10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

Solving High School Timetabling with Satisfiability Modulo Theories

Emir Demirović · Nysret Musliu

Abstract High School Timetabling (HSTT) is a well known and wide spread problem. The problem consists of coordinating resources (e.g. teachers, rooms), time slots and events (e.g. lectures) with respect to various constraints. Unfortunately, HSTT is hard to solve and just finding a feasible solution for simple variants of HSTT has been proven to be NP-complete. In addition, timetabling requirements vary from country to country and because of this many variations of HSTT exist. Recently, researchers have proposed a general HSTT problem formulation in an attempt to standardize the problem from different countries and school systems.

In this paper, for the first time we provide a new detailed modeling of the general HSTT as a Satisfiability Modulo Theory (SMT) problem in the bit vector form. In addition, we present preliminary experimental results and compare to the winner of the Third International Timetabling Competition 2011 (ITC), using both artificial and real-world instances, all of which were taken from ITC 2011 benchmark repository. Our current approach provides feasible solutions for some examples, which in some cases could not have been obtained with the competition winner algorithm within 24 hours.

Vienna University of Technology
Database and Artificial Intelligence Group
E-mail: {*musliu* ∨ *demirovic*}@dbai.tuwien.ac.at

Keywords SMT · High School Timetabling · Modeling

1 Introduction

In this paper, we describe a modeling of the high school timetabling problem (HSTT) as a Satisfiability Modulo Theory (SMT) problem. By doing so, we were able to find feasible solutions to some problem instances, which were proposed by the International Timetabling Competition 2011 [13], which in some cases could not have been obtained using the winning algorithm of the competition in 24 hours, GOAL. In two smaller instances, optimization could also be performed, rather than just finding a feasible solution, but optimization is difficult for our method at its current state.

The problem of timetabling is to coordinate resources (e.g. rooms, teachers, students) with time slots in order to fulfill certain goals or events (e.g. lectures).

Timetabling is encountered in a number of different domains. Every educational institution, airport, public transport system, etc requires some form of timetabling. The difference between a good and a bad timetable can be significant, but constructing timetables by hand can be time consuming, very difficult, error prone and in some cases impossible. Therefore, developing high quality algorithms which would automatically do so is of great importance. Note that there are many different timetabling problems and algorithms for one type of problem (e.g. HSTT) might not directly be suitable for another problem (e.g. University Timetabling), because of their different requirements. In this work, we focus on HSTT. Respecting constraints is very important, as timetables directly contribute to the quality of the educational system, satisfaction of students and staff and other matters. Every timetabling decision affects hundreds of students and teachers for prolonged amounts of time, since each timetable is usually used for at least a semester.

Unfortunately, High School Timetabling is hard to solve and just finding a feasible solution of simple variants of High School Timetabling has been proven to be NP-complete [7].

Apart from the fact that problems that need to be solved can be very large and have many different constraints, high school timetabling requirements vary from country to country and because of this many variations of the timetabling problem exist. Because of this, it was unclear what the state of the art was, as comparing algorithms was difficult.

Recently have researchers proposed a general high school timetabling problem formulation [14] in an attempt to standardize the problem from different countries and school systems and this formulation has been endorsed by the Third International Timetabling Competition 2011 (ITC 2011) [13] [14]. This was a significant contribution, as now algorithms can be compared on standardized instances, that were proposed from different researchers [12].

The winner of the competition was the group GOAL, followed by Lectio and HySST. All of the algorithms were based on heuristics. In GOAL, an initial solution is generated, which is further improved by using Simulated Annealing and Iterated Local Search, using seven different neighborhoods [8]. Lectio uses an Adaptive Large Neighborhood Search [16], while HySST uses a Hyper-Heuristic Search [9]. Recently, [17] used Integer Programming (IP) in a Large Neighborhood Search algorithm and [15] introduced a two phase IP algorithm for a different timetabling problem, but have managed to adjust the method for a number of high school timetabling instances.

All of the best algorithms on the competition were heuristic algorithms and this is why introducing a new exact method (our approach) is important. Some advantages are being able to provide proofs of optimality or infeasibility, calculate lower bounds as well as an opportunity to hybridize algorithms, as well as create valuable benchmarks for SMT solvers. Even though significant work has been put into HSTT, optimal solutions for most instances are still not known and this is still an active research area.

