Timetabling Based on SAT Encoding: a Case Study

Filip Marić

Faculty of Mathematics, University of Belgrade, Serbia

Abstract

In this paper we present our experience in automated course timetabling using a propositional satisfiability (SAT) encoding. Timetable requirements are represented by propositional formulae and SAT solvers are used to search for their models. Each model represents a valid timetable. We describe an appropriate SAT encoding that makes possible to formulate a very wide set of different timetable requirements. We also give some techniques used to reduce the problem size. For instance, room allocation was done in a novel and very efficient way. We present case studies of teaching timetabling for one high school and two university departments. The results obtained are encouraging and they show that this approach is sound and promising for other applications as well.

1 Introduction

Propositional satisfiability problem (SAT) is the problem of deciding if there is a truth assignment under which a given propositional formula (in conjunctive normal form) evaluates to true. It is a canonical NP-complete problem [11] and it holds a central position in the field of computational complexity. SAT problem is also important in many practical applications such as electronic design automation, software and hardware verification, artificial intelligence, and operations research. Thanks to recent advances in propositional solving technology [16,12,13], SAT solvers are becoming the tool for attacking more and more difficult practical problems. Unlike some other constraint programming tools, SAT solvers use very simple language of propositional logic which allows them to use specific data structures that make them extremely efficient.

Email address: `filip@matf.bg.ac.yu` (Filip Marić).

SAT solvers have successfully been applied to planning and scheduling problems and there have been some records of using SAT solvers in timetabling applications. However, as we are aware of, there are not many published papers that describe these applications. This paper gives detailed description of our technique that we have successfully used to build timetables for three different educational institutions.

Our strategy is to encode all timetable requirements in propositional logic and generate a CNF formula that describes all timetable constraints. We use SAT solvers to search for the models of the generated formula. Each found model represents a valid teaching timetable. If the formula is found to be unsatisfiable, some requirements have to be relaxed and the process is repeated until a solution is found.

Our experience shows that different institutions impose a range of specific teaching timetable requirements. Many of these requirements are obligatory and must be satisfied in order to have a valid timetable. Therefore, a timetabling system must be general enough and allow users to specify of very rich set of different constraints. Although the system that we have developed is not a universal timetabling tool, we will illustrate the approach we are proposing can be easily modified and adapted to fit a wide set of different requirements — wider than other commonly used approaches.

Overview of the paper. In Section §2 we give the main problem description. In Section §3 we describe a SAT encoding of the problem. In Section §4 we give a case study of timetabling for three educational institutions in Belgrade, including brief institution descriptions, description of our implementation, and the results obtained. In Section §5 we describe some related work on course timetabling. In Section §6 we describe some drawbacks of our approach and outline some ways to overcome them. In Section §7 we draw some final conclusions.

2 Problem Description

The *course timetabling* problem is to assign given lessons to given time slots while obeying some given requirements. The *room allocation* problem is to additionally assign given lecture rooms to given lessons while obeying some given requirements.

An important assumption in our approach is that timetabling is done on per-week-basis, i.e., all working weeks during the semester are considered the same. A week consists of several working days divided into a number of equal-length time-slots (we will call them *periods*) and *lessons* are required to fit into these time slots. However, lessons can have different duration, i.e., they can take for more than a single period. Each lesson is taught by one or more *teachers* in one *subject* to one or more *groups* of students. We assume that groups, teachers and all lessons for each teacher and group are known in advance [1] .

Other requirements for the timetable may vary from institution to institution. However, we identified some requirements common for many teaching institutions that we have worked with. This allows us to use the same encoding and same tool for timetabling in several different institutions. We have identified two sorts of requirements. *Correctness requirements* are essential for timetable correctness and all these requirements have to be satisfied. *Comfort requirements* represent additional wishes of the staff. Some of these are considered to be *hard* requirements that are obligatory and have to be satisfied, while some are considered to be optional, *soft* requirements and need not be satisfied if it is not possible.

Many optimization approaches to timetabling exist, and most of them introduce many soft requirements trying to find solutions that satisfy as many of them as possible. Since we use complete SAT solvers, all requirements are encoded by propositional clauses and solvers treat them as hard requirements. Therefore, our approach generates a propositional formula that encodes as much imposed requirements as possible. This ensures that every single solution found by the solver will be the solution that is good enough to be considered as the final timetable. Only if requirements are found to be unsatisfiable, some less important soft requirements are omitted. In some cases requirements can be incrementally added and better and better solutions obtained until a satisfying quality solution is found.

In the following we list the main correctness and comfort requirements that we identified.

Correctness Requirements.

- Each lesson from a predefined list of lessons has to be scheduled, and it should be scheduled exactly once in the timetable.
- A teacher cannot teach two different subjects at the same time. It is possible that a teacher is required to teach the same subject to several different groups at the same time.
- A group cannot attend two or more different lessons at the same time.
- Only one teacher can occupy one room in one given period.

[1] The literature also describes the problem of forming groups of students based on the courses that students take.

Comfort Requirements.

Forbidden and requested working hours.

- Teachers are allowed to state their forbidden hours, i.e., the hours in which they cannot give lessons. These conditions are given in several different forms (e.g., it is requested that someone cannot teach on Monday from 10am to 3pm, or somebody cannot teach on Tuesday, or someone cannot teach before 10am and after 6pm each day).
- Some teachers (e.g., visiting professors, senior professors) are allowed to explicitly state their teaching hours, i.e., the hours in which their lessons have to be scheduled (e.g., Wednesday from 1pm to 3pm and Friday from 9am to 1pm).
- Groups are also allowed to have forbidden hours (e.g., senior year students are not allowed to attend lessons in time of some research seminars).

Group and teacher overlapping.

- It can be required that some different groups do not attend lessons in the same time (e.g., if some second year students have to attend some first year courses, then second year and first years students should have non-overlapping lessons).
- Some teachers require not to give lessons in the same days as some of their colleagues (e.g., one of the two colleagues always has to be present at the lab, so they cannot teach at the same time).
- Some teachers require to give lessons in the same time (or at least in the same days) as some of their colleagues (e.g., they want to arrange meetings and joint work before or after teaching in those days).

Number of teaching days.

- Some teachers ask to have their lessons scheduled only in a given number of working days (e.g., only in two days in a week, irrelevant of what those two days actually are and how many lessons they have to teach in those two days). For instance, they want to be free to do research in other week days.
- At high schools the law requires that each full-time employee has lessons in every day of the week i.e., to teach for five days in a week.

