, \exists! i \in [1, n] : s \in S_i$.*

Notation 2. *We denote C_P a set of partitions of class $C \in \mathcal{C}$ defined in the problem input by the school administration.*

Note: $\{C\}$ for $C \in \mathcal{C}$ is a trivial partition and $\forall C \in \mathcal{C} : \{C\} \in C_P$.

Notation 3. *We denote $C_G = \bigcup_{P \in C_P} \{S \mid S \in P\}$ for each $C \in \mathcal{C}$. An element of C_G is called* study group.

Example 1. *Let $C \in \mathcal{C}$ and let $C_P = \{Class, Language, PhysicalExercise\}$ where the partitions are defined as follows:*

$$
\begin{aligned}
Class &= \{C\}, \\
Language &= \{EnglishBeginners, EnglishIntermediate\}, \text{ and} \\
PhysicalExercise &= \{Boys, Girls\}.
\end{aligned}
$$

Study groups are then:

$$
\begin{aligned}
C_G = \{ & \\
& \{Class, EnglishBeginners, Boys\} \\
& \{Class, EnglishBeginners, Girls\}, \\
& \{Class, EnglishIntermediate, Boys\}, \\
& \{Class, EnglishIntermediate, Girls\} \\
\}. &
\end{aligned}
$$

Definition 6. *Block lesson* $l \in \mathcal{L}$ *is the 6-tuple:*

$$(t, C, S, n, A, B)$$

where

- $t \in \mathcal{T}$ *is a teacher teaching lesson* l*,*
- $C \in \mathcal{C}$ *is a class,*
- $S \in C_S$ *is a subgroup*[7] *of class* C *attending lesson* l*,*
- $n \in \{1, 2, \ldots, |\mathcal{P}|\}$ *is a number of consecutive periods that block lesson* l *takes,*
- $\emptyset \neq A \subseteq R$ *represents permitted classrooms for lesson* l*, and*
- $B \subseteq A \subseteq R$ *represents preferred classrooms for lesson* l*.*

We will refer to particular elements of the 6-tuple l *by subscripts – i.e.* l_t, l_C, l_S, l_n, l_A *and* l_B*. A classroom and a starting time are assigned to block lesson* l *by an assignment function (see below).*

Note: The previous definition does not allow fortnight lessons and thus it is necessary to deal with them in a data preprocessing.

Definition 7. *Problem is defined by 10-tuple*

$$(\mathcal{D}, \mathcal{P}, \mathcal{C}, \mathcal{T}, \mathcal{L}, \mathcal{R}, roomCap, homeRooms, tAvail, B).$$

We define assignment function $g : \mathcal{L} \rightarrow (\mathcal{D}, \mathcal{P}, \mathcal{R})$ *that assigns to each block lesson a day, a starting period and a classroom. An assignment function represents a feasible solution if all the hard constraints from Definition 9 are satisfied.*

Note: Set $\mathcal{L}$ has to be defined in accordance with study programs to satisfy HC11 and HC12.

Definition 8. *Let* f *be an assignment function. We define auxiliary variables as follows:*

- $x_{CStdpr} = \begin{cases} 1 & l = (t, C, S, n, A, B) \in \mathcal{L} \wedge f(l) = (d_s, p, r) \wedge d_s \leq d < d_s + n \\ 0 & otherwise \end{cases}$

  The variables say if subgroup S *of class* C *and teacher* t *meet at day* d *and period* p *in classroom* r*.*

- $\forall C \in \mathcal{C}, G \in C_G, d \in \mathcal{D}, p \in \mathcal{P} : y_{CGdp} = \sum_{t \in \mathcal{T}} \sum_{r \in R} \sum_{S \in G} x_{CStdpr}$

  Variables y_{CGdp} *represent if study group* G *of class* C *attends a lesson at period* p *and day* d*.*

[7] Note that the *subgroup* definition allows $S = C$.

- $\forall C \in \mathcal{C}, G \in C_G, d \in \mathcal{D}, p \in \mathcal{P}, r \in \mathcal{R} : z_{CGdpr} = \sum_{t \in \mathcal{T}} \sum_{S \in G} x_{CStdpr}$
  Variables z_{CGdpr} represent if study group G of class C attends a lesson at period p and day d in classroom r.

Definition 9. *Let f be an assignment function.* Hard constraints *are specified as follows:*

$$\text{HC1: } \sum_{C \in \mathcal{C}} \sum_{S \in C_S} \sum_{r \in \mathcal{R}} x_{CStdpr} \leq 1 \qquad \forall t \in \mathcal{T}, d \in \mathcal{D}, p \in \mathcal{P}$$

$$\text{HC2: } \sum_{t \in \mathcal{T}} \sum_{C \in \mathcal{C}} \sum_{S \in C_S} x_{CStdpr} \leq 1 \qquad \forall d \in \mathcal{D}, p \in \mathcal{P}, r \in \mathcal{R}$$

$$\text{HC3(1): } y_{CGdp} \leq 1 \qquad \forall C \in \mathcal{C}, G \in C_G, d \in \mathcal{D}, p \in \mathcal{P}$$

$$\text{HC4: } \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} \sum_{S \in G} \sum_{p \in \mathcal{P}} x_{CStdpr} \leq 9 \qquad \forall C \in \mathcal{C}, G \in C_G, d \in \mathcal{D}$$

$$\text{HC5: } \sum_{t \in \mathcal{T}} \sum_{r \in \mathcal{R}} \sum_{S \in G} \sum_{p=1}^{|P|-7} \sum_{i=p}^{p+7} x_{CStdir} \leq 7 \qquad \forall C \in \mathcal{C}, G \in C_G, d \in \mathcal{D}$$

$$\text{HC6: } f(l) = (d, p, r) \wedge roomCap(r) \geq |l_S| \qquad \forall l \in \mathcal{L}$$

$$\text{HC7: } f(l) = (d, p, r) \wedge r \in l_A \qquad \forall l \in \mathcal{L}$$

$$\text{HC8: } x_{CStdpr} = 1 \implies p \in O(bld(r)) \qquad \forall C \in \mathcal{C}, S \in C_S, t \in \mathcal{T}, d \in \mathcal{D}, p \in \mathcal{P}, r \in \mathcal{R}$$

$$\text{HC9: } \sum_{C \in \mathcal{C}} \sum_{r \in \mathcal{R}} \sum_{S \in C_S} \sum_{p \in \mathcal{P}} x_{CStdpr} \leq 8 \qquad \forall t \in \mathcal{T}, d \in \mathcal{D}$$

$$\text{HC10: } \sum_{C \in \mathcal{C}} \sum_{r \in \mathcal{R}} \sum_{S \in C_S} x_{CStdpr} \leq tAvail(t, d, p) \qquad \forall t \in \mathcal{T}, d \in \mathcal{D}, p \in \mathcal{P}$$

$$\text{HC13: } \forall d \in \mathcal{D}, p_1 \in \mathcal{P}, p_2 \in \mathcal{P}, p_1 + 1 = p_2, C \in \mathcal{C}, G \in C_G, r_1 \in \mathcal{R}, r_2 \in \mathcal{R}, bld(r_1) = \text{Štupartská} \wedge bld(r_2) = \text{Panská} : z_{CGdp_1r_1} + z_{CGdp_2r_2} \leq 1$$

$$\text{HC3(2): } \forall d \in \mathcal{D}, p \in \mathcal{P}, C \in \mathcal{C}, P_1 \in C_P, P_2 \in C_P, P_1 \neq P_2, S_1 \in P_1, S_2 \in P_2 : \sum_{r \in \mathcal{R}} \sum_{t \in \mathcal{T}} x_{CS_1tdpr} + \sum_{r \in \mathcal{R}} \sum_{t \in \mathcal{T}} x_{CS_2tdpr} < 2$$

where HC3(1) and HC3(2) are two inequations that define HC3 constraint. HC3(2) specifies which subgroups' lessons can take place at the same time.

Definition 10. *Let f be an assignment function.* Soft constraints *are specified in the following manner.*

SC1: Compactness of a timetable is given by the number of idle periods for all study groups of classes:

$$Cost_{SC1} = \sum_{\substack{C \in \mathcal{C} \\ G \in C_G \\ p_1 \in \mathcal{P} \\ p_2 \in \mathcal{P} \\ d \in \mathcal{D}}} \{p_2 - p_1 - 1 \,|\, y_{CGdp_1} = 1, y_{CGdp_2} = 1, p_2 > p_1, \forall p_3 \in (p_1, p_2)\, y_{CGdp_3} = 0\}$$

SC2: Home classroom should be used as a default classroom for lessons of class C. The number of violations is computed this way:

$$Cost_{SC2} = \sum \{1 \,|\, l \in \mathcal{L}, f(l) = (d, p, r), r \neq homeRoom(l_C)\}$$

SC3: We denote $Cost_{SC3}$ the following expression whose result is the total number of moves between buildings:

$$\sum_{\substack{C \in \mathcal{C} \\ G \in C_G \\ d \in \mathcal{D} \\ p_1 \in \mathcal{P} \\ p_2 \in \mathcal{P} \\ r_1 \in \mathcal{R} \\ r_2 \in \mathcal{R}}} \{1 \,|\, z_{CGdp_1r_1} = 1, z_{CGdp_2r_2} = 1, p_2 > p_1, bld(r_1) \neq bld(r_2), \forall p_3 \in (p_1, p_2)\, y_{CGdp_3} = 0\}$$

C1: Some classrooms are more desired than others:

$$Cost_{C1} = \sum \{1 \,|\, l \in \mathcal{L}, f(l) = (d, p, r) \wedge r \notin l_B\}.$$

Definition 11. *An* assignment function *represents* optimized solution *if all the hard constraints from Definition 9 are satisfied and the expression*

$$w_1 \cdot Cost_{SC3} + w_2 \cdot Cost_{SC1} + w_3 \cdot Cost_{C1} + w_4 \cdot Cost_{SC2}$$

is minimalized for given weights $w_1, w_2, w_3, w_4, w_5 \in \mathbb{R}$ such that $w_4 \ll w_3 < w_2 \leq w_1$.

3. Existing approaches

Many different solving techniques and approaches have been tried in order to solve various school timetabling problems. Since school timetabling problems are NP-complete in most instances, the appropriate approaches were taken to tackle the problems, e.g. heuristic methods. The approaches include evolutionary algorithms, integer programming, constraint programming (CP) and Greedy Randomized Search Procedure (GRASP) [4] [9]. Apart from these methods there are many hybrid approaches. So far there is no outstanding approach that is better or more preferred than the others.

In the following sections we introduce approaches that are used for solving ST problems and we discuss their applicability to our problem instance. We would like to note beforehand that we were unable to find a constraint model that would be equivalent or very much alike to our problem. Most of the models in studied articles are simpler because only one building is typically used and classes are not typically divided into groups. It is unclear if the successful solving techniques used for those models are still able to solve more complex ST problems, such as our one, or not.

3.1 Sequential methods

Sequential methods [19][9] are a set of techniques based on succesive augmentation of timetables and they often simulate how the problem is solved by a human scheduler. These methods are known from early sixties and their advantage is their minimal computational requirements. A sequential methods work in two phases:

1. Lessons are sorted using a domain heuristic.
2. A timetable is created by sequentially assigning lessons to time slots following a rule so that the timetable is conflict-free at each step.

The most common heuristic for the sorting phase is called *Largest degree first* and it is just an application of *first-fail principle*[1] used in constraint programming: *assign the most restricted lesson first*. The most restricted lesson can be either the one with the largers number of constraints applied to the lesson or the one with the least number of time slots to which the lesson can be assigned. Intuitively, it is obvious the heuristic helps to find a feasible solution of a timetabling problem sooner. This is why the heuristic is still so popular in many other approaches to school timebling.

In the second phase we need a rule according to which we assign lessons to valid time slots. There are many rules usable for the purpose. For example, a lesson can be assigned to a first valid time slot or to the best time slot according to an objective function.

Sequential methods are not typically used nowadays because that approach is outperformed by more complex solving techniques. However, the heuristics, such

[1] The principle says that the variable with the fewest possible remaining alternatives is selected for instantiation.

as the presented one, are still useful. In fact they are often part of local search methods and constraint programming techniques.

3.2 Evolutionary algorithms

An evolutionary algorithm (EA) simulates evolution of biological organisms in nature. EA is a popular algorithm, or rather a framework, that was applied to many different problems and school timetabling is no exception. In case of school timetabling, the goal is to breed a good feasible timetable by the means of mutation and crossover of timetables. The general form of the algorithm is as follows:

1. Generate an initial population.
2. Evaluate each individual in the population.
3. Repeat until a termination condition is satisfied:
   (a) select parents
   (b) recombine pairs of parents (via so called *crossover operator*),
   (c) mutate the resulting offspring (via so called *mutation operator*),
   (d) evaluate new candidates (fitness function), and
   (e) select individuals for the next generation.

Crossover operators are based on the idea that when we recombine parent timetables we may get a better timetable because some building blocks are put together. However, it is not clear what are the building blocks in timetables which should a crossover operator work with and which should be recombined. Unfortunately, crossover operators do not work very well for timetabling problems in general. The impediment is that a crossover of feasible timetables may result in an unfeasible timetable. There are several possibilities how to deal with the situation, such as *genetic repair* [20] or filtering unfeasible timetables out after the application of evolutionary operators. Bufé et al. [21] mention an undesired property of genetic repair functions, i.e. there is often very low correlation between parent timetables and the offsprings. We think that these repairing techniques represent a way how to use an inappropriate crossover operator at all costs.

Papers [22] [23] present solving algorithms for ST based on EA where the authors decided deliberately not to use a crossover operator as they were unable to find out a useful crossover operator that would exert sufficient selective pressure. Consequently, all the hard work is done by mutation operator and the resulting algorithm is rather similar to a local search metaheuristic, particularly the one with restarts.

Wilke and Ostler [24] compared performance of Tabu Search, Genetic Algorithm, Simulated Annealing and Branch & Bound algorithms on a German high school. The result was that Genetic algorithm was noticeable slower in finding equivalently good solutions than the other algorithms. Abramson and Abela [25] also note that genetic algorithms require long runtimes to find a solution. Interestingly, the similar results were reported by Bajeh and Abolarinwa [26] who

evaluated performance of a genetic algorithm and tabu search on an examination timetabling problem.

We believe that EA can be successfully used to solve our problem. However, with regard to the aforementioned issues, we decided not to use this approach.

3.3 Metaheuristics

The following metaheuristics are particularly useful in case we need to optimize an existing initial feasible solution. Therefore, the following techniques represent possible further optimization steps after a primary solver is created.

3.3.1 Simulated annealing

Simulated annealing is a Monte-Carlo technique used for solving large-scale optimization problems. This probabilistic meta-heuristic was used with success on school timetabling problems by Abramson [27] [28] and Zhang et al. [29].

The viability of simulated annealing in practice was proved by the system THOR[2] [30] that employs the solving technique and that was used to create timetables on more than 100 Portuguese schools. Since the constraint model used in THOR does not match our constraint model and the software is commercial, we can only conclude that simulated annealing seems to be a promising technique for solving school timetabling problems.

3.3.2 Tabu search

Tabu search [31] is a technique that is based on local search and so called *tabu list*. The tabu list represents a short-term memory that is used to escape local optima and plateaus in a search space. The length k of the tabu list is a parameter of the algorithm. The list consists of either k solution points or k solution attributes that were stored in k previous steps of the tabu algorithm. In timetabling, a tabu list that is composed of previous k timetables is not particularly useful. A list of assignments that were made during the construction of a timetable is more commonly used.

According to the survey [5] tabu search is one of the most popular techniques for solving school timetabling problems.