In this paper, we investigate the formulation of HSTT as SMT. A SMT problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. It is a generalization of the Satisfiability problem (SAT) in which sets of variables are allowed to be replaced by predicates from a variety of underlying theories. SMT is usually used for verification and program analysis, but researchers have recently been investigating solving Constraint Satisfaction Problems with SMT [3] and other optimization problems [11]. There is a natural connection between timetabling and logical formulas. HSTT as itself has many logic based characteristics and as such some of its constraints can easily be encoded as SMT. This has motivated us to investigate how efficient can a SMT formulation for HSTT be. However, due to the generality of the specification that we use, devising a complete model is not a trivial task, because as we will see later, some of the constraints are cumbersome. In addition to formulating a general formulation, one needs to take care of important special cases which arise in practice and can significantly simplify the encoding.

The main contributions of this paper are as follows:

- We show that HSTT can be modeled as a SMT problem, despite the fact that HSTT is very general and has many different constraints, both hard and soft versions. All constraints are included in their general formulations, as well as important alternative encodings for special cases.
- We give preliminary experimental evaluation of our model using both artificial and real-world instances, all of which were taken from the Third International Timetabling Competition 2011 benchmark repository. A comparison with the winning algorithm from ITC 2011 is given.

The rest of the paper is organized as follows: in the next section, we give a more detailed look into the problem description which serves as an introduction for Section 3, where the detailed presentation of our approach in modeling HSTT as

SMT is given. In Section 4, we provide computational results obtained on artificial and real life problems. Finally, we give concluding remarks and ideas for future work.

2 Problem Description

In our research we consider the general formulation of the High School Timetabling problem, as described in [14].

The general High School Timetabling formulation specifies three main entities: times, resources and events. Times refer to time slots which are available, such as Monday 9:00-10:00, Monday 10:00-11:00, etc. Resources correspond to available rooms, teachers, students, etc. The main entities are the events, which in order to take place require certain times and resources. An event could be a Mathematics lecture, which requires a math teacher and a specific student group (both considered resources) and two time slots.

Constraints impose limits on what kind of assignments are legal. These may constraint that a teacher can teach no more than five lessons per day, that younger students should attend more demanding subjects (e.g. Mathematics) in the morning, etc. We describe the constraints in the next section when we present the SMT formulations.

Each constraint has a nonnegative cost function associated with it, which penalizes assignments that violate it. It is important to differentiate between hard and soft constraints. Hard constraints are constraints that define the feasibility of the solution and are required for the solution to make sense, while soft constraints define desirable situations, which define the quality of the solution. Therefore, the cost function consists of two parts: infeasibility value and objective value. The goal is to first minimize the infeasibility and then minimize the objective function value part. The exact way these two are calculated will be discussed in the next section.

3 Our Approach - Modeling HSTT for SMTs

Modern SMT solvers (e.g. z3 [5], Yices [6]) offer a number of underlying theories to choose from, which are described in detail in the standardization SMT-LIB [2]. Modeling of the problem at hand depends heavily on which theory we have chosen. In our initial phase, we used two different theories: linear integer arithmetic and bit vector. In the following, we present a bit vector formulation for HSTT, as it was more successful in initial experiments and afterwards discuss briefly the problems encountered with linear integer arithmetic.

3.1 Bitvector Theory

A bitvector is a vector of bits. The size of the vector is arbitrary, but fixed. A number of standard operations (e.g. *addition*, *and*, *or* operations on bitvectors) and predicates (e.g. equality) are defined over bitvectors and an instance consists of a conjunction of predicates. Most SMT solvers accept formulas written in SMT-LIB file format, but can have their own formats, like Yices. Since these files use prefix notation, we will do so as well in the description of the constraints with the addition of brackets and comas in order to ease reading. E.g. In infix notation one would write $(a = b)$, while in prefix notation the same expression would be written as $(= \ a \ b)$, while we choose to write $(= (a, b))$.