Work day duration.

- Groups of pupils/students are allowed to have only up to N (e.g., 7) working hours a day, including idle periods. Exceptions to this rule are allowed only if explicitly required by the institution management.
- Teachers are allowed to have only N (e.g., 6) teaching hours in a day, including idle periods. Exceptions to this rule are allowed only if explicitly

required by the institution management.
- It is required that both teachers and groups have at least two lessons in a day in which they have lessons.

Idle periods.

- Teachers usually ask not to have idle periods. Still, some teachers explicitly require to have idle periods.
- Groups at high school usually should not have idle periods.
- Groups at university are usually allowed to have idle periods, but their number is restricted.
- Idle periods for groups are sometimes explicitly required (e.g., if a group is scheduled to attend many lessons in one day, it can be required to have an idle period that day for a lunch break).

Lessons that have fixed or forbidden periods.

- In high school, it is required that some more demanding subjects (e.g., Mathematics) can not be scheduled for the last period in a shift, because pupils get tired after a hard days work.
- Some subjects are required to be scheduled either for first or last period in a day (e.g., one subject is held in three consecutive lessons and is required to be scheduled in periods 1-3 or periods 5-7, in a 7 period working day).
- Some non-obligatory subjects are not attended by all students. Therefore, they should be scheduled as first or last lesson in a day, so that students who do not attend those lectures would not have idle periods (e.g., since Programming Lab is not attended by all students in a group, it is required to be first or last lesson in a day for that group). The period assigned for non-obligatory subject could be any of available periods, as long as there are no other lessons for that group before or after it.

Consecutive days.

- Some courses are taught in several lessons during a week (e.g., a course with 4 lessons in a week is required to be scheduled as 2+2, i.e., two lessons one day and two lessons another day). For pedagogical reasons, it can be required that these days are not consecutive days.
- Some teachers who do not teach every day a week want their lessons scheduled in consecutive working days. This way, if they take some days off, these days can be joined with weekend days and they can have a mini-break during the semester without interrupting their lessons and their students.

Changing shifts and buildings.

- In institutions that work in different shifts, neither pupils nor teachers are allowed to work in different shifts during the same day.

- In institutions that have different buildings that are far apart, neither teachers nor students should have lessons in different buildings the same day.

3 SAT Encoding

In this section we will describe a SAT encoding of timetabling constraints. First we will introduce the encoding of the basic timetabling problem and afterwards we will describe two different encodings of room allocation problem — a naive and more advance one.

3.1 Basic Encoding

We will assign time slots to lessons by introducing a propositional variable for each combination of a lesson and a time slot. The truth values of these basic variables determine the whole timetable. Although all constraints can be expressed directly using these basic variables, we will introduce a number of additional implied variables that ease constraint specification. We will use propositional clauses to encode various constraints. Clauses that describe relationships between all these variables will be specified along the way. These clauses are regarded to be a part of the final propositional formula that describes the timetable conditions and whose models represent the timetable. For better readability we will express some constraints by implications and equivalences, but these can trivially be converted to clauses using simple propositional tautologies.

3.1.1 Variables and Their Relationships

Working hours. The set of working days of a given institution will be denoted by days. Each working day is divided in a number of equal periods (e.g., 1 hour periods in university departments or 45 minute periods in high school). The set of periods for a day $d \in \mathsf{days}$ will be denoted by $\mathsf{periods}(d)$.

Lessons. All lessons that should be scheduled are denoted by a quadruple of the form $tsgn$ which represents the fact that the teacher t teaches the subject s for the group g for the n-th time in a week. Each lesson has its own duration which is denoted by $\mathsf{duration}(tsgn)$ and is expressed in number of periods. For example, if the teacher T has to teach the subject S to the group G two times in a week, one time for 2 periods and the other time for 3 periods, his lessons would be denoted by $TSG1$ and $TSG2$, where $\mathsf{duration}(TSG1) = 2$ and $\mathsf{duration}(TSG2) = 3$. The list of lessons for a given teacher t will be denoted by $\mathsf{lessons}(t)$. The list of lessons for a given group g will be denoted by $\mathsf{lessons}(g)$.

Basic variables. A variable x'_{tsgndp} is formed for each lesson $tsgn$. It represents the fact that the lesson $tsgn$ begins in a day d and a period p. Since the lesson can be held only during working hours of the institution, the variables x'_{tsgndp} are defined only for periods p such that $\min(\mathsf{periods}(d)) \leq p \leq \max(\mathsf{periods}(d)) - \mathsf{duration}(tsgn) + 1$. These are the only *basic variables* in the system as their values uniquely determine the values of all other variables and uniquely determine the whole timetable.

Implied variables. In order to have an easy way of specifying different timetable requirements, we define a number of *implied variables.*

A variable x_{tsgndp} is formed for each lesson $tsgn$, each working day d and each working period p. It represents the fact that the lesson $tsgn$ is given in a day d, in a period p. The following implications [2] connect the start time of a lesson with its duration:

$$x'_{tsgndp_1} \Rightarrow x_{tsgndp_2},$$

where $d \in \mathsf{days}$, $\min(\mathsf{periods}(d)) \leq p_1 \leq \max(\mathsf{periods}(d)) - \mathsf{duration}(tsgn) + 1$ and $p_1 \leq p_2 \leq p_1 + \mathsf{duration}(tsgn) - 1$.

$$x_{tsgndp_2} \Rightarrow \bigvee_{\substack{p_2 - \mathsf{duration}(tsgn) + 1 \leq p_1 \leq p_2,\\ \min(\mathsf{periods}(d)) \leq p_1 \leq \max(\mathsf{periods}(d)) - \mathsf{duration}(tsgn) + 1}} x'_{tsgndp_1},$$

where $d \in \mathsf{days}$ and $\min(\mathsf{periods}(d)) \leq p_2 \leq \max(\mathsf{periods}(d))$.