A short-term memory of tabu search does not eliminate possibility of getting stuck in a plateaus of the search space. In school timetabling problems, this problem is solved by diversification strategies: random restarts [32] and adaptive relaxation. In random restart, tabu search is restarted and all information from previous tabu search is discarded. Adaptive relaxation [31] is a strategy based on dynamic adjusting of an objective function during search in order to navigate the search to new regions of the search space.

Santos et al. [33] used a tabu search algorithm that is guided by a cost function which calculates cost of hard and soft constraints violations. Initial solution is found by a constructive algorithm that rely on a heuristic approach

[2]THOR stands for *Tabelas Horárias* which can be translated as *timetabling charts*.

of placing most difficult lessons first. The solution is then improved by a tabu search algorithm employing a useful diversification strategy.

We did not try a tabu search algorithm. We wanted to concentrate our effort on creating a constructive algorithm instead. However, we expect that using tabu search on top of an constructive algorithm would eventually lead to improved solutions for our problem instance.

It is wise to take into account that the constructive algorithm and tabu search algorithm should rather be implemented in the same type of programming languages. If a declarative and an imperative languages were chosen, the constraints would have to be implemented twice. Reusability of such a solver would suffer.

3.4 Integer programming

Integer programming (IP) is not a frequently used technique to solve school timetabling problems. However, several successful attempts were reported in papers [34] and [35].

Santos et al. [35] discuss a variant of the timetabling problem which they call *Class-Teacher Timetabling Problem with Compactness Constraints* for a Brazilian school based on an IP technique called Fenchel cuts. Besides basic constraints, the presented model is designed to deal with timetable compactness, teacher preferences and distribution of lessons throughout the week. The constraint model presented in [35] does solve only a subset of constraints that are imposed on our problem since only one building is taken into account, classes are not divided into groups, etc. It is still the closest model to the formal model we present.

We did not employ an integer programming technique for several reasons. First, real-life problems are prone to be hard to express strictly by mathematical equations. Therefore, the approach may prove restrictive when we would like to extend our model further. Morever, integer programming techniques are not suitable for problems with many variables and constraints [36]. The second reason is that we do not have adequate knowledge of integer programming to solve such a robust task as is ours.

3.5 Constraint programming

Constraint programming (CP) is a programming paradigm where a problem is formulated as *constraint satisfaction problem* (CSP) by variables with finite domains and constraints over these variables. A solution to the CSP problem is an assignment of variables such that all constraints are satisfied. Chronological backtracking or its improved variants are typically used to find a solution for the constraint satisfaction problem. Constraint programming systems take advantage of a technique called *constraint propagation* that prunes variables' domains after an assignment of a value to a variable and thus the size of the search space is reduced.

Valouxis et al. [37] used constraint programming to search initial solutions of a school timetabling problem in Greece and then they applied a local search to improve solutions further.

Marte [38] used a CP model for solving a timetabling problem on a German school. The model involves dead-end driven learning and an ability to restart the CP solver. Marte demonstrates the importance of learning and restarting and their impact on the performance of the solver. We consider the model very interesting since it combines nice features of constraint programming, such as constraint propagation, with randomization of lesson placement.

A practical disadvantage of constraint programming systems based on Prolog is that they typically behave as black boxes in several aspects, such as constraint propagation which makes it hard to work with large problems. Moreover, these systems differ in support of built-in constraints considerably.

3.6 Comparison of approaches

A comparison of existing approaches for the introduced problem is difficult. There are typically several ways how to represent the formal model described in Section 2.4 in each of the mentioned approaches. Moreover, each of these representations may have a set of parameters that affect the performance of the solver considerably. Therefore, we focused our attention rather on flexibility of a resulting solver.

4. Timetabling software

In this work we examined only open-source software packages since commercial programs may provide most of the functionality we require to solve the presented problem but we would not be able to add the missing features.

It is very time consuming to fully test an open-source solver to see if it can solve the problem since it requires us to implement the problem according to the formal model in Section 2.4 or to program a converter of the problem's input to the solver's data format. The conversion phase is not as straightforward as usual since some features may not be supported by the target solver and these features need to be worked around. The time needed for the conversion may be significantly cut if the author of the solver provides support. However, open-source solvers are typically maintained by their authors in their free time and therefore support is on a best-effort basis.

4.1 Open-source timetabling software packages

We examined school timetabling solvers more or less thoroughly depending on the state of their development and on properties related to the introduced problem. We compiled a list of open-source timetabling software packages from the surveys [5] and [39] that seemed promising for our work. Only mature software packages are listed. Especially, we did not examine any software designated as (pre)alpha and beta version.

Package	**Problems**	**License**	**Prog. language**
UniTime Timetabler	University timetabling	LGPL	Java
Free Timetabling Software (FET)	University timetabling, School timetabling	GPL v2	C++
KHE	School timetabling	GPL v3	ANSI C

Table 4.1: List of open-source timetabling software

We examined closely FET and KHE software packages that we found particularly fit to our problem.

4.1.1 UniTime Timetabler

UniTime is an educational scheduling system developed by Tomáš Müller that is based on Constraint Solver Library[1] (CPSolver) written in Java. The solver uses a local search technique called *Iterative forward search algorithm* (IFS). Although UniTime is primarily used for solving large-scale university timetabling problems [40], the solver's architecture seems general enough to add the model of our problem on top of it.

The distribution of CPSolver contains implemented problems such as course timetabling, student sectioning and examination timetabling. We can see from

[1] `http://www.unitime.org/index.php?tab=1`

the implementations of these problems that it is necessary to implement many Java interfaces specifying all components of a problem to use CPSolver. So we would have to program the whole formal model in Section 2.4 in Java. It means implementing representation of our model from basic concepts such as lessons and their properties to high-level features such as evaluating timetables by a score and all the soft and hard constraints.

CPSolver is a very well programmed piece of software. We examined the CPSolver source codes and as far as we know the solver should be able to solve the introduced problem. Yet, the primary purpose of the solver is to solve university timetabling problems that differ substantially from school timetabling problems. To test CPSolver on the introduced problem would require a long-term time investment with a high risk of failure so we decided not to try the solver.

4.1.2 KHE

KHE[2] is a software platform on which a fully-fledged solvers may be built upon. The platform is an open-source project of Jeff Kingston and it is written in ANSI C. Jeff Kingston participated in the specification of XHSTT format [15] and therefore the format was naturally used as an input format for KHE.

Since XHSTT data format was designed so as to specify the common core of school timetabling problems, the format allows to specify many constraints of our problem. However, the constraint HC13 specifying moves between buildings cannot be expressed in XHSTT. The XHSTT data format was not designed with buildings in mind and, therefore, constraints useful for schools with more buildings are not supported. Moreover, it is impossible to limit the number of moves between buildings to satisfy the constraint SC3.

The fact that the other constraints of our problem can be expressed in XHSTT does not mean that it can be done in a straightforward way. The constraint HC5 can be expressed in XHSTT but it is necessary to specify all time blocks of 8 periods in length for each day in the timetable and then specify that students can study at most seven periods in each of these time blocks. Obviously, a constraint added by the mentioned type of enumeration will have a negative impact on performance.

A minor problem is that we cannot specify classrooms' capacities in XHSTT directly because XHSTT is designed so that the user specifies *resources* required by lessons and the meaning of resources is not predefined by the XHSTT specification. In order to satisfy the constraint HC6 we would need to work it around by a set of labels called *resource types* specifying capacities of classrooms not by numbers but by categories with the meanings: *room for up to 32 students*, *room for up to 16 students*, etc.

The way how KHE assigns lessons to time slots and classrooms is also important. We would need to adjust current strategy because a simple placement of lessons to classrooms without considering the buildings where the classrooms is prone to return bad timetables.

Another important aspect of a timetable for us is its compactness. XHSTT is designed to support limiting of idle periods for resources. We do not know how well the support is but it is still possible to improve the compactness of resulting

[2] `http://sydney.edu.au/engineering/it/~jeff/khe/`

timetables by specifying that we prefer lessons belonging to a partition of a class to be taught at the same times. The possibilities for improving compactness of a timetable are similar in XHSTT and in FET (see below).

We were considering extending KHE to fit our needs. Programs written in C are generally used for performance-critical applications so performance would not be an issue. However, we denied the possibility for several reasons. The code base of KHE is about 4 mega bytes large. This fact alone is quite intimidating and therefore not many people would invest their time to study an extended version of such a program. Hence, the resulting program would not be useful for broad public. It is also necessary to take into account versatility of school timetabling problems. C language, or an imperative language for that matter, makes these changes hard. This deliberation led us to employ a declarative programming language instead.

An attempt was made to contact participants of 3rd International Timetabling Competition in order to try the solvers on the introduced problem. A few participants kindly provided their solvers. However, the effort was unfruitful. Some solvers were not open-source and as such they were unusable for our purposes. The open-source solvers were built upon an older version of KHE where we dealt with some bugs.

4.1.3 FET

FET (Free Educational Timetabling)[3] is a robust open-source timetabling solver for primary schools, secondary schools and universities from Romanian programmer Liviu Lalescu. The solver is based on a local search technique called *recursive swapping*. Since FET is oriented on real-life timetabling problems, it supports a large set of constraints that are useful for various timetabling problems.

We decided to use FET to solve the introduced problem because the solver is time-tested and it supports the constraints we require. Chapter 6 describes how FET works in more detail and it introduces a solution of the presented problem.

[3] `http://www.lalescu.ro/liviu/fet/`

5. Data collection and representation

5.1 Data collection

Prior to solving a problem, it is necessary to obtain all necessary data. The school administration at SSTBTP was kind to provide data we asked for. However, the data was a set of internal school documents written for its employees who were acknowledged with various sorts of shortcuts and conventions used in the documents. The following process of data interpretation was very slow and it required many interviews with the school administration to clarify details.

A somewhat hapless consequence of this approach was the fact that we got answers for our questions, however, we did not know the proper questions beforehand. This lead to a series of late revelations about the introduced problem. Actually, we found out that there were some lessons taught every fortnight only thanks to manual inspection of timetables. This fact was not obvious from the timetables as software generating the timetables did not support fortnight lessons and, therefore, they were printed out the same way as weekly ones. So much time would be spared if there were a document describing how the school works internally.

A second task was to transform the data we received to a useful data format. The data we had was stored in a variety of data formats such as Microsoft Excel XLS format, PDF, plain text and HTML. It was necessary to pick one data format to which all relevant information would be converted.

The next section describes how the format was selected and what criteria of the selection were considered.

5.1.1 Data format selection

First, it remains to clarify what data needs to be stored for the representation of our problem. The vital data includes:

- set of periods,
- set of teachers,
- set of classes,
- set of subjects,
- set of classrooms and information about their function,
- set of study programs and their specifications,
- information teachers' availability for teaching, and
- set of constraints.

As sets were a prevalent data structure that we needed to store, we started off by storing data in an open-source relation database management system called MySQL. The database system was chosen because it allows to query and manage stored data by a powerful language called SQL[1]. Moreover, MySQL is a popular choice among software programmers because it is pluggable to many programming languages.

Despite the forgoing facts, the decision turned out to be a poor one. MySQL requires from its users to provide a schema of the database in advance. This is not an issue when data of a fully decomposed problem are to be stored. We faced a different situation, though. We were gradually getting grasp of the full extent of the problem and data representation was subject to massive changes. Finally, the MySQL was abandoned in favor of a more flexible data storage because it took too long to repeatedly change structure of the database.

It is important to realize that the problem at SSTBTP will change in future not only due to decisions of the school administration but also due to changes of school education acts. Therefore, it is reasonable to use as flexible data format as possible.

A decision was made to employ a human-readable data format with great expressibility that was easy to work with. This particularly involved possibilities such as editing data both manually and automatically. Easy convertibility to other data formats was essential as well. There were two interesting candidates. The first one was Extensible Markup Language (XML) and the second one was JavaScript Object Notation (JSON).

JSON is a lightweight data-interchange format based on name/value pairs and on ordered lists. XML is a markup language used for document exchange. XML format is more robust in comparison with JSON. The main advantages of XML over JSON are:

- It is possible to check if a XML document follows a defined XML schema and therefore if it is valid.
- There is a query language called XPath that can be used to select required parts of the XML document.
- One XML document can be converted into another one by using Extensible Stylesheet Language Transformations (XSLT) language.

In the end, we chose JSON because we did not need the additional features of XML and because the data format was designed in a way that common data structures map directly to its counterparts in programming languages. The same holds for XML but it is more complicated. The decision was rather subjective because both formats could be used. However, we found JSON to be more readable and simpler so we chose this data format. The readability may be underestimated but we dealt with a file that was 11,000 lines long and it was of utmost important to spot mistakes easily.

Note that we did not use XHSTT data format for the representation of our problem instance because it was not suitable as a primary data format in our case. XHSTT data format cannot express things, such as study programmes and

[1]Structured Query Language

many other details. Moreover, XHSTT would be inconvenient for exporting to other data formats.

We assume the reader is familiar with the JSON format in the following sections. If it is not the case, please refer to `http://www.json.org/`.

5.1.2 JSON data format

We propose a set of JSON files that specify particular parts of our problem:

- `buildings.json`,
- `classes.json`,
- `classrooms.json`,
- `macros.json`,
- `study.programs.json`,
- `subjects.json`,
- `teachers.json`, and
- `timetable.json`.

JSON format does not support comments as it is customary in many data formats by prefixing a line with special characters such as `//`, `#`, `%`, etc. We will use two slashes (`//`) to comment particular parts of our JSON format. Beware that this is not syntactically valid[2].

All the JSON listings in following sections are annotated by the meaning of particular name/value pairs. We use the mark `[Required]` to specify which name/value pairs are required, failing that leads to an invalid JSON format. The mark `[Informative]` specifies that the corresponding name/value pair is intended only for informative purposes and as such the pair is not required to be present. However, we strongly recommend to specify these informative pairs. Last mark we use is `[Optional]` and it denotes a name/value pair that need not be present and a default value is used when the pair is missing.

In the following section, we show the classroom representation in JSON. The rest of the files is described in Appendix A.

Classrooms (classrooms.json)

Classrooms are represented straightforwardly by their designations, capacities and their locations. JSON data format is shown in Fig. 5.1.

```
{
   // A unique identifier of the classroom.
   //
   // Note: The identifier is used to denote lessons that take place
   // in classroom "Mechanical workshop" in a timetable. The identifier
   // is typically a shortcut of the classroom name.
```

[2]A simple workaround is to simply add a new key-value pair to a JSON file. However, we did not use the workaround in annotated JSON samples for reader's convenience.

```
    "M": {
        // [Required] A unique integer assigned to the classroom.
        // NOTE: The identifier is useful for the solvers that work
        // with numerical variables such as SICStus Prolog.
        "id": 1,
        // [Required] The same identifier (shortcut) as the one above.
        "shortcut": "M",
        // [Informative] A name of the classroom.
        "room_name": "Mechanical workshop",
        // [Required] A capacity of the classroom specified as
        // the maximum number of students for which the classroom
        // was designed.
        "room_capacity": 16,
        // [Required] A shortcut of the building where the classroom
        // is located (see file buildings.json).
        "building_shortcut": "Panska",
        // [Informative] A note about the classroom.
        "note": ""
    },
    "PS": {
        "id": 2,
        "shortcut": "PS",
        "room_name": "Photographic studio",
        "room_capacity": 16,
        "building_shortcut": "Panska",
        "note": ""
    },
    // ...
}
```

Fig 5.1: classrooms.json

Informative name/value pairs demonstrates how JSON format is flexible in general because when we need to add more information about classrooms we simply add a new name/value pair and it does not affect a convertor that works with the JSON files.

5.2 Solver properties

Chapter 2 covers important aspects of the timetabling problem. However, there are a few open questions that need to be addressed in order to propose a solution for the timetabling problem. We will do so in the following paragraphs.