Most operations are interpreted as usual and all bitvector operands are of the same length. In the following we present some of the notations we will use in which bv_a and bv_b are bitvectors and k is a constant integer:

- $inv(bv_a)$ - inverts bv_a bits (e.g. $inv(1011001) = 0100110$).
- $add(bv_a, bv_b)$ - adds two bitvectors in the same way two unsigned integers would be added (overflow might occur).
- $or(bv_a, bv_b)$ - performs bitwise *or* on its operands.
- $lshift(bv_a, k)$ - applies noncyclic left shift by k operation on bv_a (e.g. $lshift(10011, 2) = 01100$).
- $rshift(bv_a, k)$ - similar to $lshift$, but uses right shifting.

– $extract(bv_a, k)$ - returns the $k-th$ bit of bv_a

An example SMT instance would be the following:

$$(= (1010, lshift(bv_a, 1)) \wedge (< (bv_a, 1000)) \qquad (1)$$

The problem is to determine whether there exists a bitvector bv_a for which the above formula holds. It states that bv_a must be equal to 1010 after it is shifted to the left by one place (first clause) and bv_a must be less than (in the standard way binary numbers are compared) 1000 (second clause). The formula is satisfiable and $bv_a = 0101$ is a model since it satisfied both clauses, while $bv_a = 1101$ is not due to not satisfying the second clause. Note that this is a decision problem.

In the optimization variant, weights may be assigned to clauses and the goal is to find a model which will satisfy all clauses without weights and will minimize the sum of weights of unsatisfied clauses. Optimization is not part of standard SMT solvers by default, although Yices [6] supports it. E.g. if we assigned a weight of 10 to the second clause in the previous example, both $bv_a = 1101$ and $bv_a = 0101$ would be considered solutions to the problem, but the latter would be considered a better solution.

3.1.1 Variables and Definitions

For each event e (e.g. a lesson), we create a number of bit vectors all of length n, where n is the number of time slots available in the instance. The vectors along with their meanings are as follows:

– Y_e - the $i-th$ bit is set (a bit is *set* if it has value 1) if the event is taking place at time slot i and is not set otherwise. In xHSTT terminology, Y_e covers all subevents of event e. This implies that two subevents of the same event can never clash in this representation.
– S_e - the $i-th$ bit is set if the $i-th$ time slot is declared as a starting time for event e and is not set otherwise.

– $K_{e,d}$ - the $i-th$ bit is set if the $i-th$ time slot is declared as a starting time of duration d for event e and is not set otherwise.

As an example of the above variables, take the following bitvectors:

7	6	5	4	3	2	1	0	$(time\ slot)$
0	1	1	0	0	1	1	0	(Y_e)
0	1	1	0	0	0	1	0	(S_e)
0	1	1	0	0	0	0	0	$(K_{e,1})$
0	0	0	0	0	0	1	0	$(K_{e,2})$

(2)

From Y_e, we see that event e (e.g. a Math lesson) is taking place at time slot 1, 2, 5 and 6, because those bits are set within Y_e. Similarly, time slots 1, 5 and 6 are labeled as starting times from S_e, meaning event e has been split into three subevents. Time slot 1 is labeled as a double lesson by $K_{e,2}$, while 5 and 6 as lessons of duration 1 by $K_{e,1}$. Note that time slot 5 could have also been labeled as a double lesson instead of having two lessons of duration 1. Reasons for choice one possibility over the other is regulated by constraints.

In the formal specification of HSTT, there are no restrictions on what can be defined as a starting point. One could regard a starting point as a time t where a lecture takes place, but has not took place at $t-1$. However, while this is true, this cannot be the only case when a time would be regarded as a starting time, since e.g. time $t=5$ and $t=6$ might be interpreted as *last time slot of Monday* and *first time slot of Tuesday* and an event could be scheduled on both of these times, but clearly we must regard both times as starting times, since a double lecture does not extend over such long periods of time. Therefore, any time can in general be regarded as a starting time. It is of interest to note that the previous assignment, by the general formulation, could also be treated as a double lesson for the purpose of constraints, even though it extends over two days. Constraints give more control over these kind of assignments.