A variable x_{tsgnd} is formed for each lesson $tsgn$ and each working day d. It represents the fact that the lesson $tgsn$ is held in the day d. The following implications connect these variables with duration of a lesson:

$$x_{tsgndp} \Rightarrow x_{tsgnd},$$

where $p \in \mathsf{periods}(d)$.

$$x_{tgsnd} \Rightarrow \bigvee_{p \in \mathsf{periods}(d)} x_{tsgndp}$$

[2] Note that these implications are trivially converted to clauses using the identity $p \Rightarrow q \equiv \neg p \vee q$

Also, the following implication connects these variables with the beginning of a lesson:

$$x'_{tsgndp} \Rightarrow x_{tsgnd},$$

where $p \in \mathsf{periods}(d)$.

$$x_{tsgnd} \Rightarrow \bigvee_{\min(\mathsf{periods}(d)) \leq p \leq \max(\mathsf{periods}(d)) - \mathsf{duration}(tsgn)+1} x'_{tsgndp}$$

Note that one of the last two pairs of implications is redundant, and there is no need to add both of them to the encoding.

A variable x_{tdp} is formed for each teacher t, each working day d and each working period p. It represents the fact that the teacher t gives a lesson in a day d, in a period p. The following implications connect these variables with duration of lessons:

$$x_{tsgndp} \Rightarrow x_{tdp},$$

where $tsgn \in \mathsf{lessons}(t)$ and $\mathsf{lessons}(t)$ is the list of all lessons $tsgn$ for the teacher t,

$$x_{tdp} \Rightarrow \bigvee_{tsgn \in \mathsf{lessons}(t)} x_{tsgndp}.$$

Similarly, a variable x_{gdp} is formed for each group g, working day d and working period p. It represents the fact that the group g attends a lesson in the day d and period p. The following implications connect these variables with lesson duration:

$$x_{tsgndp} \Rightarrow x_{gdp},$$

where $tsgn \in \mathsf{lessons}(g)$ and $\mathsf{lessons}(g)$ is the list of all lessons $tsgn$ for the group g.

$$x_{gdp} \Rightarrow \bigvee_{tsgn \in \mathsf{lessons}(g)} x_{tsgndp}.$$

A variable x_{td} is formed for each teacher t and each working day d. It represents the fact that the teacher t teaches during the day d. Then, the following implications hold:

$$x_{tdp} \Rightarrow x_{td},$$

where $h \in \mathsf{periods}(d)$.

$$x_{td} \Rightarrow \bigvee_{p \in \mathsf{periods}(d)} x_{tdp}.$$

A variable x_{tp} is formed for each teacher t and each working period p. It represents the fact that a teacher t gives lessons in a period p. Then, the

$$\begin{tikzcd}
 & x'_{tsgndp} \arrow[d, two heads, tail reversed] \arrow[dr, two heads, tail reversed] & \\
 & x_{tsgndp} \arrow[r, two heads, tail reversed] \arrow[dl, two heads, tail reversed] \arrow[d, two heads, tail reversed] & x_{tsgnd} \arrow[d, two heads, tail reversed] \\
x_{gdp} & x_{tdp} \arrow[r, two heads, tail reversed] \arrow[d, two heads] & x_{td} \\
 & x_{th} &
\end{tikzcd}$$

Fig. 1. Variables and their connections

following implications hold:

$$x_{tdp} \Rightarrow x_{tp},$$

where $d \in \mathsf{days}$.

$$x_{tp} \Rightarrow \bigvee_{d \in \mathsf{days}} x_{tdp}$$

The last implication is usually omitted, because this variables are only used to forbid a certain hours for some teachers (e.g. some people do not like teaching in the morning). In this case, the use of the first implication is better understood when it is looked in its contrapositive form $\neg x_{tp} \Rightarrow \neg x_{tdp}$.

Since students are generally available each working day and each working hour, variables x_{gd} and x_{gp} are omitted, but can be used if needed.

Variables and their connections are show in Figure 1.

Idle periods. We say that a teacher (or a group) has an idle period if it does not have lessons in that period, but it has lessons before and has lessons after it. The shortest idle period is a one period, and the longest idle period occurs if someone has lessons only in the first and last period of a day.

A variable i^k_{tdp} is formed for each teacher t, day d, period p, and number k such that $1 \leq k \leq \mathsf{duration}(d) - 2$ and $\min(\mathsf{periods}(d)) + 1 \leq p \leq \max(\mathsf{periods}(d)) - k$. It represent the fact that the teacher t has idle period of length k in the day d, starting with the period p. These variables are defined by the following equivalence.

$$i^k_{tdp} \Leftrightarrow \left(x_{td(p-1)} \ \wedge \bigwedge_{0 \leq j < k} \neg x_{td(p+j)} \ \wedge \ x_{td(p+k)} \right)$$

It is converted to clausal form via

$$\begin{aligned}
&i^k_{tdp} \Rightarrow x_{td(p-1)} \\
&i^k_{tdp} \Rightarrow \neg x_{td(p+j)}, \quad 0 \leq j < k
\end{aligned}$$

$$i^k_{tdp} \Rightarrow x_{td(p+k)}$$

and
$$\left(\neg x_{td(p-1)} \;\vee \bigvee_{0\leq j<k} x_{td(p+j)} \;\vee\; \neg x_{td(p+k)}\right) \vee \neg i^k_{tdp}.$$

A variable i^k_{td} is formed for each teacher t, working day d and number k such that $1 \leq k \leq \mathsf{duration}(d) - 2$. It represents the fact that the teacher t has idle period of length k during a day d. Then, the following implications hold:

$$i^k_{tdp} \Rightarrow i^k_{td},$$

for each period p such that $\min(\mathsf{periods}(d)) + 1 \leq p \leq \max(\mathsf{periods}(d)) - k$, and
$$i^k_{td} \Rightarrow \bigvee_{\min(\mathsf{periods}(d))+1\leq p\leq\max(\mathsf{periods}(d))-k} i^k_{tdp}.$$

Also, a variable i^k_t is formed which represents the fact that a teacher t sometimes has idle period of length k. Then the following implications hold:

$$i^k_{td} \Rightarrow i^k_t,$$

for each $d \in \mathsf{days}$ and
$$i^k_t \Rightarrow \bigvee_{d\in\mathsf{days}} i^k_{td}.$$

A variable i_{tdp} is formed for each teacher t, working day d and period p such that $\min(\mathsf{periods}(d))+1 \leq p \leq \max(\mathsf{periods}(d))-1$. It represents the fact that the teacher t has an idle period in the day d starting with the period p. It is clear that for all k such that $1 \leq k \leq \max(\mathsf{periods}(d)) - p$ it holds that

$$i^k_{tdp} \Rightarrow i_{tdp}$$

and
$$i_{tdp} \Rightarrow \bigvee_{1\leq k\leq\max(\mathsf{periods}(d))-p} i^k_{tdp}.$$

All variables and constraints that have been defined in this section for teachers can be defined for all groups in the exactly same way.