5.2.1 Fortnight lessons

Generally, a better timetable can be reached by joining two fortnight lessons to a regular one. We examined fortnight lessons at SSTBTP and we concluded that automatized joining of fortnight lessons would be troublesome and potential gain would be negligible. This is why we manually added constraints to the JSON representation of study programs, each of which links two fortnight lessons according to customs at SSTBTP. It is up to a converter how it deals with the information then.

5.2.2 Timetable compactness

We can improve compactness of the final timetable significantly by putting lessons of a class that belongs to different subgroups of one partition to the same time slots as much as possible. Failing the heuristic leads to a loss of free time slots for other lessons from other partitions or for lessons of whole class.

It may happen that a solver that is not guided by the heuristic will accomplish similarly good placement of lessons but it is improbable. The problem can be solved by adding an objective function to the solver that rewards the mentioned placement of lessons. An intelligent lesson scheduler is also an option.

6. FET solver

FET (Free Educational Timetabling) is a robust open-source timetabling solver[1] for primary schools, secondary schools and universities from Romanian programmer Liviu Lalescu.

We decided to test how difficult the introduced problem is for an existing school timetabling solver and we chose Free Educational Timetabling (FET) 5.19.0 for the purpose. The solver was successfully used to solve many other school timetabling problems. Naturally, we wanted to know how it could cope with our problem.

6.1 Introduction

FET is a cross-platform user-friendly computer software that was written in C++ and Qt Framework. The main module of the program is the *the recursive swapping algorithm implementation*. The program has two interfaces. Users typically interact with the program by a graphical user interface in order to specify their timetabling problems. However, the solver can be run from a console as a stand-alone program too.

Users typically input data to FET by its graphical user interface (GUI) and then they run the solver to find a solution for the specified task. To specify a timetabling problem in FET, a user has to provide data listed in Table 6.1.

Data	**Description**
Time grid	A time grid represented by a list of days and a list of periods.
Students	A division of students to student groups.
Subjects	All subjects that are taught at school.
Teachers	All teachers who are employed at school.
Classrooms	A list of all classrooms along with capacities of the classrooms.
Buildings	Optionally a list of buildings.
Activities	A list of lessons (called *activities* in FET).
Constraints	Constraints imposed on students, teachers, classrooms, etc.

Table 6.1: A specification of a timetabling problem in FET.

It is very tedious and time consuming to enter all the required data for a non-trivial timetabling problem to FET by hand since timetabling problems often contain hundreds of lessons and many constraints. A better approach for complicated timetabling problems is to generate input data for FET by a helper program than to use the GUI. An XML data format is used by FET to store and retrieve timetabling problems.

6.2 Algorithm

The first version of FET resulted from Lalescu's Master's Thesis [41] and it was based on a genetic algorithm. The author claims that the genetic algorithm was

[1] `http://www.lalescu.ro/liviu/fet/`

slow and that he was not able to improve the algorithm to alleviate the long runtime requirements. A local search algorithm called *recursive swapping* was incorporated instead of the genetic algorithm in 2007. The change dramatically improved solving capabilities of the system.

The recursive swapping algorithm is a recursive algorithm that starts by sorting activities in decreasing order according to how difficult it is to place the activities[2]. The activities are then recursively placed to available time slots in a timetable. If an activity A cannot be placed then all time slots are considered as a time slot for A. The time slot with the lowest number of conflicting activities (e.g. B, C, D activities) is chosen for A and the conflicting activities B, C and D are unallocated. The algorithm tries to resolve the problem with recursive placing of B, C and D. On failure the algorithm tries to place A to the time slot with the second lowest number of conflicting activities.

A more precise description of the algorithm can be found in the online documentation [42].

6.3 Data format

Unfortunately, the FET's XML data format is not documented. This is because regular users of FET do not need a specification in order to work with the program since they use the graphical user interface.

The FET's source codes are available, therefore it is possible to read how the XML data format works but it requires a certain knowledge of C/C++. The easiest way to understand the data format, we found, was to insert a constraint, an activity or any other piece of information by the graphical interface and subsequently store the result to a file for an inspection.

We present an overview of the FET's XML data format in Appendix B.

6.4 Representation of SSTBTP problem

This section outlines how we represented the introduced problem in FET and what was necessary to overcome. The actual XML code fragments are in Appendix B.

6.4.1 Time grid

We specified a time grid for SSTBTP by adding five days and twelve periods. This simple definition of the timetable is sufficient since we do not need to create timetables for both buildings separately. The periods in Panská and Štupartská are shifted only by ten minutes and the travel time constraints specified in Section 2.3.6 are not in conflict with this simple representation.

6.4.2 Students

FET works with students divided into *years*, *groups* and *subgroups*. We will refer to these terms as *FET's year*, *FET's group* and *FET's subgroup* in order to

[2]The author claims that this provides a significant speed up of the algorithm.

avoid confusion. The reader may expect a *FET's year* is just a label to denote a class with proper grade. As counterintuitive as it is *FET's years*, *FET's groups* and *FET's subgroups* are always groups of students and we can, for example, assign them a lesson or impose a constraint on them. *FET's years*, *FET's groups* and *FET's subgroups* are internally represented as one list of groups of students. Therefore, the division to years, groups and subgroups is artificial and its main purpose is to improve user experience.

To represent classes, partitions of classes and subgroups, it was necessary to match our classes and subgroups on FET's years, groups and subgroups. We will demonstrate how this can be done on an example of class 1.C with two partitions EnLang = {A1, A2} and TV = {TVB, TVG}. The straightforward representation is demonstrated in Fig. 6.1. The figure shows a natural division of students to grades, classes and subgroups.

```
<Students_List>
  <Year>
    <Name>1</Name>
    <Number_of_Students>200</Number_of_Students>
    <Group>
      <Name>1.C</Name>
      <Number_of_Students>32</Number_of_Students>
      <!-- Partition "EnLang" -->
      <Subgroup>
        <Name>1.C A1</Name>
        <Number_of_Students>16</Number_of_Students>
      </Subgroup>
      <Subgroup>
        <Name>1.C A2</Name>
        <Number_of_Students>16</Number_of_Students>
      </Subgroup>
      <!-- Partition "TV" -->
      <Subgroup>
        <Name>1.C TVB</Name>
        <Number_of_Students>16</Number_of_Students>
      </Subgroup>
      <Subgroup>
        <Name>1.C TVG</Name>
        <Number_of_Students>16</Number_of_Students>
      </Subgroup>
    </Group>
    <!-- A specification of other classes is omitted. -->
</Students_List>
```

Fig 6.1: FET's student list definition - bad solution

The problem with this representation is that FET would allow activities of all FET's subgroups A1, A2, TVB and TVH of class 1.C to take place simultaneously which is inconsistent with our model. FET supports a constraint named *A set of activities are not overlapping* that allows to specify which lessons cannot take place at the same time. However, the constraint is inefficient due to the way how it is implemented in FET. Moreover, we would need a large number of these constraints to ensure correct placement of class' lessons. We have found a more efficient solution.

Fig. 6.2 demonstrates how to implement *partitions* in FET. Firstly, the reader can see that a class is represented by FET's `<Year>` tag and subgroups are

represented by FET's `<Group>` tags. Although the code is counterintuitive, it is perfectly valid. Secondly, the code listing contains dummy FET's subgroups with names in form *SUBGROUP_A-share-SUBGROUP_B*[3]. The purpose of these dummy subgroups is to say that some FET's groups have students in common and therefore, the activities of these FET's groups cannot take place simultaneously. For example, the reader can see in Fig. 6.2 that FET's subgroup *1.C A1-share-1.C TVB* was added to forbid lessons of *1.C A1* subgroup and *1.C TVB* to take place at the same time.

The number of students in a dummy subgroup can be an arbitrary number because we do not assign to these subgroups any activities and the number is used only to check if a capacity of a classroom is not exceeded.

```
<Students_List>
  <Year>
    <Name>1.C</Name>
    <Number_of_Students>32</Number_of_Students>
    <Group>
      <Name>1.C A1</Name>
      <Number_of_Students>16</Number_of_Students>
      <Subgroup>
        <Name>1.C A1-share-1.C TVB</Name>
        <Number_of_Students>0</Number_of_Students>
      </Subgroup>
      <Subgroup>
        <Name>1.C A1-share-1.C TVG</Name>
        <Number_of_Students>0</Number_of_Students>
      </Subgroup>
    </Group>
    <Group>
      <Name>1.C TVB</Name>
      <Number_of_Students>16</Number_of_Students>
      <Subgroup>
        <Name>1.C A1-share-1.C TVB</Name>
        <Number_of_Students>0</Number_of_Students>
      </Subgroup>
      <Subgroup>
        <Name>1.C A2-share-1.C TVB</Name>
        <Number_of_Students>0</Number_of_Students>
      </Subgroup>
    </Group>
    <!-- Other groups are defined similarly. -->
  </Year>
</Students_List>
```

Fig 6.2: FET's student list definition - good solution

6.4.3 Teachers, subjects, classrooms, buildings and activities

Teachers, subjects and buildings are defined by lists that contain names of teachers or subjects or buildings, respectively.

Classrooms are represented in the similar straightforward way, and moreover, we need to specify student capacities of the classrooms and the buildings where

[3]Note that the names of the dummy subgroups are not in any way special for FET.

the classrooms are located. The constraint HC6 is thus very easy to add.

An activity is specified by a subject, a teacher, a length of the activity, and a student set. We have all the information so this is straightforward too.

6.4.4 Constraints

We present implementation details of the constraints specified in Section 2.4 in Appendix B. Those constraints are easy to add. In the following sections we discuss the constraints imposed on the introduced problem that are problematic to add.

Travelling between buildings

The constraint HC13 specifying moves between buildings cannot be currently expressed in FET. However, FET supports the constraint *Min gaps between building changes for a students set* (MGBBC) that, as the name suggests, is stronger than we require. A soft variant for the MGBBC constraint would make it easier to satisfy SC3 constraint.

The compactness of generated timetables

A problem we encountered repeatedly was how to specify FET's constraints so that a generated timetable would be compact for students. The solver was intended evidently for schools that do not allow gaps in their timetables or for universities that only try to limit the numbers of gaps. However, we were also interested where the gaps were located in timetables. For example, it is not particularly desirable for a student to have a gap after a first lesson he or she attends.

The possibilities how to improve the compactness of timetables for SSTBTP in FET were limited. Issues with the timetables' compactness went hand in hand with the inconvenience that lessons of some study groups were placed in the timetables in a way that students of the study groups started their day in afternoon. The constraints we impose on the problem do not forbid this placement of lessons. However, the placement is not desirable because students would not have time for studying after they return from school.

We have the following constraints to control an arrangement of lessons in FET:

1. Hard constraint *Max gaps per day for all students* (MGPD). The constraint limits the number of gaps for *study groups*[4]. Unfortunately, a soft version of the constraint is missing. This makes the use of constraint somewhat clumsy because it is necessary to start with a high value of permitted gaps and lower the number as long as FET is able to find a solution.

2. Hard constraint *All students begin early (max beginnings at second hour)* (ASBE). We can use the constraint to ensure that study groups attend their first lessons at first or second period. We would benefit from an existence of a constraint specifying the latest period when students can attend their

[4]See Section 2.4.

first lesson on each day. Sadly, this is not supported in the current version of FET. A soft variant of ASBE constraint is also missing.

3. Soft time constraint *Activities Preferred Time Slots* (APTS). The constraint specifies how preferred are certain time slots for placing lessons.

With the help of APTS constraint we can create *preferred time zones* as shown in Table 6.2. The idea is simply to add APTS constraints so that periods denoted by the letters A in Table 6.2 are more preferred than periods denoted by the letters B and so on.

Day / Period	1	2	3	4	5	6	7	8
Monday	A	B	C	D	E	F	G	H
Tuesday	A	B	C	D	E	F	G	H
Wednesday	A	B	C	D	E	F	G	H
Thursday	A	B	C	D	E	F	G	H
Friday	A	B	C	D	E	F	G	H

Table 6.2: Priority zones in a timetable.

6.5 Missing constraints

We examined the source codes of the solver with the intention to add the constraints we miss. We found a method that takes care of swapping of lessons. However, the method was about 6,000 lines of code long and it was not commented. Unfortunately, the method was not an exception. Moreover, there was no software documentation to substitute the lack of source code comments. That is why we gave up the idea to add the constraints we missed to FET.

7. CLP solver

This chapter presents a model of the introduced problem in constraint logic programming and we discuss properties of the model. The model is based on Marte's model [38].

7.1 Introduction

Constraint Programming [38] [43] is an approach where combinatorial problems are specified declaratively in terms of a set of variables and constraints over them. A constraint is a restriction of values that variables can be assigned concurrently. The formal definitions [44] are as follows:

Definition 12. *Constraint satisfaction problem (CSP) is specified by a triple* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *where*

- $\mathcal{X}$ *is a set of variables* $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
- $\mathcal{D}$ *is a set of finite domains* $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ *for the variables, and*
- $\mathcal{C}$ *is a set of constraints* $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ *such that the scope of each constraint is a subset of* $\mathcal{X}$.

Definition 13. *The domain of a variable is a finite set of all values that can be assigned to the variable.*

Definition 14. *A constraint* C_l *is a pair* (s_l, R_l)*, where*

- $s_l = (x_{l_1}, \ldots, x_{l_m})$ *is an* m*-tuple of variables called scope,*
- R_l *is a subset of the Cartesian product* $\prod_{x_i \in s_l} D_i$*, i.e.* R_l *is a subset of all possible variable values representing the allowed combinations of simultaneous values for the variables in* s_l.

Definition 15. *An instantiation* $\mathcal{I}$ *is a set of pairs* (x_i, a_i) *such that* $x_i \in \mathcal{X}$ *is a variable and* $a_i \in D_i$ *and each variable appears at most once in the instantiation.*

Definition 16. *Let* P *be CSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *and let* $\mathcal{I}$ *be a complete instantiation of the variables in* $\mathcal{X}$*. A value assignment is then the function defined as follows:*

$$\delta(x_i) = v \iff (x_i, v) \in \mathcal{I}$$

A solution of P *is a value assignment satisfying all the constraints in* $\mathcal{C}$.

7.2 Choice of a constraint programming system

There are many constraint programming (CP) systems such as Choco, CHIP, SWI-Prolog, SICStus Prolog, etc. We chose SICStus Prolog because we implemented a CP model that is based on the Marte's model [38] which was implemented in SICStus Prolog as well. Therefore, timetabling practitioners with an

intention to build their solvers upon the Marte's model can draw from the experience of the Marte's model and our model.

SICStus Prolog belongs to the category of constraint logic programming systems. Essentially, these systems combine Prolog programming language that is based on backtracking with a constraint programming library such as CLP(FD)[1]. The constraint library provides typically a set of well-known global constraints such as *all_different*, *global cardinality constraint*, etc.

SICStus Prolog is a commercial product. There is an alternative open-source constraint programming system called ECLiPSe[2] that should be able to run the source code we wrote for SICStus Prolog. We did not try to run the source code in ECLiPSe because the global constraint `disjoint2` we use (see below) will not be available until the release of ECLiPSe 6.1.

7.3 Global constraints

All constraint programming problems can be expressed by a set of binary constraints. However, in practice constraint programming systems provide a set of carefully crafted n-ary *global constraints* that are designed for efficient filtering of variables' domains. Thus, the global constraints can significantly improve performance of CP programs. We describe the global constraints we use in the following paragraphs.