Formalities that are tied to starting times with regard to the specification are expressed as follows:

If a starting time for event e has been assigned at time t, then the corresponding event must also take place at that time:

$$\bigwedge_{\forall e \in E} (= (or(S_e, Y_e), Y_e)) \tag{3}$$

The *or* and = statements are required to ensure that Y_e has bits set at least in the same positions as S_e. This type of encoding is used frequently and one should become accustomed to it.

Event e starts at time t if e is taking place at time t and it is not taking place at time $(t-1)$:

$$\bigwedge_{\forall e \in E_{spec})} = (or(and(Y_e, lshift(inv(Y_e, 1)), S_e), S_e) \tag{4}$$

Note that the ordering of the application of inv and $lshift$ is important.

Let K_e^+ be the bit vector which $i-th$ bit is set if any of $K_{e,d}$ vectors have an $i-th$ bit set. This is obtained by taking the *or* of all of the $K_{e,d}$. If time t has been set as a starting time, associate a duration with it:

$$\bigwedge_{\forall e \in E_{spec}} (= (K_{e,d}^+, or(S_{e,t}, K_{e,d}^+)) \tag{5}$$

Let S_e^d be the vector obtained as $rshift(S_e, d)$. If a subevent of duration d has been assigned and immediately after the event is still taking place, then assign that time as a starting time:

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ \forall d \in D}} (= (or(and(rshift(Y_e, d), K_{e,d}), S_e^d), S_e^d)) \tag{6}$$

Let $K_{e,d}^*$ be the vector obtained by taking the *and* of all of Se^k for $k = 1..d$ and Y_e. When a bit in $K_{e,d}$ is set, ensure that

the event in question must take place for d consecutive hours during this specified time:

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ \forall d \in D}} (= (or(K^*_{e,d}, K_{e,d}), K^*_{e,d})) \tag{7}$$

Let $K^{inv(k)}_{e,d}$ be the vector obtained as $rshift(inv(K_{e,d}), k)$, $K^{\#k}_{e,d}$ be the vector obtained by taking *and* of all $K^j_{e,d}$ for $j = 1..(k-1)$ and $K^{\&k}_{e,d}$ be the vector obtained by taking the *and* of all $K^{\#i)}_{e,d}$ for $i \neq k$. If a duration has been specified for time t, make sure that no other starting point other appropriate $K_{e,t,d}$ variables must be false:

$$\bigwedge_{\forall e \in E_{spec}\ atop d \in D} (= (or(K_{e,d}, K^{\#d}_{e,d}), K_{e,d})) \tag{8}$$

3.2 Cardinality Encodings

An important constraint that arises often is to determine the number of set bits in a bit vector, as well as to impose penalties if the appropriate number of bits are not set. E.g. if an event must take place for two hours, then exactly two bits in its Y_e must be set.

Let us define a unary operation $reduceBit(bv_a) = bv_a \wedge sub(bv_a, 1)$. When applied to bv_a, as the name suggests, it produces a new bitvector which has one less bit set then bv_a. For example:

$$\begin{array}{cccccccc} & 1 & 1 & 0 & 1 & 0 & 0 & (bv_a) \\ \wedge & 1 & 1 & 0 & 0 & 1 & 1 & (sub(bv_a, 1)) \\ \hline & 1 & 1 & 0 & 0 & 0 & 0 & (reduceBit(bv_a)) \end{array} \tag{9}$$

The original bitvector had three bits set, while the produced one was two set. The *reduceBit* operations is an important part for defining cardinality constraints.

In order to ensure that *at least* k bits are set in a bitvector, we apply *reduceBit* $k-1$ times and require that the resulting

bitvector must be different from zero. For *at most* k, we apply $reduceBit$ k times and constrain that the resulting bitvector must be equal to zero. For *exactly* k we encode *at least* k and *at most* k. For example, asserting that *at least* 2 bits are set is done in the following way:

$$\begin{array}{cccccccc} & 1 & 1 & 0 & 1 & 0 & 0 & (bv_a) \\ \wedge & 1 & 1 & 0 & 0 & 1 & 1 & (sub(bv_a, 1)) \\ \hline & 1 & 1 & 0 & 0 & 0 & 0 & (reduceBit(bv_a)) \\ \wedge & 1 & 0 & 1 & 1 & 1 & 1 & (sub(reduceBit(bv_a)), 1) \\ \hline & 1 & 0 & 0 & 0 & 0 & 0 & (reduceBit(reduceBit(bv_a))) \end{array} \tag{10}$$

Since the final bitvector is different from the zero bitvector, we conclude that *at least* 2 bits are set in bv_a.