3.1.2 Specification of Different Requirements

Now we will show how to encode different requirements listed in §2 using variables defined in §3.1.1. The rich set of variables allows encoding of some additional sorts of constraints that are not even listed in §2, what makes our approach very flexible. Each given timetable requirement is represented by propositional clauses and these clauses are added together with clauses that describe relationships between variables and that are introduced in Section 3.1.1, yielding the final propositional formula.

First we introduce several constructions that are going to be used to specify different requirements. Let us denote by $\mathsf{single}(\{v_1, \ldots, v_k\})$ the fact that only one of the variables v_1, ..., v_k can be true. A trivial way to define this is by using a quadratic number of clauses:

$$\mathsf{single}(\{v_1, \ldots, v_k\}) = \bigwedge_{1 \leq i < j \leq k} (\neg v_i \vee \neg v_j).$$

We generalize this concept to an arbitrary cardinality constraint. Let us denote by $\mathsf{cardinality}(\{v_1, \ldots, v_k\}) \leq m$ the fact that at most m of the variables v_1, ..., v_k can be true. An efficient way to encode cardinality constraints by propositional clauses is given in [18].

Correctness Conditions.

- Each lesson has to be scheduled. Each lesson is held in a single working day and therefore, a clause
$$\bigvee_{d \in \mathsf{days}} x_{tsgnd}$$
must hold for each lesson $tsgn$. If this holds, the condition
$$x_{tsgnd} \Rightarrow \bigvee_{\min(\mathsf{periods}(d)) \leq p \leq \max(\mathsf{periods}(d)) - \mathsf{duration}(tsgn)+1} x'_{tsgndp}$$
that we have already introduced, will ensure that all lessons are eventually going to be scheduled.
- Each lesson is scheduled exactly once in the timetable. Therefore, the beginning of each lesson is uniquely determined and the condition
$$\mathsf{single}(\{x'_{tsgndp} \mid d \in \mathsf{days},\ p \in \mathsf{periods}(d)\})$$
must hold for each lesson $tsgn$. To reduce the number of clauses this condition can be replaced by
$$\mathsf{single}(\{x_{tsgnd} \mid d \in \mathsf{days}\})$$
and for each $d \in \mathsf{days}$
$$\mathsf{single}(\{x'_{tsgndp} \mid p \in \mathsf{periods}(d)\}).$$

- Each group can attend only one lesson at a time. Therefore, for each group g, each $d \in \mathsf{days}$, and each $p \in \mathsf{periods}(d)$

  $$\mathsf{single}(\{x_{tsgndp} \mid tsgn \in \mathsf{lessons}(g)\})$$

  must hold.
- Every teacher can teach only one lesson at a time. Still, in some cases it is required that several groups (e.g., $g_1, \ldots, g_k$) are joined into a cluster of groups and that they attend the lesson tsn together, in the same period of time. This requirement is encoded with the equivalence

  $$x_{tsg_1ndp} \Leftrightarrow x_{tsg_jndp}, \quad 1 < j \leq k.$$

  If this is the case, the $\overline{\mathsf{lessons}}(t)$ will denote the list that will contain only one representative (e.g. tsg_1n) for each cluster of groups.

  Therefore, since it is required that every teacher can teach only one lesson at a time, for each teacher t, each $d \in \mathsf{days}$, and each $p \in \mathsf{periods}(d)$

  $$\mathsf{single}(\{x_{tsgndp} \mid tsgn \in \overline{\mathsf{lessons}}(t)\})$$

  must hold.[3]

Comfort Requirements.

Forbidden and requested working hours. Forbidden hours and explicit working hours for teachers are directly encoded by negation of variables x_{tdp}, x_{td}, and x_{tp}. Forbidden hours for groups are encoded using the variables x_{gdp}. These constraints are represented by single literal clauses.

Groups and teachers overlapping. The condition that two groups g_1 and g_2 are not allowed to attend lessons in the same time, is encoded by $x_{g_1dp} \Rightarrow \neg x_{g_2dp}$ and $x_{g_2dp} \Rightarrow \neg x_{g_1dp}$, for each day d and period p. Overlapping of teachers' teaching hours is encoded in a similar way.

Number of teaching days. The condition that a teacher t teaches for exactly n days in a week is encoded by

$$\mathsf{cardinality}(\{x_{td} \mid d \in \mathsf{days}\}) \leq n \ \wedge \ \mathsf{cardinality}(\{\neg x_{td} \mid d \in \mathsf{days}\}) \leq |\mathsf{days}| - n.$$

Work day duration. Duration of a working day for student groups is encoded using variables l^k_{gd} which are formed for each group g, day d, and number

[3] A similar technique could be used to encode the requirement that several teachers teach to the same students in the same time.

$k \leq |\mathsf{periods}(d)|$. The variable l^k_{gd} represents the fact that teaching time for a group g spans for at least k periods (including idle periods) in a day d. It is defined by:

$$x_{gdp} \wedge x_{gd(p+k-1)} \Rightarrow l^k_{gd},$$

for all p such that $\min(\mathsf{periods}(d)) \leq p \leq \max(\mathsf{periods}(d)) - k + 1$, and

$$l^k_{gd} \Rightarrow \bigvee_{\min(\mathsf{periods}(d)) \leq p \leq \max(\mathsf{periods}(d)) - k + 1} (x_{gdp} \wedge x_{gd(p+k-1)}).$$

The requirement that a work day duration for a group is limited to n periods, is encoded by single literal constraints $\neg l^k_{gd}$, for each $k > n$.

The requirement that a work day duration for a group is at least n periods is encoded by the constraint $x_{gd} \Rightarrow l^n_{gd}$.

Work day duration for teachers is encoded in a similar way.

Idle periods. The requirement that idle periods of length k are not allowed for the teacher t is specified by single literal constraint $\neg i^k_t$.