The *disjoint2* constraint is a two dimensional disjunctive constraint that constrains the placement of a collection of rectangles $R = \{R_1, \ldots, R_n\}$ to not overlap in their areas. A rectangle R_i is defined by a 4-tuple (x_i, y_i, w_i, h_i) specifying top-left coordinates of the rectangle (variables x_i, y_i), its width w_i and its height h_i. The constraint $disjoint2(R)$ is satisfied in a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ with a value assignment function δ if and only if

$$\begin{aligned}\forall 1 \leq i < j \leq n : & \delta(x_i) + \delta(w_i) \leq \delta(x_j) \vee \\ & \delta(x_j) + \delta(w_j) \leq \delta(x_i) \vee \\ & \delta(y_i) + \delta(h_i) \leq \delta(y_j) \vee \\ & \delta(y_j) + \delta(h_j) \leq \delta(y_i).\end{aligned}$$

The *global cardinality constraint* (`gcc`) [45] over a set of variables and values states that the number of variables instantiating to a value must be between a given upper and lower bound, where the bounds can be different for each value. More precisely, if $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ with an assignment function δ, $x_1, \ldots, x_n \in \mathcal{X}$, $v_1, \ldots, v_m \in \bigcup \mathcal{D}$, $l_1, \ldots, l_m, u_1, \ldots, u_m \in \mathbb{N}$, and

$$C = \mathrm{gcc}(\{x_1, \ldots, x_n\}, \{(v_1, F_1, l_1, u_1), \ldots, (v_m, F_m, l_m, u_m)\}) \in \mathcal{C}$$

then C is satisfied if and only if

$$\forall 1 \leq j \leq m : \delta(F_i) = |\{v_i | 1 \leq i \leq n : \delta(x_i) = v_j\}|,\ l_j \leq \delta(F_i) \leq u_j.$$

The `automaton` is a global constraint in SICStus Prolog whose input is a sequence of variables and a finite-state machine. The finite-state machine checks

[1]CLP(FD) stands for Constraint Logic Programming - Finite Domain.

[2]`http://eclipseclp.org/`

a ground instance of the sequence and if the machine ends in a final state the automaton constraint holds true; otherwise it fails.

The `element` constraint is satisfied in a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ with an assignment function δ, $x, y \in \mathcal{X}$, L is an arbitrary tuple of variables from $\mathcal{X}$, and

$$C = \text{element}(x, L, y) \in \mathcal{C}$$

then C is satisfied iff the $\delta(x)$-th element of tuple L is equal to $\delta(y)$.

7.4 Constraint model

In this section we propose a constraint programming model based on Marte's CP model [38] that specifies core requirements of school timetabling problems.

We presented the formal model of the introduced problem in Section 2.4 and we will use the notation established in the section in the following text – namely the sets $\mathcal{D}, \mathcal{P}, \mathcal{C}, \mathcal{B}, \mathcal{L}$ and the functions *tAvail* and *bld*.

7.4.1 Representation of time slots

Marte proposes a straightforward representation of the time grid $G = \mathcal{D} \times \mathcal{P}$ where the pair $(0, 0)$ represents the first time slot of the first day, $(0, |\mathcal{P}| - 1)$ represents last time slot of the first day, $(1, 0)$ represents first time slot of the second day, etc. Unfortunately, variables in finite-domain constraint programming can be only integers and therefore Marte proposes mapping of pairs (i, j) to numbers $i \cdot |\mathcal{P}| + j$ which can be easily decoded back by the modulo operation and by computing integer quotients.

7.4.2 Variables and initial domains

We use function δ_0 to denote initial domains of variables. For each block lesson $l \in \mathcal{L}$ we add six finite-domain integer variables:

- a period-level starting time variables PLST(l) and PLST2(l)[3],
- a day-level starting time variable DLST(l),
- a classroom variable R(l) and R2(l),
- a building variable B(l).

We initialize the variables as follows:

- The initial domain of PLST(l) is obtained

  $$(i, j) \in \delta_0(\text{PLST}(l)) \iff \forall j \leq k < j + l_n : \ (i, k) \in G \setminus U(l)$$

  where $U(l)$ are teacher unavailabilities:

  $$U(l) = \{(d, p) \mid \forall d \in \mathcal{D}, p \in \mathcal{P} : tAvail(l_t, d, p) = 1\}.$$

  The initialization ensures that the constraint HC10 is not violated.

[3] Variable PLST2(l) is an auxiliary variable used to encode custom orders of PLST(l) domain values. Variables R2(l) and R(l) are in the same relation as PLST2(l) and PLST(l).

- To obtain the initial domain of DLST(l), we use the following projection:

$$\delta_0(\text{DLST}(l)) = \{i : (i, j) \in \delta_0(\text{PLST}(l))\}$$

- Changes in the domain of PLST(l) are propagated to the domain of DLST(l) and vice versa, by imposing the following constraint:

$$\text{DLST}(l) = \lfloor \text{PLST}(l)/|\mathcal{P}| \rfloor$$

- The domain of R(l) is initialized with the classrooms that are permitted for lesson l:

$$\delta_0(\text{R}(l)) = l_A.$$

  We assume that the classrooms l_A are encoded by integers. For the encoding we need to know the number $MaxRooms$ defined as:

$$MaxRooms = \max_{b \in \mathcal{B}} |bld^{-1}(b)|$$

  The $MaxRooms$ represents the maximum number of classrooms in the buildings. Classrooms from the first building are represented by integers $1, 2, \ldots, MaxRooms - 1$. Classrooms from the second building are represented by integers $MaxRooms, MaxRooms + 1, \ldots, 2 \cdot MaxRooms - 1$.

  The initialization ensure that the hard constraints HC6, HC7 and C1 are satisfied.

- The variable R2(l) is an auxiliary variable that enforces a custom order of the classrooms permitted for lesson l. We need a new variable and a constraint for that purpose because a finite-domain variable cannot be specified with a custom order of the domain values.

- The domain of B(l) is initialized with the buildings that are permitted for lesson l:

$$\delta_0(\text{B}(l)) = \{\lfloor r/MaxRooms \rfloor \mid r \in l_A\}$$

- To propagate changes in the domain of R(l) to the domain of B(l) and vice versa, the following constraint is imposed:

$$\text{B}(l) = \lfloor \text{R}(l)/MaxRooms \rfloor + 1$$

7.4.3 Lesson decomposition

We decompose each block lesson $l \in \mathcal{L}$ to one-period long lessons $u_1, \ldots, u_{l_n}$ such that the following variables and constraints are introduced:

- We introduce PLST(u_k) variables for $1 \le k \le l_n$ such that:

$$\delta_0(\text{PLST}(u_k)) = \{(i, j + k - 1) : (i, j) \in \delta_0(\text{PLST}(l))\}.$$

- The constraint $\mathrm{PLST}(u_1) = \mathrm{PLST}(l)$ is imposed to bound the lesson l with the first lesson of its decomposition. Moreover, the lessons $u_1, \ldots, u_{l_n}$ can only take place in succesive order. The following constraints are imposed for that purpose:

$$\mathrm{PLST}(u_{k+1}) = \mathrm{PLST}(u_k) + 1 \qquad \forall 1 \leq k < l_n.$$

We use the lesson decompositions because it allows us to work with block lessons in global cardinality constraints. For example, HC3 constraint is implemented this way. Lesson decompositions also ensures that the constraint HC12 is satisfied.

7.4.4 Students and a dual time model

The constraints HC3, SC1, HC13 and HC5 are specified for individual students. This is convenient but we cannot work with students as individuals because the problem input contains only classes' partitions and their subgroups. For this reason we define a *study group*[4] that represents a set of students who have the same timetable in common.

We implemented the basic constraint HC3 for students by adding `gcd` constraint for each study group:

$$\mathrm{gcc}(L, \{(0, E_0, 0, 1), (1, E_1, 0, 1), \ldots, (x, E_x, 0, 1)\})$$

where

- $x = |\mathcal{D}| \cdot |\mathcal{P}| - 1$ is the number of time slots in each timetable of the study group,
- L is a set of PLST variables corresponding to decomposed lessons of the study group as described in Section 7.4.3,
- variables $E_0, \ldots, E_x$ are new finite-domain variables with the domains $\{0, 1\}$.

It is hard to express constraints such as SC1 and SC3 with the help of PLST variables. That is why we use a dual model to the time model described in Section 7.4.1. The dual model describes which time slots of a study group timetable are occupied by lessons of the study group. The variables $E_0, \ldots, E_x$ are used for the purpose. Each variable E_i for $0 \leq i \leq x$ is equal to 1 if a lesson of the study group takes place at period i and 0 otherwise.

Moreover, we have a set of variables for each student group $F_0, \ldots, F_x$ so that variable F_i specify a building where the study groups is at period i:

$$F_i = \begin{cases} 0 & \text{idle period,} \\ 1 & \text{Panská, and} \\ 2 & \text{Štupartská.} \end{cases}$$

[4] See Section 2.4.

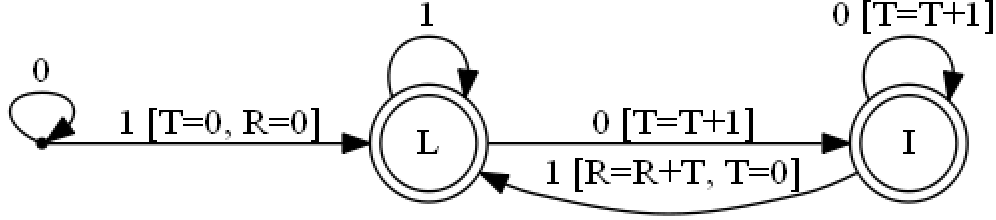


Figure 7.1: A finite-state automaton for counting gaps in a timetable for a day.

To obtain $F_0, \ldots, F_x$ variables for a study group, we impose `element` constraints and one `gcc` constraint. We decompose each lesson $l \in \mathcal{L}$ of the study group to variables $u_1, \ldots, u_{l_n}$ and we impose:

$$\text{element}(\text{PLST}(u_k), (F_0, \ldots, F_x), B(l)) \qquad \forall 1 \leq k < l_n.$$

The `gcd` constraint is used straightforwardly to enforce zeros to those variables $F_0, \ldots, F_x$ whose values were not enforced by an `element` constraint.

7.4.5 Constraints based on the dual time model

The constraints SC1, SC3 and HC13 can be easily expressed by finite-state machines. Figure 7.1 shows the automaton that counts gaps. An input of the automaton is an ordered sequence of time slots variables of a day, i.e. variables $E_s, \ldots, E_e$ where $s \in \{i \,|\, i \in \mathcal{P} : i \bmod |\mathcal{P}| = 0\}$ and $e = s + |\mathcal{P}| - 1$. The automaton uses counters T and R. The former counter is an auxiliary one and the later counter contains the actual number of gaps. The automaton is posted to CP system in form of `automaton` constraint for each study group and for each day of the study group's timetable.

The automaton shown in Figure 7.2 ensures that a study group have an idle period before a movement from Štupartská to Panská. The input of the automaton is a sequence of variables $F_s, \ldots, F_e$ where $s \in \{i \,|\, i \in \mathcal{P} : i \bmod |\mathcal{P}| = 0\}$ and $e = s + |\mathcal{P}| - 1$. The reader can notice that there is no edge between the states P and S. The missing edge guarantees that HC13 constraint is satisfied.

The automaton to counts movements between buildings is simple. It can be constructed straightforwardly, therefore, we do not present it here.

7.4.6 Bounds on daily workload

We can limit bounds of daily workload for a study group or a teacher by means of `gcc` constraint:

$$\text{gcd}(L, \{(d, F_d, X, Y) \,|\, d \in \mathcal{D}\})$$

where the X and Y are integers that represent the lower and the upper bound of daily workload for a resource, and

$$L = \{ \bigcup_{l \in L \subseteq \mathcal{L}} \{\text{DLST}(m_i) \,|\, m_1, \ldots, m_{l_n} \text{ is decomposed } l, 1 \leq i \leq l_n\}\}.$$

The approach is used to implement the maximum workload for study groups and for teachers (i.e.HC4 and HC9).

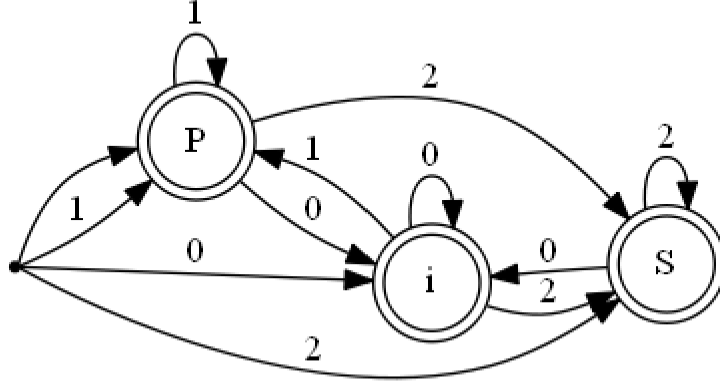


Figure 7.2: Representation of HC13 constraint by a finite-state automaton where i stands for idle period, state P stands for Panská and state S stands for Štupartská.

7.4.7 Constraints for classrooms

Double-booking of classrooms (i.e. HC2) is forbidden by a `disjoint2` global constraint in the following manner:

$$\text{disjoint2}(L \cup U)$$

where

- $L = \{(\text{PLST}(l), \text{R}(l), l_n, 1) \mid l \in \mathcal{L}\}$, and
- $U = \{(11, r, 1, 1) | r \in \mathcal{R} \wedge bld(r) = \text{Štupartská}\}$.

The set U is used to specify unavailabilities of classrooms specified in HC8.

7.5 Search procedure

We adopted Marte's search procedure described in paper [38]. The procedure is an extension of chronological backtracking that adds a restarting capability when the search hits a dead end. We explain the search procedure only briefly since a detailed description is available in Marte's paper [38].

```
1  solve(Lessons) :-
2          catch(probe(Lessons), restart, solve(Lessons)) .
3
4  probe([]).
5  probe(Lessons) :-
6          chooseLesson(Lessons, Lesson),
7          if(scheduleLesson(Lesson),
8                  (remainingLessons(Lessons, Lessons1), probe(Lessons1)),
9                  (registerDeadEnd(Lesson), fail).
```

Fig 7.1: Shortened version of Marte's search procedure

The `solve` procedure is the main method of the search procedure and it initiates the search by calling `probe` procedure that schedules lessons one by one

until all lessons are successfully scheduled or until a lesson cannot be scheduled. If `scheduleLesson` procedure fails when scheduling a lesson $l \in \mathcal{L}$, counter deadEndCount(l) for the lesson is incremented. If the counter reaches a global user-defined limit called *restart threshold* then the search is stopped, all lessons become unscheduled and program continues by initiating the search again on line 2 in Figure 7.1.

It remains to discuss how we implemented the procedures `chooseLesson` and `scheduleLesson`.

7.5.1 Heuristic for lesson selection

A time-tested heuristic for school timetabling problems is to choose the most restricted lesson to be scheduled first. We use the heuristic and we estimate how hard lesson $l \in \mathcal{L}$ is to schedule by calculating the following weight function:

$$w(l) = \frac{1}{\text{resourceDemand}(l) \cdot max(1, deadEndCount(l))}$$

where

$$\text{resourceDemand}(l) = \frac{10 \cdot \text{subgroupsNo}(l) \cdot l_n + random(1,9)}{|\text{dom}(\text{PLST}(l))| \cdot |\text{dom}(\text{R}(l))|}.$$

We know subgroup l_S for each lesson $l \in \mathcal{L}$. Subgroup l_S is an element of partition P and the function subgroupsNo(l) returns $|P|$. This means we want to schedule lessons belonging to complicated partitions sooner. The *dom* function returns current domain of a CP variable and the function $random(x, y)$ returns a pseudo-random integer between x and y.

More specifically, `chooseLesson` procedure selects the lesson that minimizes $w(l)$ from the set of lessons that are to be scheduled.