The requirement that a teacher t is not allowed to have more than one idle period per day the condition is specified by

$$\mathsf{single}(\{i_{tdp} \mid \min(\mathsf{periods}(d)) + 1 \leq p \leq \max(\mathsf{hours}(d) - 1)\}).$$

The requirement that a teacher t is allowed to have at most n idle periods per week is specified by a cardinality constraint

$$\mathsf{cardinality}(\{i_{tdp} \mid d \in \mathsf{days},\ p \in \mathsf{periods}(d)\}) \leq n.$$

Other sorts of idle period constraints and idle period constraints for groups are defined in a similar way.

Lessons that have fixed or forbidden periods.

The requirement that a lesson can begin only in period p_1, p_2, ..., or p_n is encoded by

$$x_{tsgnd} \Rightarrow x'_{tsgndp_1} \vee \ldots \vee x'_{tsgndp_n}.$$

The requirement that a lesson $tsgn$ must be the first or last lesson for the group g in a day d, is encoded by

$$x'_{tsgndp} \Rightarrow \left(\bigwedge_{\min(\mathsf{periods}(d)) \leq p' < p} \neg x_{gdp'} \right) \vee \left(\bigwedge_{p + \mathsf{duration}(tsgn) \leq p' \leq \max(\mathsf{periods}(d))} \neg x_{gdp'} \right).$$

Consecutive days If some lessons for the same course should not occur in consecutive days, the conditions

$$x_{tsgnd} \Rightarrow \neg x_{tsg(n+1)(d+1)}$$

must hold for each day d except the last working day in a week.

Changing shifts or buildings Timetabling of different shifts and different buildings is done separately, one at a time. This means that a timetable for one shift (or building) is created first, and when the timetable for the other shift (or building) is created, the constraints related to changing shifts (or buildings) are represented as forbidden day constraints.

3.1.3 Complexity of Encoding

The set of variables is clearly dominated by the variables x'_{tgsndp} and x_{tgsndp}. Therefore, its size is $O(n_l \cdot n_d \cdot n_p)$, where n_l is the total number of lessons that have to be scheduled, n_d is the number of working days, and n_p is the number of periods in a day.

Different single conditions are encoded by quadratic sets of clauses, so they dominate the total clause set size. The requirement that each lesson is scheduled only once in the timetable introduces $O(n_l \cdot (n_d^2 + n_d \cdot n_p^2))$ clauses. The requirement that each group can attend only one lesson at a time introduces $O(n_d \cdot n_p \cdot n_g \cdot {n_{gl}}^2)$ clauses, where n_g is the number of groups, and n_{gl} is the number of lessons for a group. Similarly, the requirement that each teacher can teach only one lesson at a time introduces $O(n_d \cdot n_p \cdot n_t \cdot {n_{tl}}^2)$ clauses, where n_t is the number of teachers, and n_{tl} is the number of lessons for a teacher.

3.2 Naive Room Allocation

In this and the following section we will describe a way to solve the more complex problem of timetabling with room allocation. It requires to assign not only time slots but also rooms in which lessons are held.

Basic variables. One way to encode the room allocation conditions is to extend the basic variables, so that they also encode the room in which the lesson is given. Since it is assumed that rooms are not changed during the lesson duration, the room in which the lesson is given is uniquely determined by the room in which the lesson begins. Then, for each lesson $tsgn$, day d, period p, and a room r, a basic variable $x'_{tsgndpr}$ is formed, and it represents the fact that the lesson $tsgn$ begins in the day d, the period p and is held in the room r. Since, it is usually assumed that rooms are not equivalent, i.e., that there are some lessons that can be scheduled only in some specific rooms, some of the variables that have just been introduced can be omitted. For each lesson $tsgn$, the set $\mathsf{rooms}(tsgn)$ will denote the set of rooms in which the lesson $tsgn$ could be scheduled. Then, a variable $x'_{tsgndpr}$ should be introduced only for those rooms r which are in $\mathsf{rooms}(tsgn)$.

Implied variables. An implied variable $x_{tsgndpr}$ is also formed, for each lesson $tsgn$, day d, period p, and room r that is in $\mathsf{rooms}(tsgn)$. It represents the fact that the lesson $tsgn$ is held in day d, period p in the room r. Since it is required that room is not changed during the lesson duration, the following implications hold:

$$x'_{tsgndp_1r} \Rightarrow x_{tsgndp_2r},$$

where $d \in \mathsf{days}$, $\min(\mathsf{periods}(d)) \leq p_1 \leq \max(\mathsf{periods}(d)) - \mathsf{duration}(tsgn) + 1$ and $p_1 \leq p_2 \leq p_1 + \mathsf{duration}(tsgn) - 1$, and

$$x_{tsgndp_2r} \Rightarrow \bigvee_{\substack{p_2 - \mathsf{duration}(tsgn) + 1 \leq p_1 \leq p_2,\\ \min(\mathsf{periods}(d)) \leq p_1 \leq \max(\mathsf{periods}(d)) - \mathsf{duration}(tsgn) + 1}} x'_{tsgndp_1r},$$

where $d \in \mathsf{days}$ and $\min(\mathsf{periods}(d)) \leq p_2 \leq \max(\mathsf{periods}(d))$.

For each teacher t, day d, period p, and a room r, a variable x_{tdpr} is formed. It represents the fact that teacher t occupies the room r in the period p of day d.

All variables except $x'_{tsgndpr}$ become implied variables and the following implications hold:

$$x'_{tsgndpr} \Rightarrow x'_{tsgndp}, \qquad x'_{tsgndp} \Rightarrow \bigvee_{r \in \mathsf{rooms}(tsgn)} x'_{tsgndpr},$$

$$x_{tsgndpr} \Rightarrow x_{tsgndp}, \qquad x_{tsgndp} \Rightarrow \bigvee_{r \in \mathsf{rooms}(tsgn)} x_{tsgndpr},$$

$$x_{tdpr} \Rightarrow x_{tdp}, \qquad x_{tdp} \Rightarrow \bigvee_{r \in \mathsf{rooms}} x_{tdpr}.$$

The variables and their relationships are shown in Figure 2.