7.5.2 Heuristic for lesson scheduling

To schedule a lesson, we need to assign:

- a start time to the lesson, and
- a classroom where the lesson should take place.

As for a start time selection strategy we basically use a *first available time slot* strategy. Table 7.1 shows the order of time slots the strategy tries for each lesson. However, we use the strategy in a modified form. First, we try to assign the lesson to time slots where there are already scheduled lessons of the same class and of the same partition as the currently scheduled lesson. In case of failure we try the *first available time slot* strategy.

Classrooms are encoded with CP variables R2(l) so that the preferred classrooms l_B for lesson l are encoded by lower numbers than the rest of permitted classrooms $l_A \setminus l_B$ of the lesson l. Moreover, lesson l should respect home classroom H of class l_C. Therefore, an encoding function $f : \mathcal{R} \to \mathbb{N}^+$ is used to satisfy the following order:

$$\{f(r) \,|\, r \in l_B\} < \{f(H)\} < \{f(r) \,|\, r \in l_A \setminus l_B \setminus \{H\}\}. \tag{7.1}$$

Day / Period	1	2	3	4	5
Monday	1	6	11	16	21
Tuesday	2	7	12	17	22
Wednesday	3	8	13	18	23
Thursday	4	9	14	19	24
Friday	5	10	15	20	25

Table 7.1: The order of time slots in lesson scheduling.

The function f satisfies Equation 7.1, and moreover, classrooms are sorted so that the classrooms located in the same building as home classroom H come first. Consequently, the domain values of R2(l) can be sequentially tried and the order of classrooms is to our liking. The order of values is also likely to improve how well the constraints C1 and SC2 are satisfied.

We assign a start time to a lesson and consequently a classroom is assigned to the lesson.

7.6 Score function

To compare solutions produced by our CLP solver, we use a score function. There were many possible score functions that could be used for the introduced problem. We chose the following one that was computed from classes' timetables:

$$
\begin{aligned}
score(timetables) = & 2 \cdot \sum_{C \in \mathcal{C}} \sum_{G \in C_G} \text{\# idle periods for study group } G \\
& 2 \cdot \sum_{C \in \mathcal{C}} \sum_{G \in C_G} \text{\# buildings changes for study group } G
\end{aligned}
$$

The goal is to minimize the score. The score variable is used in the CLP solver and it is restricted in each run of the solver so that only integers from the lower half of its initial domain range are allowed. Whenever a solution is found by the CLP solver, the solver is restarted and the score variable can be assigned a value that is lower than the score of the last solution.

8. Experimental results

In this chapter we discuss the results of the introduced CLP solver and the results we obtained from FET. All experiments were run on Intel® Core™2 Quad Q9550 CPU at 2.83 GHz with 4 GB of RAM.

8.1 Data

Since timetables from previous years were not archived at SSTBTP, we made experiments on the timetabling data from the academic year 2012/2013. The data set is supplied in the attached CD in JSON format. A summary of the data set is in Table 8.1.

Classes	26
Study groups	381
Study programs	7
Block lessons	974
Teachers	94
Buildings	2
Classrooms	46
Subjects	70

Table 8.1: Summary of timetabling data at SSTBTP

The following graph shows a number of study groups in each class. The numbers are in agreement with the fact that students attend optional subjects only in 4th grade, and therefore, there are more study groups in fourth grade classes.

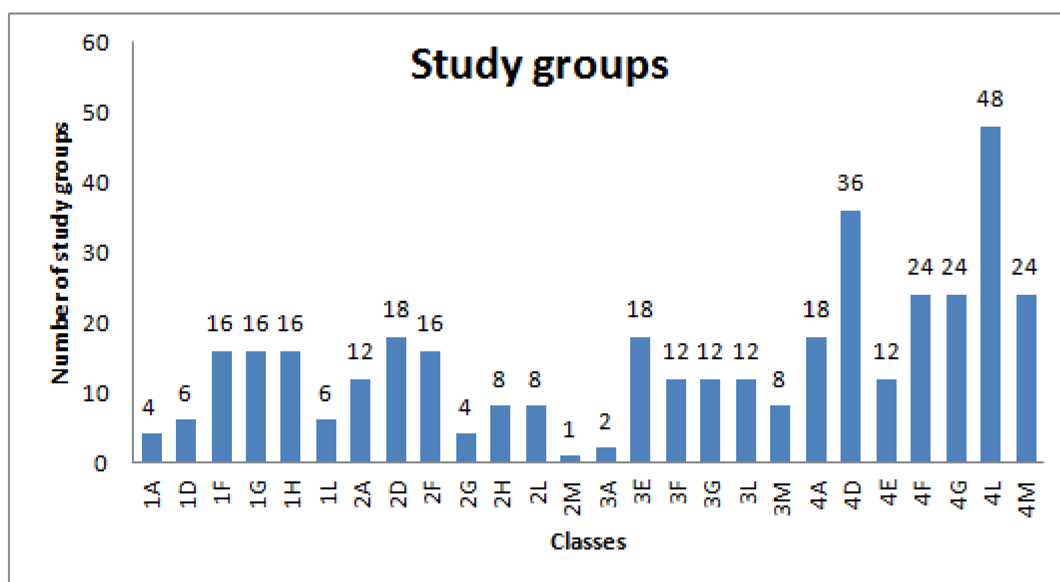


Figure 8.1: Study groups for 2012/2013 data

8.2 FET

8.2.1 Experiments

We tried FET on the data from the academic year 2012/2013. The following parameters were necessary to set:

- *Max gaps per day for all students* (MGPD) - We can set an integer that specifies the maximum of idle periods for each study group.
- *All students begin early (max beginning at second hour)* (ASBE) – We can either use the constraint or not.
- *Activities preferred time slots* (APTS) – We can set priority zones and weights of particular zones. See Table 6.2 for more details.
- *Max building changes per day* (MBCPD) – We can set an integer that specifies the maximum of movements between buildings for each study group.

We experimented with the following configurations:

Config.	MGPD	MBCPD	ASBE	APTS
1	5	3	No	No
2	5	3	No	1-20, 2-18, 3-15, 4-12, 5-9, 6-6
3	5	3	No	1-40, 2-20, 3-15, 4-12, 5-9, 6-6
4	5	3	No	1-60, 2-30, 3-15, 4-12, 5-9, 6-6
5	5	3	No	1-70, 2-40, 3-25, 4-15, 5-10, 6-5

Table 8.2: FET – configurations

In the configurations 2, 3, 4 and 5 we used priority zones so that each zone was two periods wide[1]. MGPDF and MBCPD were chosen as the worst acceptable values[2]. The syntax `x-y` in Table 8.2 means that x-th zone was added by the *Activities preferred time slots* constraint with weight y.

The experiments we conducted are listed in Table 8.3. We chose time limits for the experiments to be rather excessive. Each experiment was repeated 10 times and we list only best scores. Table 8.3 shows that priority zones may improve scores of final timetables.

We repeated the experiments in Table 8.3 with enabled ASBE constraint and we obtained no solution. This result is not surprising because when there are many study groups it is not probable that all study groups can start at first or second period of each day.

The results of the experiments were disappointing. We did not get a solution for the timetabling problem at SSTBTP. We obtained some solutions when we reduced the number of activities by searching a solution for the first three classes of the timetabling problem. However, the scores in Table 8.3 are still much worse than the ones that our CLP solver produces in up to 10 seconds.

[1] The introduced problem is specified by a time grid with 5 days and 12 periods per day. So we used 6 priority zones in total.

[2] We consulted the values with the school administration at SSTBTP.

Experiment #	Config.	Classes	Avg. time (s)	Best score
1	1	All	N/A	N/A
2	2	All	N/A	N/A
3	3	All	N/A	N/A
4	4	All	N/A	N/A
5	5	All	N/A	N/A
6	1	3	73.1	806
7	2	3	109.7	636
8	3	3	167.9	524
9	4	3	183.6	524
10	5	3	N/A	N/A

Table 8.3: FET – results

8.2.2 Conclusions

The experiments with FET shows that it was not possible to use FET to solve the introduced problem. We tested FET on a small-sized timetabling problem and we obtained some solutions for the problem. However, we did not like the timetables because lessons seemed to be placed "wherever was a free time slot" and the timetables were not as compact as we would like them to be.

8.3 CLP

In the following sections we present results of several experiments we carried out in order to show how well the CLP solver works.

8.3.1 Experiment 1 – Basic properties of the CLP solver

To see how the CLP model behaves, we wanted to know its runtime requirements and other characteristics. Since we had only one data set, we always used first n classes as an input for the CLP solver. We used first 3, 6, 9, 12, 15, 18, 21, 24 and 26 classes from the data set as the timetabling inputs. We ran the CLP solver on each of these 9 timetabling problems 10 times and we computed the arithmetic mean for the results. We use this setup in the following experiments too.

Before the CLP solver was run on a timetabling problem we always had randomly removed up to 20 lessons from the input to add variation to the tests. The CLP solver was stopped after a first solution was found.

The CLP solver was set to use the heuristic that we described in Section 7.5.2. The results of the experiment are shown in the tables 8.4, 8.5 and 8.6. Note that all the values in the tables are the average values from the ten runs of the CLP solver for each number of classes.

Table 8.4 shows the relation between the number of classes and an average time required to find a first solution of the problem. We can see that the time required to find a solution increases quite rapidly. Yet, first solutions were found in less than one hour for the hardest test cases on average. We find the times satisfactory. Figure 8.2 shows the required time with an interpolation to a power function.

Classes	Lessons	Study groups	Score	Time (s)	V.P.C.L.	L.w.P.C.
3	110.1	26	316	5.6	5.1	19.8
6	233.3	64	1009.6	28.6	15.6	42.9
9	344.3	110	1538.4	128.8	15.8	62.6
12	455.44	130	2009.11	231	21.89	83.44
15	554.1	151	2318.2	341.9	30.1	102.3
18	673.6	187	2887.8	1234.1	37.5	125.9
21	779.2	249	3319.2	1122.5	46.8	149.3
24	885.3	309	4292.2	1866	56.9	169.4
26	960.1	381	5509.8	2060	68.8	194.1

Table 8.4: Experiment 1 – V.P.C.L. stands for "Violations of preferred classrooms for lessons" and L.w.P.C. stands for Lessons with preferred classroooms".

The numbers of lessons that were not assigned to preferred classrooms are between 25 to 35 per cent. We find the result acceptable too.

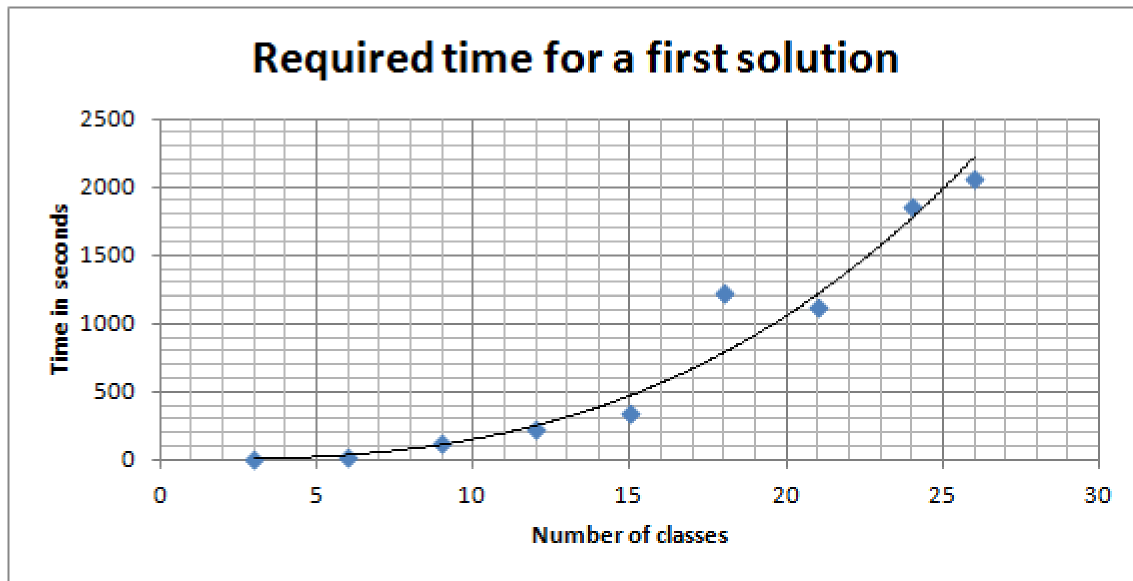


Figure 8.2: Experiment 1 – average time requirements

Table 8.5 shows average numbers of idle periods for students per week. The maximum values are not satisfactory but Experiment 1 was conducted to find first solutions, not best ones. The mean values are very good because a study group has on average about one gap per day.

8.3.2 Experiment 2 – Dead end limit

We were interested how good solutions we can expect from the CLP solver when it is run a longer time. Therefore, we let the CLP solver to search for solutions for each input until 30 dead ends were reached. The experiment was conducted five times for 3, 6, 9, 12 and 15 classes. Moreover, we removed up to 20 lessons from the input for each run as in Experiment 1.

The results are in tables 8.7, 8.8 and 8.9. These tables can be compared with the tables 8.4, 8.5 and 8.6 from the previous experiment. The *Total* and *Mean*

Classes	Total	Maximum	Mean
3	76.3	5.6	2.93
6	250.4	7.7	3.91
9	518.3	11.8	4.71
12	688.22	12.56	5.29
15	790.9	12	5.24
18	995.2	13.2	5.32
21	1120.6	13.1	4.50
24	1563.6	12.3	5.06
26	1950	12.8	5.12

Table 8.5: Experiment 1 – Average numbers of gaps per student per week. The *Total* value represents a sum of all gaps of all study groups. The *Maximum* value represents maximum number of gaps for a study group per week. The *Mean* value is an average number of gaps for a study group per week.

Classes	Total	Maximum	Mean
3	81.7	4.3	3.14
6	254.4	6.9	3.98
9	250.9	5.9	2.28
12	316.33	6.67	2.43
15	368.2	6.7	2.44
18	448.7	7.3	2.40
21	539	7.2	2.16
24	582.5	7	1.89
26	804.9	7.1	2.11

Table 8.6: Experiment 1 – Average numbers of movements between buildings per student per week. *Total* value represents a sum of all movements of all study groups. *Maximum* value represents maximum number of movements between buildings for a study group per week. *Mean* value is an average number of movements between buildings for a study group per week.

numbers of movements between buildings improved by about 8.5 per cent and *Maximum* values improved only by about 0.5 per cent. The *Total*, *Mean* and *Maximum* numbers of gaps improved by about 10 per cent.

We can conclude that even though it took much longer time to obtain the solutions, we obtained better timetables for students. Therefore, the CLP solver seems to be able to improve solutions over time.

8.3.3 Experiment 3 – Adjusted lesson scheduling strategy

We use the lesson scheduling strategy that was described in Section 7.5.2. Experiment 3 was conducted with:

- **not** modified *first available time slot strategy*,
- 30 dead ends limit, and
- 5 times for 3, 6, 9, 12 and 15 classes.

Classes	Lessons	Study groups	Score	Time (s)	V.P.C.L
3	110.14	26	259.71	224	5
6	234.4	64	921.6	701.2	13.4
9	343.6	110	1436.8	1997.2	13.4
12	455.2	130	1875.6	3282.6	22.6
15	552.71	151	2181.71	4881.86	26.57

Table 8.7: Experiment 2 – V.P.C.L stands for "Violations of preferred classrooms for lessons"

Classes	Total	Maximum	Mean
3	58.43	4.43	2.25
6	228.4	8	3.57
9	481.4	10	4.38
12	640.4	11.4	4.93
15	761.14	10.71	5.04

Table 8.8: Experiment 2 – Average numbers of gaps per student per week. The *Total* value represents a sum of all gaps of all study groups. The *Maximum* value represents maximum number of gaps for a study group per week. The *Mean* value is an average number of gaps for a study group per week.