Correctness conditions. For each day d, period p, and room r, correctness is encoded by

$$\mathsf{single}\{x_{tdpr} \mid t \in \mathsf{teachers}\}.$$

$$\begin{tikzcd}
x'_{tsgndpr} \arrow[rr, <->>] \arrow[d, <->>] & & x'_{tsgndp} \arrow[d, <->>] \arrow[dr, <->>] & \\
x_{tsgndpr} \arrow[rr, <->>] \arrow[d, <->>] & & x_{tsgndp} \arrow[r, <->>] \arrow[d, <->>] \arrow[dl, <->>] & x_{tsgnd} \arrow[d, <->>] \\
x_{tdpr} \arrow[rr, <->>, bend right] & x_{gdp} & x_{tdp} \arrow[r, <->>] \arrow[d, ->>] & x_{td} \\
& & x_{th} &
\end{tikzcd}$$

Fig. 2. Variables and their connections - room allocation

Since one room is sufficient for a lesson to be held, for each day d, period p, and room r, the following condition holds:

$$\mathsf{single}\{x_{tdpr} \mid r \in \mathsf{rooms}\}.$$

3.2.1 *Complexity of Encoding*

When different rooms are introduced, the set of variables becomes dominated by the variables $x'_{tsgndpr}$ and $x_{tsgndpr}$, and their number is $O(n_l \cdot n_d \cdot n_p \cdot n_r)$, where n_l is the total number of lessons that have to be scheduled, n_d is the number of working days, n_p is the number of periods in a day, and n_r is the number of rooms.

3.3 *Cardinality Based Room Allocation*

In this section we introduce a novel way of room allocation, based on using cardinality constraints. Cardinality based room allocation works in two phases. In the first phase, only the periods are assigned to lessons, and in the second phase, the rooms are allocated. However, additional requirements for room allocation have to be added during the first phase in order to make the second phase possible.

Let us first consider the simplest case of room allocation problem. Assume that the institution has n_r rooms, and all rooms are considered to be equal. Then, the first phase has to ensure that for each period at most n_r teachers are scheduled. This is encoded by conditions:

$$\mathsf{cardinality}(\{x_{tdp} \mid t \in \mathsf{teachers}\}) \leq n_r,$$

for each day d and period p.

Now consider the case where some lessons can only be held in a specific kind of rooms. For example, programming lessons can only be held in a computer lab, and there are n_{cl} computer labs in an institution. Apart for general room cardinality constraints, the first phase also has to ensure that for each period at most n_{cl} programming lessons are scheduled. This is encoded by additional conditions

$$\mathsf{cardinality}(\{x_{tgsndp} \mid tsgn \in \text{lessons that must be held in computer lab}\}) \leq n_{cl},$$

for each day d and period p.

Once the first phase is finished, rooms are allocated in the second phase. The input to the second phase are the values for all variables x_{tsgndp}. Then, variables $x_{tsgndpr}$ and x_{tdpr} are formed and connected with x_{tsgndp} as in the naive room allocation, but only for those variables x_{tsgndp} and x_{tdp} that are found to be true after the first phase. Also, the correctness conditions are the same as in the naive room allocation. If it is required that rooms are not changed during the lesson, the conditions $x_{tsgndpr} \Leftrightarrow x_{tsgnd(p+1)r}$, for each lesson $tsgn$ that is assigned in day d to both periods p and $p+1$, and each room r.

3.3.1 Complexity of Encoding

The first phase of encoding does not introduce any new variables, and it inherits the complexity $O(n_l \cdot n_d \cdot n_p)$ of the basic encoding. Cardinality constraints introduce $O(n_d \cdot n_p \cdot n_t \cdot n_r)$ auxiliary variables and $O(n_d \cdot n_p \cdot n_t \cdot n_r)$ clauses.

In the second phase there are only $O(n_l \cdot n_r)$ variables, and it is much simpler that the first phase.

4 Case Study

In this section we give a case study of timetabling for three educational institutions in Belgrade based on the described technique.

4.1 Implementation

In this section we describe our implementation that is entirely done in C++.

The timetabling system that we have implemented consists of three components:

SAT Encoder - The role of the SAT encoder is to convert a timetable specification file to a SAT formula in DIMACS format. We found that the most convenient for us way to specify various kinds of requirements was by using plain text specification. Although our current implementation does not offer a GUI for input specification it is possible to implement one if some users prefer to use it. Currently, the timetable specification files are plain ASCII files that contain list of lessons that need to be scheduled and various other timetable requirements. They are given in custom syntax. For example:

```
days: mon tue wed thu fri
periods: 1-7
lessons:
   teacher1     group1, group2    subject1   2+1  room1
   teacher2     group1            subject2   3    room1, room2
   teacher2     group2            subject2   3    room1, room2
requirements:
   -teacher1_mon
   -group2_tue_7 | -group2_thu_1
```

In the given example, there are five working days with seven working periods. The teacher $teacher_1$ teaches the $subject_1$ simultaneously to groups $group_1$ and $group_2$ two times in a week — once for two periods and once for one period in the room $room_1$. The teacher $teacher_2$ teaches the $subject_2$ separately to $group_1$ and $group_2$ once a week for three periods in room $room_1$ or room $room_2$. It is required that $teacher_1$ does not give lessons on Monday and that $group_2$ does not attend lessons either on Tuesday's seventh period or on Thursday's first period.

SAT Solver - Once the DIMACS file is generated it is given to a SAT solver which searches for its models. The system uses the SAT solver ARGO-SAT developed by the author within the Automated Reasoning GrOup at Faculty of Mathematics [4] , Belgrade. We have also experimented with using different SAT Solvers (e.g., MiniSAT, yices), and the performance was very similar on our benchmarks.

SAT Decoder - If the solver successfully finds a model of the generated formula, the decoder converts it to a readable timetable. The timetable is given in HTML format and separate web pages are generated for all teacher, groups and rooms. [5]

4.2 Institutions.

In this section we briefly describe the institutions for which we have constructed timetables.

[4] The web page of ARGO-SAT is `http://argo.matf.bg.ac.yu/`

[5] A sample timetable output is available at `http://www.matf.bg.ac.yu/~filip/timetable/`

The Architectural Technical High School (ATS) can be considered a large school: it has 30 different groups of pupils (around 35 pupils each) and 85 teachers, working mostly as full-time employees. The school works for five days in a week in two shifts. The first shift consists of 15 groups of pupils (1st and 3rd grade) and the second shift consists of 15 groups of pupils (2nd and 4th grade). Each shift is divided to seven 45 minute periods and single lessons (which are also 45 minutes long) are scheduled to fit in those seven periods. Longer lessons span through several periods. Each pupil group has its own classroom. The only shared rooms are a gym, three computer laboratories, and three science cabinets, so room allocation was not an important issue for this institution. The total number of lessons that had to be scheduled was around 1200.