Results of the experiment are in Table 8.10 where the reader can see that the modified strategy takes longer time and it seems to produce slightly better solutions. However, the results for 15 classes are surprising and they make the results of the experiment inconclusive.

8.3.4 Experiment 4 – Comparison with the official timetable

Table 8.11 compares the official solution of the timetabling problem from the academic year 2012/2013 with the solutions we obtained from our CLP solver. Note that the official solution contained a few errors. Some movements between buildings did not satisfy HC13 constraint. Moreover, a human scheduler of the official scheduler assigned some lessons from different partitions of a class to the same time slots. Therefore, we solved with CLP a little bit harder problem than the human scheduler.

Both solutions were obtained by restricting the score variable to be less than 5000. As the reader can see, our solutions are not better than the official solution when we compare the solutions by their scores. However, our solutions are only slightly worse than the official solution. In fact CLP 1 solution has even less gaps than the official solution. Both our solutions could be actually used by SSTBTP and it is up to the school administration to choose a solution they like.

A great advantage of the CLP solver is that it is fast in comparison with a human scheduler. The human scheduler can use the CLP solver to generate a timetable and he or she can consequently improve the timetable further. There are other scenarios in which the CLP solver can be useful. For example, the solver can be used to finish a partial timetable that the human scheduler created or the solver can be used to generate several timetables for different assignments of teachers.

Classes	Total	Maximum	Mean
3	71.43	3.86	2.75
6	232.4	6.4	3.63
9	237	6.6	2.15
12	297.4	7	2.29
15	329.71	6.57	2.18
18	446	7	2.39

Table 8.9: Experiment 2 – Average numbers of movements between buildings per student per week. *Total* value represents a sum of all movements of all study groups. *Maximum* value represents maximum number of movements between buildings for a study group per week. *Mean* value is an average number of movements between buildings for a study group per week.

	Experiment 2		**Experiment 3**		**Comparison**	
Classes	**Score**	**Time** (s)	**Score**	**Time** (s)	**Δ Score** (%)	**Δ Time** (%)
3	259.71	224	288.57	159.14	10.00	-40.76
6	921.6	701.2	1076.4	550.6	14.38	-27.35
9	1436.8	1997.2	1732	1659.75	17.04	-20.33
12	1875.6	3282.6	2146	2471.4	12.60	-32.82
15	2181.71	4881.86	1960.14	2407.29	-11.30	-102.79

Table 8.10: Comparison of *first available time slot strategy* (FATSS) and the modified FATSS in the sense of Section 7.5.2.

	Official solution	**CLP 1**	**CLP 2**
Score	4152	4748	4770
Building changes			
– **Total**	459	867	648
– **Maximum**	4	4	4
– **Mean**	1.20	2.28	1.7
Gaps			
– **Total**	1617	1507	1737
– **Maximum**	8	8	8
– **Mean**	4.24	3.96	4.56
Preferred classrooms violations	192	73	251
Time	14 days	102 minutes	138 minutes

Table 8.11: Comparison of timetables for data 2012/2013

Conclusions

Real problems are tough, the current specification of SSTBTP timetabling problem arised from questioning of school employees and examining of other school timetabling problems. The school was kind to provide study programs and the timetable for the academic year 2012/2013. Yet, it took several months to obtain all the necessary data and to understand the internal workings of the school.

We managed to represent the school timetabling problem by an existing timetabling solver called FET. However, the software provided solutions only for small instances of the presented problem. For this reason we applied a constraint logic approach (CLP) and we implemented a CLP solver.

Using our CLP solver we can produce a solution to the school timetabling problem at SSTBTP significantly faster than a human scheduler can. We can produce several solutions by the CLP solver in contrast with a human scheduler who is typically able to provide just one solution. Moreover, solutions provided by the CLP solver can be improved further by a human scheduler. The CLP solver can also use the solver to find a solution based on a partial solution.

Further work

There many improvements left to do and many ideas left to try. We see several key issues that should be further improved:

- run time requirements – We would like to decrease the runtime requirements of the CLP solver.
- new constraints – There are many schools in the Czech Republic. A comparison of their timetables and timetables generated by the CLP solver would certainly bring some ideas how to improve the CLP solver further.
- optimizations – An optimization heuristic working on top of solutions provided by the CLP solver could improve quality of final timetables.
- CP system – The current source codes of the CLP solver are run in SICStus Prolog. The Prolog is a commercial product and the fact may prevent schools from using the solver. An effort to port the solver to ECLiPSe or any other free CP solver would be beneficial.

There are many approaches that can used to solve timetabling problems. We would be particularly interested in a comparison of the constraint logic approach to a local search technique such as tabu search.

Bibliography

[1] P. Procházková, *Information system which solves scheduling*, pp. 27–30. The University of Economics, Prague, June 2011. `http://isis.vse.cz/zp/92977`.

[2] C. Valouxis, C. Gogos, P. Alefragis, and E. Housos, "Decomposing the high school timetable problem," in *Proceedings of the 9th international conference on Practice and theory of automated timetabling*, PATAT'12, pp. 209–221, SINTEF, 2012.

[3] J. Kingston, "The KTS high school timetabling system," in *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pp. 181–195, 2006.

[4] N. Pillay, "An overview of school timetabling research," in *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, pp. 321–335, 2010.

[5] N. Pillay, "A survey of school timetabling research," *Annals of Operations Research*, pp. 1–33, 2013.

[6] M. Marte, *Models and Algorithms for School Timetabling.* PhD thesis, Ludwig-Maximilians-Universität München, October 2002.

[7] R. Willemen, *School Timetable Construction: Algorithms and Complexity.* PhD thesis, Technische Universiteit Eindhoven, 2002.

[8] A. Wren, "Scheduling, timetabling and rostering - a special relationship?," in *Practice and Theory of Automated Timetabling* (E. Burke and P. Ross, eds.), vol. 1153 of *Lecture Notes in Computer Science*, pp. 46–75, Springer Berlin Heidelberg, 1996.

[9] A. Schaerf, "A survey of automated timetabling," *ARTIFICIAL INTELLIGENCE REVIEW*, vol. 13, no. 2, pp. 87–127, 1999.

[10] D. de Werra, "An introduction to timetabling," *European Journal of Operational Research*, vol. 19, no. 2, pp. 151–162, 1985.

[11] J. E. Hopcroft and R. M. Karp, "An nˆ5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[12] W. Junginger, "Timetabling in germany—a survey," *Interfaces*, vol. 16, no. 4, pp. 66–74, 1986.

[13] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, (Washington, DC, USA), pp. 184–193, IEEE Computer Society, 1975.

[14] G. Post, S. Ahmadi, S. Daskalaki, J. Kingston, J. Kyngas, C. Nurmi, D. Ranson, and H. Ruizenaar, "An xml format for benchmarks in high school timetabling. in the proceedings of the 7th international conference on the practice and theory of automated timetabling (patat 2008), montreal," 2008.

[15] J. H. Kingston, "High school timetable data format specification." `http://sydney.edu.au/engineering/it/~jeff/hseval.cgi?op=spec`. Retrieved on 2013-06-02.

[16] "Xhstt datasets." `http://www.utwente.nl/ctit/hstt/datasets/`. Retrieved on 2013-06-10.

[17] "International timetabling competition 2011." `http://www.utwente.nl/ctit/hstt/itc2011/welcome/`. Retrieved on 2013-06-11.

[18] P. De Haan, R. Landman, G. Post, and H. Ruizenaar, "A case study for timetabling in a dutch secondary school," in *Proceedings of the 6th international conference on Practice and theory of automated timetabling*, PATAT'06, (Berlin, Heidelberg), pp. 267–280, Springer-Verlag, 2007.

[19] E. K. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal of Operational Research*, vol. 140, pp. 266–280, 2002.

[20] R. M. Lewis and B. Paechter, "New crossover operators for timetabling with evolutionary algorithms," in *Proceedings of the 5th International Conference on Recent Advances in Soft Computing (RASC 2004)*, pp. 189–195, 2004.

[21] M. Bufé, T. Fischer, H. Gubbels, C. Häcker, O. Hasprich, C. Scheibel, K. Weicker, N. Weicker, M. Wenig, and C. Wolfangel, "Automated solution of a highly constrained school timetabling problem-preliminary results," in *Applications of Evolutionary Computing*, pp. 431–440, Springer, 2001.

[22] G. N. Beligiannis, C. N. Moschopoulos, G. P. Kaperonis, and S. D. Likothanassis, "Applying evolutionary computation to the school timetabling problem: The greek case," *Computers & Operations Research*, vol. 35, no. 4, pp. 1265–1280, 2008.

[23] J. Domrös and J. Homberger, "An evolutionary algorithm for high school timetabling," in *Proceedings of the 9th international conference on Practice and theory of automated timetabling*, PATAT'12, pp. 485–488, SINTEF, 2012.

[24] P. Wilke and J. Ostler, "Solving the school time tabling problem using tabu search, simulated annealing, genetic and branch & bound algorithms," in *7th international conference on the practice and theory of automated timetabling, PATAT2008*, 2008.

[25] D. Abramson and J. Abela, "A parallel genetic algorithm for solving the school timetabling problem," in *Division of Information Technology, CSIRO*, 1992.

[26] A. Bajeh, "Optimization: A comparative study of genetic and tabu search algorithms," *Optimization*, vol. 31, October 2011.

[27] D. Abramson, "Constructing school timetables using simulated annealing: sequential and parallel algorithms," *Management Science*, vol. 37, no. 1, pp. 98–113, 1991.

[28] D. Abramson, M. Krishnamoorthy, H. Dang, *et al.*, "Simulated annealing cooling schedules for the school timetabling problem," *Asia-Pacific Journal of Operational Research*, vol. 16, pp. 1–22, 1999.

[29] D. Zhang, Y. Liu, R. M'Hallah, and S. C. Leung, "A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems," *European Journal of Operational Research*, vol. 203, no. 3, pp. 550–558, 2010.

[30] F. Melício, J. P. Caldeira, and A. Rosa, "Thor: A tool for school timetabling," *Practice and Theory of Automated Timetabling VI*, p. 532, 2006.

[31] A. Schaerf, "Local search techniques for large high school timetabling problems," *systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 29, no. 4, pp. 368–377, 1999.

[32] M. J. F. Souza, L. Ochi, and N. Maculan, "A grasp-tabu search algorithm for solving school timetabling problems," *Metaheuristics: Computer Decision-Making. Kluwer Academic Publishers, Boston*, pp. 659–672, 2003.

[33] H. G. Santos, L. S. Ochi, and M. J. Souza, "A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem," *Journal of Experimental Algorithmics (JEA)*, vol. 10, pp. 2–9, 2005.

[34] S. Ribic and S. Konjicija, "A two phase integer linear programming approach to solving the school timetable problem," in *Information Technology Interfaces (ITI), 2010 32nd International Conference on*, pp. 651–656, 2010.

[35] H. G. Santos, E. Uchoa, L. S. Ochi, and N. Maculan, "Strong bounds with cut and column generation for class-teacher timetabling," *Annals of Operations Research*, vol. 194, no. 1, pp. 399–412, 2012.

[36] J. P. Newall, *Hybrid methods for automated timetabling.* PhD thesis, University of Nottingham, 1999.

[37] C. Valouxis and E. Housos, "Constraint programming approach for school timetabling," *Computers & Operations Research*, vol. 30, no. 10, pp. 1555–1572, 2003.

[38] M. Marte, "Towards constraint-based school timetabling," *Annals of Operations Research*, vol. 155, no. 1, pp. 207–225, 2007.

[39] P. Karich, "Timetabling software survey." `http://gstpl.wikispaces.com/Timetabling+Software+Survey`, Jan. 2013.

[40] T. Müller, *Constraint-based timetabling.* PhD thesis, Charles University in Prague, 2005.

[41] L. Lalescu and C. Badica, "Timetabling experiments using genetic algorithms," *Proc. TAINN*, vol. 3, pp. 221–225, 2003.

[42] L. Lalescu, "Fet - algorithm." `http://lalescu.ro/liviu/fet/doc/en/generation-algorithm-description.html`. Retrieved on 2013-05-12.

[43] M. Gavanelli and F. Rossi, "Constraint logic programming," in *A 25-year perspective on logic programming*, pp. 64–86, Springer, 2010.

[44] R. Barták, M. A. Salido, and F. Rossi, "Constraint satisfaction techniques in planning and scheduling," *Journal of Intelligent Manufacturing*, vol. 21, no. 1, pp. 5–15, 2010.

[45] J.-C. Régin, "Generalized arc consistency for global cardinality constraint," in *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 1*, pp. 209–215, AAAI Press, 1996.

[46] L. Lalescu, "Fet manual." `http://timetabling.de/manual/FET-manual.en.html`. Retrieved on 2013-07-12.

List of Tables

1.1 An example of a timetable 8

2.1 List of periods at Panská and Štupartská 15

4.1 List of open-source timetabling software 31

6.1 A specification of a timetabling problem in FET. 41
6.2 Priority zones in a timetable. 46

7.1 The order of time slots in lesson scheduling. 55

8.1 Summary of timetabling data at SSTBTP 57
8.2 FET – configurations 58
8.3 FET – results 59
8.4 Experiment 1 – V.P.C.L. stands for "Violations of preferred classrooms for lessons" and L.w.P.C. stands for Lessons with preferred classroooms". 60
8.5 Experiment 1 – Average numbers of gaps per student per week. The *Total* value represents a sum of all gaps of all study groups. The *Maximum* value represents maximum number of gaps for a study group per week. The *Mean* value is an average number of gaps for a study group per week. 61
8.6 Experiment 1 – Average numbers of movements between buildings per student per week. *Total* value represents a sum of all movements of all study groups. *Maximum* value represents maximum number of movements between buildings for a study group per week. *Mean* value is an average number of movements between buildings for a study group per week. 61
8.7 Experiment 2 – V.P.C.L stands for "Violations of preferred classrooms for lessons" 62
8.8 Experiment 2 – Average numbers of gaps per student per week. The *Total* value represents a sum of all gaps of all study groups. The *Maximum* value represents maximum number of gaps for a study group per week. The *Mean* value is an average number of gaps for a study group per week. 62
8.9 Experiment 2 – Average numbers of movements between buildings per student per week. *Total* value represents a sum of all movements of all study groups. *Maximum* value represents maximum number of movements between buildings for a study group per week. *Mean* value is an average number of movements between buildings for a study group per week. 63
8.10 Comparison of *first available time slot strategy* (FATSS) and the modified FATSS in the sense of Section 7.5.2. 63
8.11 Comparison of timetables for data 2012/2013 63
8.12 Terminology used in the thesis and in FET 85

List of Abbreviations

- *CLP* - Constraint Logic Programming
- *CLP(FD)* - Constraint Logic Programming(Finite Domain)
- *CP* - Constraint Programming
- *FET* - Free Timetabling Software
- *GRASP* - Greedy Randomized Search Procedure
- *GUI* - Graphical User Interface
- *ICT* - International Timetabling Competition
- *LP* - Logic Programming
- *PATAT* - Practice and Theory of Automated Timetabling; an international conference on timetabling
- *ST* - School Timetabling
- *STP* - School Timetabling Problem
- *ST solver* - School Timetabling solver
- *SSTBTP* - Secondary School of Telecommunication and Broadcasting Technologies in Panská 3, Prague 1
- *XHSTT* - XML High School Timetabling data format

Appendix A

The Appendix A introduces details of JSON representation of the formal model in Section 2.4.