The Faculty of Mathematics (MATF) can be considered a medium-to-large size university department: it has 30 different student groups and 80 teachers (working mostly as full-time employees). The department works for five days in a week in twelve one-hour periods (from 8am to 8pm). Single lessons that are 45 minutes long are scheduled to fit into these periods (1 hour period includes a 45 minute lesson and a 15 minute break). Longer lessons span through several periods. It has two different buildings, in the different parts of the city. First building has eleven lesson rooms varying in size and installed equipment (three amphitheaters, three computer laboratories - two small and one large, and five classrooms - three medium size and two small). The second building has three lesson rooms (one amphitheater and two computer laboratories). The total number of lessons that had to be scheduled was around 600.

The Faculty of Computer Science (RAF) can be considered a small-to-medium size university department: it has 20 different student groups and 36 teachers (working mostly as part-time employees). The department works for five days in a week in twelve one-hour periods (from 9am to 9pm). Single lessons that are around 45 minutes long are scheduled to fit into these twelve periods. Longer lessons span through several periods. It has one building and six different rooms (two amphitheaters, three computer laboratories and one small classroom). The total number of lessons that had to be scheduled was around 300.

4.3 Results

One of the main concerns when building a timetabling system is its efficiency and performance. Our main goal was also to be able to formulate and satisfy different requirements that were imposed on us. We are quite satisfied with the performance of our system, and we are also aware that its efficiency can further be improved as suggested in §6.

Institution	Room allocation	Variables	Clauses	Time [6]
MATF- building A	naive	$\approx 43\,000$	$\approx 305\,000$	1 min
MATF- building B	cardinality	$\approx 132\,000$	$\approx 781\,000$	5 min
RAF	naive	$\approx 117\,000$	$\approx 961\,000$	4 min
ATS - shift A	cardinality	$\approx 62\,000$	$\approx 790\,000$	350 min
ATS - shift B	cardinality	$\approx 65\,000$	$\approx 825\,000$	223 min

Fig. 3. Results summary.

All timetables that we have constructed were constrained with a large number of different specific constraints described in Section 2. Usually not all constraints were satisfiable if taken together. We used an ad hoc approach to determine the minimal set of constraints that should be relaxed in order to get a valid timetable.

In the first phase, mutually conflicting requirements were detected and resolved by removing the less important ones. Set of formula instances, differing in the number of imposed constraints were being solved. Upon unsatisfiability groups of constraints were manually withdrawn and upon satisfiability groups of constraints were reintroduced. Finally, a satisfiable formula that contains maximal number of given constraints was identified. Usually, the unsatisfiability was due to local conflicting constraints and was very quickly detected. Also, in all our cases, a very small number of constraints had to be withdrawn in order to get a satisfiable solution.

In the following phase, we tried to improve the quality of the found solution. The least priority was given to idle periods constraints and therefore, the total number of idle periods was used as a measure of timetable quality. We generated and solved several instances varying the total number of allowed idle periods. In high school, only the teachers were allowed to have idle period and in university departments only students were allowed to have idle periods (idle periods for teachers that specifically asked to have idle periods were not counted). In every instance, only single idle periods were allowed. The exception was the small building of MATF where, due to its small number of rooms, double idle periods had to be allowed in order to get any satisfiable solution.

Some summary statistics about the final satisfiable instances are given in Figure 3. The runtime significantly varied from instance to instance. However, no consistent trend in those variations was detected. Sometimes, the more con-

[6] All computations were performed on a 2.33MHz computer with 1GB of RAM memory using ARGO-SAT.

Institution	Room allocation	Variables	Clauses	Time
MATF- building B	naive	$\approx 344\,000$	$\approx 1\,433\,000$	23 min
MATF- building B	cardinality	$\approx 132\,000$	$\approx 781\,000$	5 min

Fig. 4. Room allocation.

straint problems needed less time to finish, and sometimes they needed more time. In any case, the times given in Figure 3 can be seen as fair representatives of the average system efficiency and, because of that, we choose to give them in minutes instead of in seconds.

The most amount of time is spent in the SAT solving process. Namely, the encoding and decoding together take less than a minute for every timetable we have constructed.

Although the overall time varied from instance to instance, all run times for both university institutions were under 10 minutes. We have observed that the time needed to build a timetable for high school was more than an order of magnitude higher than time needed to build a timetable for an university department, but the size of formulae were similar. This was not surprising because the number of lessons in high school was larger than in the university while the requirements were stronger. As already said, two shifts in high school were processed independently. We assume that the hardness of the high school timetable construction comes from the fact that every group in high school had 33 or 34 lessons per week that had to fit in 35 available slots, and every group in university had about 25 lessons that had to fit to 60 available time slots. Also, the teachers had 20 lessons in average compared to only 8 in the university. The longest runtime that we have encountered for a high school shift was little under 6 hours and the final generated timetable had 25 idle hours total. Interestingly, the timetable for the other shift was constructed in less than 4 hours and had only 10 idle hours total. This suggest that not only the size of the problem, but also the structure of the formula influenced by interactions between groups and teachers in the lesson plan plays a big role in the hardness of the timetabling problem.

Our experience show that room allocation constraints add much to the problem complexity since on average every room is occupied for more that 65% of the time. In some situations, 58 lessons were successfully scheduled in a room that has 60 available time-slots, which almost impossible to accomplish if timetabling is done manually. Naive room allocation was possible to use only for small buildings with a small number of lessons that should be scheduled. Figure 4 shows that on the same instance cardinality based room allocation clearly outperforms the naive one.

5 Related Work

Course timetabling is a multi-dimensional NP-Complete problem that has generated hundreds of papers and thousands of students have attempted to solve it for their own school. There are several types of algorithms that have been applied to these problems and some of their surveys can be found in [21,9,2].

Various meta-heuristic approaches such as simulated annealing, tabu search, genetic algorithms and hybrid approaches have been investigated over the last two decades for timetabling. Some very good results have been reported in PATAT conferences [6,3,5,4,8,7]. Meta-heuristic methods begin with one or more initial solutions and employ search strategies that try to avoid local optima. All of these search algorithms can produce high quality solutions but often have a considerable computational cost.