All the JSON listings in following sections are annotated by the meaning of particular name/value pairs. We use the mark `[Required]` to specify which name/value pairs are required, failing that leads to an invalid JSON format. The mark `[Informative]` specifies that the corresponding name/value pair is intended only for informative purposes and as such the pair is not required to be present. However, we strongly recommend to specify these informative pairs. Last mark we use is `[Optional]` and it denotes a name/value pair that need not be present. A default value is used if the name/value pair is missing.

A.4 buildings.json

The following JSON representation shows how buildings are defined.

```
{
    // A unique identifier of the building.
    //
    // A classroom is assigned to a building by
    // specifying a building identifier in the
    // "building_shortcut" name/value pair in
    // classrooms.json file.
    "Panska": {
        // [Required] The same identifier as above.
        "shortcut": "Panska",
        // [Informative] An address of the building.
        "address": "Panska 3, Prague 1",
        // [Informative] A note about building.
        "note": "Main building"
    },
    "Stupartska": {
        "shortcut": "Stupartska",
        "address": "Mala Stupartska 7\/8, Prague 1",
        "note": "The second building"
    }
}
```

Fig 8.1: Description of buildings.json

A.5 classes.json

The following JSON representation defines classes and teachers' assignments. The file also defines classes' partitions and their subgroups.

```
{
   // A unique identifier of the class.
   "1.A":{
      // [Required] A grade that the class belongs to.
      "year":1,
      // [Required] A unique identifier of the class. The characters a-z, A-Z and
      // dot (.) are permitted. The value is the same as the one specified above.
```

```
"class_name":"1.A",
// [Informative] A designation of the class that does not change over time.
"invariable_class_name":"12A",
// [Required] A home classroom for class 1.A.
"home_room":"S4",
// [Required] A study program that the class follows. The value refers to
// a key in "study.programs.json" file.
"study_program_shortcut":"GST12",
// [Required] Statistics for the class.
"statistics":{
  // [Required] Total number of students in the class.
  "total":30,
  // [Informative] A number of boys in the class.
  "boysNo":30,
  // [Informative] A number of girls in the class.
  "girlsNo":0,
  // [Informative] A number of boys in the class who can
  // attend physical education.
  "exercisingBoysNo":30,
  // [Informative] A number of girls in the class who can
  // attend physical education.
  "exercisingGirlsNo":0
},
// Constraints that affect lessons of the class.
"constraints":{
  // Only one constraint is supported so far and it is
  // called "theSameStartingTime". The constraint is used
  // to specify that two or more lessons should/have to
  // be taught at the same time.
  //
  "theSameStartingTime": [
    {
      //
      "weight": 100,
      "value": [
         "F-practice:A1",
         "CH-practice:A2"
      ]
    }
  ],
  // Other constraints can follow.
},
"subjects":{
  // An identifier of a subject.
  //
  // Subjects' identifiers are specified in "subjects.json" file
  // as "shortcut" name/value pairs.
  "A":{
    // Generally lessons of a subject are not the same. Students
    // are typically taught theory and practice. Lessons differ
    // then in length, how often are the lessons repeated etc.
    "activities":{
      // Key "theory" must be specified in study.programs.json in:
      // GST12 -> subjects -> A -> activities -> year-1 -> theory.
      "theory":{
        "partition":{
          // [Required] A partition name used for division of class 1.A.
          // Note: Only characters a-z, A-Z and 0-9 are permitted.
```

```
                // Note: If lessons are for whole class then the name
                // must be "class".
                "name":"L1",
                // [Required] Subgroups that belongs to the partition L1.
                "parts":[
                  // A definition of the first subgroup.
                  {
                    // [Required] A name of the subgroup.
                    "part":"A1",
                    // [Required] A teacher assigned to the subgroup A1.
                    "teacher":{
                      "shortcut":"Bo"
                    }
                  },
                  // A definition of the second subgroup.
                  {
                    // [Required] A name of the subgroup.
                    "part":"A2",
                    // [Required] A teacher assigned to the subgroup A2.
                    "teacher":{
                      "surname":"Machacova",
                      "first_name":"Michaela",
                      "shortcut":"Mh"
                    }
                  }
                ]
              }
            }
          },
          //
          // Other activities
          //
        }
      }
    }
}
```

Fig 8.2: Description of classes.json

Partitions of classes are defined implicitly. A program processing `classes.json` file has to go through the file and for each class it has to store each partition name and it has to assign to the partition name a union of all `part` names corresponding to the partition name in the specification of the class.

This file is subject to change each year and, unfortunately, it will take some time to update the data. However, if a school has a set of rules for assigning teachers to lessons then the file can be generated automatically. The other JSON files except `teachers.json` either do not change so frequently or required changes are small.

A.6 macros.json

There are several cases when we need to specify the exact same information in JSON files on several places. For example, when a new study plan is released, it is typically very similar to the previous one. We decided to add support for macros to our JSON files to get rid of this source of errors.

Each JSON file has to be pre-processed before it is used in a convertor. In the pre-processing the strings "`@identifier`" has to be replaced by the corresponding values specified in `macros.json` file as long as there is no such sequence in the processed JSON file. Note that this property guarantees us that macros can be recursive.

```
{
    // Notice the at sign (@) character.
    // All names in name/value pairs
    // have to start with the at sign in
    // macros.json file.
    "@language-practice--preferred-rooms": {
        "type": "soft",
        "value": [
            "PJ",
            "D3",
            "SJ"
        ],
        "weight": 50
    },
    // Macros are useful for defining constants.
    "@constraint-HardConstraint": 100,
    "@constraint-VeryHighWeight": 99,
    "@constraint-HighWeight": 80
}
```

Fig 8.3: Description of macros.json

A.7 study.programs.json

The following JSON listing documents study programs.

```
{
  // A unique identifier of a study program
  "KAM12": {
     // [Informative] An official study program name
     "program": "Communication and multimedia",
     // [Required]
     "shortcut": "KAM12",
     // [Informative] An academic year when the study program
     // was used for the first time.
     "released": 2012,
     // [Informative] An arbitrary comment.
     "note": "The study plan replaces the old study program KAM05.",
     // [Required] A specification of subjects in the study program.
     "subjects": {
        // An identifier of a subject (see subjects.json file).
        "En": {
           // [Required] A specification of English lessons for all grades.
           "activities": {
              // A name/value pairs following name convention "year-<Grade>".
              "year-1": {
                 // A unique identifier of a set of block lessons for each
                 // class in the first grade that follows the study program "KAM12".
                 //
                 // Example: We want to specify 4 block lessons so that each
                 // block lesson takes 1 period, and moreover, each block
```

```
// lesson is taught every fortnight.
"practice": {
   // [Required] A number of lessons in the set. The
   // following name/value pairs specify properties of the
   // lessons in the set.
   "count": 4,
   // [Optional] A number of consecutive periods that
   // each lessons should take.
   // Note: If omitted the default value is "1".
   "blockLength": 1,
   // [Optional] An interval how often each lesson should
   // be repeated. Permitted values are "per week" and
   // "every fortnight". As the name suggests "frequency"
   // specify if each block lesson of the set is taught
   // once or if it is taught every two weeks.
   // Note: The default value is "per week".
   "frequency": "every fortnight",
   // [Optional] Constraints for all the block lessons of
   // the set.
   "constraints": {
      // [Optional] A specification of preferred classrooms.
      // All supported constraints follow the same structure
      // as the example below.
      "preferredRooms": {
         // [Required] A constraint type.
         // Two values are supported:
         // * "hard" for a hard constraint, and
         // * "soft" for a soft constraint.
         "type": "soft",
         // [Required] A weight of the soft constraint.
         // The weight is a number between 1 (the lowest weight)
         // and 100 (the highest weight). The larger the number
         // the more we want the constraint to be satisfied.
         // More precisely, we assume linear dependency.
         //
         // Note: A hard constraint has always the weight 100.
         "weight": 50,
         // [Required] A list of classrooms that are preferred.
         // The values are classrooms' identifiers (see
         // classrooms.json file).
         "value": [
              "PP"
         ]
      },
      // [Optional] A specification of classrooms that are
      // permitted for the set of lessons. The constraint
      // has to be always specified as a hard constraint.
      // If the "allowedRooms" constraint is not specified
      // all classrooms are permitted for the set of lessons.
      "allowedRooms": {
         "type": "hard",
         "weight": 100,
         // [Required] A list of classrooms' identifiers.
         "value": [
              "PP"
         ]
      }
   },
```

```
                // [Informative] An arbitrary comment.
                "note": ""
            }
        },
        "year-2": {
            "theory": {
                // ...
            }
        },
        "year-3": {
            // ...
        },
        "year-4": {
            // ...
        }
    }
    },
    // ...
  }
 },
}
```

Fig 8.4: Description of study.programs.json

A.8 subjects.json

The following JSON representation shows how subjects are defined and how constraints related to a subject can be added.

```
{
    // A unique identifier of the subject.
    "En": {
        // [Required] A name of the subject.
        "name": "English language",
        // [Required] The same shortcut as above.
        "shortcut": "En",
        // [Optional] Constraints for the subject.
        "constraints": {
            // An example of constraints with value
            // in form of a macro.
            "allowedRooms" :"@regular-rooms"
        },
        // [Informative] A note about the subject.
        "note": ""
    },
    // ...
}
```

Fig 8.5: Description of subjects.json

A.9 teachers.json

The following representation shows how teachers are represented in JSON. The most importantly the file specifies when the teachers are unavailable for teaching.

```
{
    // A unique identifier a teacher
    "An": {
        // [Required] The surname of the teacher.
        "surname": "Anisimova",
        // [Required] The first name of the teacher.
        "first_name": "Elena",
        // [Required] The same identifier as the one above.
        "shortcut": "An",
        // [Required] A specification of teacher's unavailabilities.
        "unavailabilities": [
            // A first definition of unavailability.
            {
                // [Required] An identifier of a day from timetable.json file.
                "day": "Monday",
                // [Required] Periods are numbered from one and they
                // correspond to the specification of periods in
                // timetable.json file.
                "periods": "8",
                // [Informative] A reason of the teacher's unavailability.
                "note": ""
            },
            // A second definition of unavailability.
            {
                "day": "Thursday",
                // [Required] A range of periods from the first period
                // to the ninth period.
                "periods": "1-9",
                "note": ""
            },
        ]
    },
    "Ba": {
        "surname": "Bauerova",
        "first_name": "Bohumila",
        "shortcut": "Ba",
        "unavailabilities": [
            // no unavailabilities
        ]
    },
    // ...
}
```

Fig 8.6: Description of teachers.json

A.10 timetable.json

The following JSON representation defines dimensions of a timetable and it specifies names of periods and days.

```
{
    // [Required] Specification of days
    "days" :[
        // A specification of the first day.
        // Note: When this period is referred from any other
        // file it should be referred as period zero (0)
        // because the periods are numbered from zero.
```

```
        {
            // [Required] A unique identifier of the first day.
            "name": "Monday"
        },
        // A specification of the second day.
        {
            // [Required] A unique identifier of the second day.
            "name": "Tuesday"
        },
        // ...
    ],
    "periods" :[
      // A specification of the first period.
      {
            // [Required] A designation of the period.
            "name": "7:00 - 7:45"
      },
      // A specification of the second period.
      {
            "name": "7:50 - 8:35"
      },
      // ...
    ]
}
```

Fig 8.7: Description of timetable.json

Appendix B

The Appendix B introduces details of representation of our problem in FET XML data format.

B.1 FET XML data format in a nutshell

Terminology used in FET is slightly different than the one we defined in Section 1.1. Table 8.12 shows a comparison of terms used in this work and in FET.

Thesis	**FET**
Block lesson	Activity
Period	Hour
Class	Group

Table 8.12: Terminology used in the thesis and in FET

The following code listing outlines skeleton of FET XML data format.

```
<?xml version="1.0" encoding="UTF-8"?>
<fet version="5.19.0">
  <Institution_Name><!-- Arbitrary string --></Institution_Name>
  <Comments><!-- Arbitrary string --></Comments>

  <!--
    FET works with the concept of a rectangular time grid as is usual
    in school timetabling problems. The tags <Days_List> and
    <Hours_List> specify dimensions of the time grid.
  -->

  <!-- A specification of days. -->
  <Days_List>
    <!-- <Number> tag specifies the number of days that are specified
    by <Name> tags below. The information is redundant, however, it has
    to be filled out. -->
    <Number>5</Number>
    <!-- <Name> tag specifies the name of a day. The information is
    used on many places in GUI of FET and in generated timetables. -->
    <Name>Monday</Name>
    <Name>Tuesday</Name>
    <Name>Wednesday</Name>
    <Name>Thursday</Name>
    <Name>Friday</Name>
  </Days_List>

  <!-- A specification of periods. -->
  <Hours_List>
    <!-- The same content as in <Days_List> is expected except that
    periods instead of days are specified. -->
  </Hours_List>
```

```
<!-- A specification of classes and subgroups. -->
<Students_List>
  <Year>
    <Name>1.A</Name>
    <!-- The number of students is used for comparison with
    capacities of classrooms. -->
    <Number_of_Students>30</Number_of_Students>
    <Group>
      <Name>1.A A1</Name>
      <Number_of_Students>5</Number_of_Students>
      <Subgroup>
        <Name>1.A A1-share-1.A TVH1</Name>
        <Number_of_Students>0</Number_of_Students>
      </Subgroup>
      <Subgroup>
        <Name>1.A A1-share-1.A TVH2</Name>
        <!-- The number of students can be zero. However, the tag has
        to be present. -->
        <Number_of_Students>0</Number_of_Students>
      </Subgroup>
    </Group>
  </Year>
</Students_List>

<!-- A list of teachers. -->
<Teachers_List>
  <!-- Teachers are specified only by their names. All other data
  about teachers (e.g. unavailabilities of teachers) are added by
  means of constraints. -->
  <Teacher>
    <Name>John Doe</Name>
  </Teacher>
</Teachers_List>

<!-- A list of subjects. -->
<Subjects_List>
  <!-- Subjects are specified only by their names. -->
  <Subject>
    <Name>En</Name>
  </Subject>
</Subjects_List>

<!--
Activities can be designated by tags and the tags are then used in
constraint specifications in <Time_Constraints_List> and in
<Space_Constraints_List>.
-->
<Activity_Tags_List>
  <Activity_Tag>
    <!-- An example of a tag might be "morning-lessons" that we can
```

```
    assign to each activity that is supposed to take place in the
    morning when students are rested. The tag "morning-lessons" is
    then used in a constraint specifying that preferred times for
    "morning-lessons" are, for example, three first periods of each
    day. -->
    <Name>morning-lessons</Name>
  </Activity_Tag>
</Activity_Tags_List>

<!-- A list of block lessons. -->
<Activities_List>
  <Activity>
    <!-- A teacher defined in <Teachers_List>. -->
    <Teacher>Bo</Teacher>
    <!-- A subject defined in <Subjects_List>. -->
    <Subject>En</Subject>
    <!--
      Students defined in <Students_List>. Any value of <Name> tag
      specified in <Year> or <Group>, or <Subgroup> is permitted.
    -->
    <Students>1.A A1</Students>
    <!-- A number of periods the block lesson takes. -->
    <Duration>1</Duration>
    <!--
      Activities can be divided into subactivities. <Duration> then
      means the length of subactivity and <Total_Duration> is equal
      to the sum of lengths of subactivities.

      Note: We do not use subactivities in our convertor.
      Subactivities improve user experience for regular FET users who
      work with GUI. Yet, whatever is expressed by means of
      subactivities can be expressed by regular activities as well.
     -->
    <Total_Duration>1</Total_Duration>
    <!-- A unique integer identifier assigned to each activity. -->
    <Id>1</Id>
    <!-- An integer identifier for used to associate subactivities
    with each other.
    Note: Use zero (0) for a regular activity. -->
    <Activity_Group_Id>0</Activity_Group_Id>
    <!-- The activity can be disabled without removing it from a file.
      Permitted values are "true" and "false". -->
    <Active>true</Active>
    <!-- A comment can be attached to the activity. -->
    <Comments></Comments>
  </Activity>
</Activities_List>

<!-- A specification of buildings is a list of building names. -->
<Buildings_List>
  <!-- The <Building> tag can be repeated several times. -->
```

```
  <Building>
    <!-- A value specifying name of a building -->
    <Name>Panska</Name>
  </Building>
</Buildings_List>

<!-- List of classrooms -->
<Rooms_List>
  <!-- The <Room> tag can be repeated several times. -->
  <Room>
    <!-- Designation of a classroom -->
    <Name>IT Laboratory</Name>
    <!-- The building where the classroom is located. -->
    <Building>Panska</Building>
    <!-- A capacity of the classroom. -->
    <Capacity>16</Capacity>
  </Room>
</Rooms_List>
<!-- Time constraints we place upon activities. -->
<Time_Constraints_List>
  <!-- The constraint <ConstraintBasicCompulsoryTime> is required by
  FET and it cannot be omitted. The basic time constraint ensures
  that a teacher never instructs two or more activities at the same
  time. Moreover, students are allowed to have at most one activity
  per period. -->
  <ConstraintBasicCompulsoryTime>
    <!-- Each constraint has assigned a weight from the range 0.0 -
    100.0. The larger the number the more we prefer the constraint to
    be satisfied. The value 100 is special and it means the
    constraint has to be respected.
      -->
    <Weight_Percentage>100</Weight_Percentage>
    <!-- The <Active> tag is used to enable/disable a constraint
    without the need to delete the constraint from a FET file. -->
    <Active>true</Active>
    <!-- An arbitrary comment. -->
    <Comments></Comments>
  </ConstraintBasicCompulsoryTime>
  <!-- Many other constraints can be added here. -->
</Time_Constraints_List>
<!-- A list of space constraints.

     These constraints include preferred classrooms for teachers,
     students, activities and many other types of constraints that
     affect a location.
 -->
<Space_Constraints_List>
  <!-- The constraint <ConstraintBasicCompulsorySpace> is required by
  FET and it cannot be omitted. The constraint ensures that
  classrooms are not double-booked. -->
  <ConstraintBasicCompulsorySpace>
```

```xml
      <Weight_Percentage>100</Weight_Percentage>
      <Active>true</Active>
      <Comments/>
    </ConstraintBasicCompulsorySpace>
    <!-- Many other constraints can be added here. -->
  </Space_Constraints_List>
</fet>
```

Fig 8.8: FET format skeleton

A constraint in FET is always defined with a weight. An important question is how to specify the weight. The weight is not formally explained in FET's manual. However, the following excerpt from FET's documentation[46] helps to understand how the weights affect a timetabling process:

> Every constraint has a weight. A weight of 100% means that this constraint must be respected. A lower value means it should be respected, but it is not necessary. It is difficult to explain the exact function, but a simple illustration is the following: 50% weight means that in average FET retries two times to place an activity without a conflict. If FET isn't able to place the activity without a conflict after average 2 times it keeps the conflict and tries to place the next activity.

It is clear from the excerpt that setting weights to constraints is subject to trial and error in many cases.

B.2 Students

Classes, class partitions and subgroups are specified in `<Students_List>` in the way we descibe in Section 6.4.2. A problem input contains information about the number of students in each class. However, we do not know how many students are in particular subgroups of a class. This can be easily solved because we can assume that for each partition of a class students are evenly divided to all subgroups of the partition.

B.3 Constraints

This section shows how we represent the constraints imposed on the introduced problem in FET.

B.3.1 Basic constraints

HC1

The HC1 constraint is satisfied by *Basic Compulsory Time* constraint (cf. Fig. 8.9) which is an integral part of FET solver and it cannot be even removed. In fact the *Basic Compulsory Time* constraint not only ensures that teachers do not instruct two or more lessons at the same time but it also ensures that students do not attend two or more lessons at the same time.

```
<ConstraintBasicCompulsoryTime>
    <Weight_Percentage>100</Weight_Percentage>
    <Active>true</Active>
    <Comments/>
</ConstraintBasicCompulsoryTime>
```

Fig 8.9: The Basic Compulsory Time constraint.

HC2

The HC2 constraint is satisfied by *Basic Compulsory Space* constraint (cf. Fig. 8.10). This constraint is also an integral part of FET and it cannot be removed.

```
<ConstraintBasicCompulsorySpace>
    <Weight_Percentage>100</Weight_Percentage>
    <Active>true</Active>
    <Comments/>
</ConstraintBasicCompulsorySpace>
```

Fig 8.10: The Basic Compulsory Space constraint.

HC3

The constraint HC3 was implemented by the combination of *Basic Compulsory Time* constraint and a trick with dummy FET subgroups that was discussed in Section 6.4.2.

B.3.2 Student constraints

HC4

It is easy to restrict maximum number of lessons per day for students in FET. We can just add the following constraint.

```
<ConstraintStudentsMaxHoursDaily>
      <Weight_Percentage>100</Weight_Percentage>
      <Maximum_Hours_Daily>9</Maximum_Hours_Daily>
      <Active>true</Active>
      <Comments></Comments>
</ConstraintStudentsMaxHoursDaily>
```

Fig 8.11: A specification of maximum number of lessons per day.

FET applies the constraint on all students sets specified by tags `<Year>`, `<Group>`, `<Subgroup>`. Because we specified which FET's student sets cannot be taught simultaneously in HC3, FET can compute correctly the number of lessons for each study group and therefore it can correctly restrict the number of lessons for students.

HC5

The constraint HC5 can be added very similarly to the constraint HC4 by adding:

```xml
<ConstraintStudentsMaxHoursContinuously>
	<Weight_Percentage>100</Weight_Percentage>
	<Maximum_Hours_Continuously>7</Maximum_Hours_Continuously>
	<Active>true</Active>
	<Comments></Comments>
</ConstraintStudentsMaxHoursContinuously>
```

Fig 8.12: A specification of the maximum number of lessons for students.

SC1

We described problems related to the compactness of timetables in Section 6.4.4. The syntax of the constraints mentioned in the section is as follows:

```xml
<ConstraintStudentsSetEarlyMaxBeginningsAtSecondHour>
	<!-- Weight has to be 100%. This is a restriction of FET. -->
	<Weight_Percentage>100</Weight_Percentage>
	<!-- An integer value is expected. -->
	<Max_Beginnings_At_Second_Hour>NUMBER</Max_Beginnings_At_Second_Hour>
	<Students>1.A</Students>
	<Active>true</Active>
	<Comments/>
</ConstraintStudentsSetEarlyMaxBeginningsAtSecondHour>
```

Fig 8.13: A specification of the requirement that a student set has to start each day at the first or second period defined in `<Hours_List>`.

```xml
<ConstraintStudentsSetMaxGapsPerDay>
	<!-- Weight has to be 100%. This is a restriction of FET. -->
	<Weight_Percentage>100</Weight_Percentage>
	<Max_Gaps>NUMBER</Max_Gaps>
	<Students>1.A</Students>
	<Active>true</Active>
	<Comments></Comments>
</ConstraintStudentsSetMaxGapsPerDay>
```

Fig 8.14: A maximum number of gaps for students per day.

The `ConstraintStudentsSetMaxGapsPerDay` has to be added for each class.

B.3.3 Classroom constraints

HC6

We do not need a special constraint in FET for the constraint HC6 since seating capacities are a part of classroom specifications.

```xml
<Rooms_List>
	<Room>
		<Name>S1</Name>
		<Building>Stupartska</Building>
		<Capacity>32</Capacity>
	</Room>
	<!-- Analogous entries follow. -->
```

```
<Rooms_List>
```

Fig 8.15: A specification of a classroom capacity.

HC7

Inappropriate classrooms for a lesson can be eliminated by enumerating classrooms that are preferred for the lesson. For this purpose FET implements the constraint named `ConstraintActivityPreferredRooms`.

However, the effect of the FET constraint on an activity is rather surprising. FET considers only the classrooms specified in the constraint when placing the activity even if the weight of the constraint is less than 100.

```
<ConstraintActivityPreferredRooms>
        <Weight_Percentage>100</Weight_Percentage>
        <Activity_Id>7</Activity_Id> <!-- An activity ID from
        <Activity_List> -->
        <Number_of_Preferred_Rooms>25</Number_of_Preferred_Rooms> <!--
        The number of <Preferred_Room> tags below -->
        <Preferred_Room>K</Preferred_Room>
        <Preferred_Room>1</Preferred_Room>
        <Preferred_Room>2</Preferred_Room>
        <!-- ... -->
        <Active>true</Active>
        <Comments/>
</ConstraintActivityPreferredRooms>
```

Fig 8.16: Preferred classrooms for an activity.

HC8

Availability of classrooms is implemented in FET in form of the constraint `ConstraintRoomNotAvailableTimes`. Therefore, we can simply add the constraint for each classroom when it cannot be used to satisfy the constraint HC8.

```
<ConstraintRoomNotAvailableTimes>
        <Weight_Percentage>100</Weight_Percentage>
        <!-- A classroom name specified in <Room_List> -->
        <Room>S1</Room>
        <Number_of_Not_Available_Times>5</Number_of_Not_Available_Times>
        <!-- The number of <Not_Available_Time> tags below -->
        <Not_Available_Time>
                <!-- The value has to be specified in "<Days_List>". -->
                <Day>Monday</Day>
                <!-- The value has to be specified in "<Hours_List>". -->
                <Hour>16:55 - 17:40</Hour>
        </Not_Available_Time>
        <Not_Available_Time>
                <Day>Tuesday</Day>
                <Hour>16:55 - 17:40</Hour>
        </Not_Available_Time>
        <Not_Available_Time>
                <Day>Wednesday</Day>
```

```
        <Hour>16:55 - 17:40</Hour>
    </Not_Available_Time>
    <Not_Available_Time>
        <Day>Thursday</Day>
        <Hour>16:55 - 17:40</Hour>
    </Not_Available_Time>
    <Not_Available_Time>
        <Day>Friday</Day>
        <Hour>16:55 - 17:40</Hour>
    </Not_Available_Time>
    <Active>true</Active>
    <Comments></Comments>
</ConstraintRoomNotAvailableTimes>
```

Fig 8.17: An unavailability of a classroom.

C1

The constraint ConstraintActivityPreferredRooms is generated when a constraint *preferredRooms* is specified for a lesson in a JSON input.

```
<ConstraintActivityPreferredRooms>
    <!-- A weight corresponding to the weight in the
    "preferredRooms" constraint in JSON representation. -->
    <Weight_Percentage>WEIGHT</Weight_Percentage>
    <Activity_Id>NUMBER</Activity_Id>
    <!-- A number of occurences of "<Preferred_Room>" tag below. -->
    <Number_of_Preferred_Rooms>NUMBER</Number_of_Preferred_Rooms>
    <!-- Some names of classrooms defined in "<Rooms_List>" follow.
    -->
    <Preferred_Room>PJ</Preferred_Room>
    <Preferred_Room>D3</Preferred_Room>
    <Preferred_Room>SJ</Preferred_Room>
    <!-- The constraint is either enabled or disabled. The values
    "true" and "false" are permitted. -->
    <Active>true</Active>
    <Comments></Comments>
</ConstraintActivityPreferredRooms>
```

Fig 8.18: A specification of preferred classrooms for an activity.

SC2

Home classrooms can be added by the FET's constraint ConstraintStudentsSetHomeRoom. The weight of the constraint should be low, because a higher value may lead FET to try the home classrooms too often and thus prevent FET from searching better timetables.

```
<ConstraintStudentsSetHomeRoom>
    <!-- We chose a low number so the classroom is tried by FET but
    not too often. -->
    <Weight_Percentage>20</Weight_Percentage>
    <!-- A student set defined in "<Students_List>". -->
```

```
        <Students>1.D</Students>
        <!-- A classroom name specified in <Room_List> -->
        <Room>S1</Room>
        <Active>true</Active>
        <Comments/>
</ConstraintStudentsSetHomeRoom>
```

Fig 8.19: A specification of a home classroom for a class.

B.3.4 Teacher constraints

HC9

```
<ConstraintTeachersMaxHoursDaily>
        <Weight_Percentage>100</Weight_Percentage>
        <Maximum_Hours_Daily>8</Maximum_Hours_Daily>
        <Active>true</Active>
        <Comments></Comments>
</ConstraintTeachersMaxHoursDaily>
```

Fig 8.20: Maximum teaching hours for teachers

HC10

```
<ConstraintTeacherNotAvailableTimes>
        <Weight_Percentage>100</Weight_Percentage>
        <Teacher>Bo</Teacher>
        <Number_of_Not_Available_Times></Number_of_Not_Available_Times>
                <Not_Available_Time>
                        <Day>Monday</Day>
                        <Hour>7:00 - 7:45</Hour>
                </Not_Available_Time>
        <Comments>Teacher 'Bohacova' is not available on Monday 7:00 -
        7:45</Comments>
</ConstraintTeacherNotAvailableTimes>
```

Fig 8.21: Teacher unavailability

B.3.5 Educational constraints

HC12

The constraint is imposed by specifying correct durations for each activity:

```
<Activity>
        <!-- A name specified in "<Teachers_List>". -->
        <Teacher>Bo</Teacher>
        <!-- A name specified in "<Subjects_List>". -->
        <Subject>En</Subject>
        <!-- A name specified in "<Students_List>". -->
        <Students>1.A A1</Students>
        <!-- -->
        <Duration>2</Duration>
        <Total_Duration>2</Total_Duration>
```

```
        <Id>2</Id>
        <Activity_Group_Id>0</Activity_Group_Id>
        <Active>true</Active>
        <Comments>
            Teacher 'Bohacova' teaches class 1.A (subgroup A1) English
            language. The lesson is taught in block of length 2 periods.
        </Comments>
</Activity>
```

Fig 8.22: A duration of an activity.

B.3.6 Travel time constraints

HC13

The constraint `ConstraintStudentsMinGapsBetweenBuildingChanges` is the only available choice to satisfy HC13. FET does not allow to specify a direction in which the constraint should be applied, therefore, the FET's constraint is stronger than we require.

```
<ConstraintStudentsMinGapsBetweenBuildingChanges>
        <Weight_Percentage>100</Weight_Percentage>
        <Min_Gaps_Between_Building_Changes>1</Min_Gaps_Between_Building_Changes>
        <Active>true</Active>
        <Comments></Comments>
</ConstraintStudentsMinGapsBetweenBuildingChanges>
```

Fig 8.23: An idle period for students before a movement between buildings.

SC3

We can limit the number of moves between buildings by `ConstraintStudents-MaxBuildingChangesPerDay` constraint:

```
<ConstraintStudentsMaxBuildingChangesPerDay>
        <Weight_Percentage>100</Weight_Percentage>
        <Max_Building_Changes_Per_Day></Max_Building_Changes_Per_Day>
        <Active>true</Active>
        <Comments>NUMBER</Comments>
</ConstraintStudentsMaxBuildingChangesPerDay>
```

Fig 8.24: A limit on the maximum of moves between buildings.

Appendix: CD contents

The compact disk attached to the thesis has the following structure:

- *Thesis.pdf* - An electronic version of the thesis.
- *Convertors* - The directory with PHP scripts we use for converting of JSON format presented in Appendix A to FET XML format and to a format we use in our CLP solver.
- *CLP* - The folder with SICStus Prolog source codes.
- *Experiments* - The data used in experiments and the results of the experiments.