Constraint based approaches model timetabling problem as a set of variables (i.e., events) to which values (i.e., resources such as rooms and time periods) have to be assigned to satisfy a number of constraints [1,20]. Usually a number of rules is defined for assigning resources to events. When no rule is applicable to the current partial solution a backtracking is performed until a solution is found that satisfies all constraints.

A SAT encoding for timetabling was used by Chin-A-Fat and Hartog in [10,14]. However, the system that we are proposing offers specification of much wider range of requirements. For example, in [14] it is not possible to have lessons that last for more than two periods, idle periods are not constrained at all, the room allocation problem is not appropriately addressed etc. The run time only for the SAT encoding phase reported in [14] is several hundred minutes, while it is under a minute in our system.

6 Further Work

The main drawback of the approach that we have described in this paper is that all constraints in the formula that is being solved are considered to be hard constraints. When the unsatisfiability of the formula is detected, in the current implementation some constraints must be removed manually. This process can be automated and the system should detect the soft constraint with the minimal priority and remove it.

The other approach that is also planned for our further work is to make possible to incrementally and automatically add soft constraints until a good enough solution is found. When a new constraint is added, the solver should start the search from the model that was previously found. We suppose that this could significantly reduce its runtime, because some parts of the formula that represent the timetable constraints not affected by the new constraint and that are hard to solve are already solved and their solution is contained in the previous model.

Although the approach with adding constraints can take more time than the one with removing constraints, it can be safer, since in every moment in time the best timetable so far is known.

These approaches do not guarantee that the best quality solution will be found. We have also made some experiments with using max-sat solvers [15] which give that guarantee, but the obtained results were not good enough for practical use.

The only inference mechanism that modern SAT solvers use is *unit propagation.* Most constraints in our encoding are single constraints. While experimenting with our system we noticed that the unit propagation is too weak inference mechanism for these kinds of constraints, and its pruning power is very low. For instance, assume that in the current search state it is known that a teacher $T1$ gives a lesson to the group G either during the first or during the second period on Monday, and that a teacher $T2$ also gives lesson to the group G either during the first or during the second period on Monday. Since these two teachers consume the first two periods on Monday for the group G, no other teacher could give lessons to group G neither during first nor second period on Monday. Unit propagation cannot detect this until it is effectively assumed that some other teacher gives lesson during the first or second period on Monday, and then it detects the inconsistency an backtracks. Techniques from constraint programming, like the *alldifferent hyper-arc consistency* [19], can detect these kinds of situations and improve the inference and search space pruning. As As indicated in [17], combining SAT and CP can be done through the SMT (*Satisfiability modulo theories*) framework, and this is going to be the next step in our research on this topic.

7 Conclusions

In this paper, we have described a SAT based approach for teaching timetabling. Our experience shows that some institutions have very specific requirements that are considered to be obligatory and many existing approaches are not capable of efficiently dealing with such specific constraints. Through its large set of variables, the SAT encoding that we have developed offers a rich modelling language and the possibility to formulate a very diverse set of requirements.

The results obtained in three different institutions that we have worked with are very promising and show that this approach can be used as an automatic approach for constructing timetables. Also, one of the main advantages of the system that we propose is that it is very easy to implement. The encoder and decoder together take less then 1000 lines of C++ code, and there are a lot of free, efficient SAT solvers that can be used. Although the observed run time was very acceptable, it can still be improved.

References

[1] S.C. Brailsford, C.N. Potts, and B.M. Smith. Constraint satisfaction problems: Algorithms and applications. Papers 98-142, University of Southampton - Department of Accounting and Management Science, 1998.

[2] E. K. Burke and S. Petrovic. Recent research trends in automated timetabling. *European Journal of Operational Research*, pages 266–280, 2002.

[3] Edmund K. Burke and Michael W. Carter, editors. *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT'97, Toronto, Canada, August 20-22, 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*. Springer, 1998.

[4] Edmund K. Burke and Patrick De Causmaecker, editors. *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002, Gent, Belgium, August 21-23, 2002, Selected Revised Papers*, volume 2740 of *Lecture Notes in Computer Science*. Springer, 2003.

[5] Edmund K. Burke and Wilhelm Erben, editors. *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Konstanz, Germany, August 16-18, 2000, Selected Papers*, volume 2079 of *Lecture Notes in Computer Science*. Springer, 2001.

[6] Edmund K. Burke and Peter Ross, editors. *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*, volume 1153 of *Lecture Notes in Computer Science*. Springer, 1996.

[7] Edmund K. Burke and Hana Rudová, editors. *Practice and Theory of Automated Timetabling VI, 6th International Conference, PATAT 2006, Brno, Czech Republic, August 30 - September 1, 2006, Revised Selected Papers*, volume 3867 of *Lecture Notes in Computer Science*. Springer, 2007.

[8] Edmund K. Burke and Michael A. Trick, editors. *Practice and Theory of Automated Timetabling V, 5th International Conference, PATAT 2004, Pittsburgh, PA, USA, August 18-20, 2004, Revised Selected Papers*, volume 3616 of *Lecture Notes in Computer Science*. Springer, 2005.

[9] Michael W. Carter and Gilbert Laporte. Recent developments in practical course timetabling. In *PATAT '97: Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling II*, pages 3–19, London, UK, 1998. Springer-Verlag.

[10] Kenneth Chin-A-Fat. *School Timetabling using Satisfiability Solvers.* Msc thesis, Technical University Delft, The Netherlands, September 2004.

[11] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[12] Niklas Een and Niklas Sorensson. An extensible sat-solver. pages 502–518. 2004.

[13] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver, 2002.

[14] Jantien Hartog. *Timetabling on Dutch High Schools: Satisfiability versus gp-Untis.* Msc thesis, Technical University Delft, The Netherlands, 2007.

[15] Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat: A new weighted max-sat solver. In *SAT*, pages 41–55, 2007.

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

[17] R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In A. Biere and C. P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2006.

[18] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of the 11th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2005)*, pages 827–831, Sitges, Spain, October 2005.

[19] W. van Hoeve. The alldifferent constraint: A survey, 2001.

[20] G. M. White. Constrained satisfaction: not so constrainted satisfaction and the timetabling problem. In *PATAT'00, volume 1*, Konztanz, Germany, August 2000.

[21] R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Phd thesis, Technische Universiteit Eindhoven, The Netherlands, 2002.