For the soft cardinality constraints which penalize the objective value if a certain number of bits is set rather than forbidding their assignments, a similar technique. For *at least* k, it is asserted before each $i-th$ application of $reduceBit$ that the current bitvector is different from zero and is penalized by some weight if it is not the case. For example, asserting that *at least* 2 bits are set is done in the following way for the soft version:

$$\begin{array}{cccccccc} & 0 & 1 & 0 & 0 & 0 & 0 & (bv_a \neq 0, no\ penalty) \\ \wedge & 0 & 0 & 1 & 1 & 1 & 1 & (sub(bv_a, 1)) \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & (reduceBit(bv_a) = 0, penalize) \end{array} \tag{11}$$

Note that we checked for penalties in two cases (for the initial bitvector bv_a and $reduceBit(bv_a)$), but only one case was penalized. For *at most* k, a similar algorithm is used. First, $bitReduce$ is applied k times as in the regular cardinality constraint version. Then, $bitReduce$ is applied $n-k$ times to this bitvector (n is the size of bv_a) and before each application it is asserted that the current bitvector is zero and is penalized by some weight if it is not the case. Note that if we have some hard constraint limiting the maximum number of bits that

may be set in a bitvector to some k_{max}, we do not perform the second part of the algorithm $n-k$ times, but rather just $k_{max}-k$ times. This was used frequently in the implementation.

3.3 Constraints

Each constraint has its points of application and each point generates a number of deviations. Cost of the constraint is obtained by applying a cost function on the set of deviations produced and multiplying it by a weight. A cost function may simply be the sum of all deviations. Our current implementation supports cost functions of sums of deviations, while cost function sum of squares of deviations is supported by the model but not implemented. The HSTT specification allows for other cost functions as well, such as square of sums, but we do not have an encoding for them currently. Fortunately, only two instances use nonsupported cost functions (KosovaInstance1 and StPaulEngland instances). Some constraints are always encoded as hard (e.g. Avoid Clashes Constraints, Assign Times Constraints) and because of this we avoid discussing their soft constraint variants.

We simplify the objective function by not tracking the infeasibility value, rather regarding it was zero or nonzero. By doing so we simplify the computation, possibly offering a faster algorithm.

E, T and R are sets of events, times and resources, respectively. Each constraint is applied to some subset of those three and will be denoted by E_{spec}, T_{spec} and R_{spec}. These subsets are naturally in general different from constraint to constraint. Note that it is possible to have several constraints of the same type, but with different subsets defined for them.

We present encodings used in the experimental results, in which we assume that all resources are already assigned to events. We make this assumption as this eases the modeling and readability of the constraints. Later on we provide a description on how this limitation can be overcome.

3.3.1 Assign Time Constraints

Every event must be assigned a given amount of times. For example, if a lecture lasts for two hours, two time slots must be assigned to it.

Each event's Y_e vector must have exactly d bits set, where d is the duration of the event:

$$\bigwedge_{\forall e \in E} (exactly\_d[Y_{e,t} : t \in T]) \tag{12}$$

3.3.2 Avoid Clashes Constraint

Specified resources can only be used at most by one event at a time. For example, a student may attend at most one lecture at any given time.

Let $E(r)$ be the set of event which require resource r. For each resource r, each time slot i and each combination of two Y_e vectors of events from $E(r)$ at most one bit at $i-th$ location may be set.

$$\bigwedge_{\forall r \in R \forall e_1, e_2 \in E(r) e_1 \neq e_2} (= (and(Y_{e_1}, Y_{e_2}), 0)) \tag{13}$$

For example, for resource r let $E(r) = \{Y_{e_1}, Y_{e_2}, Y_{e_3}\}$ and let this constraint be defined for r.

$$\begin{array}{cccccccc} & 0 & 0 & 1 & 1 & 1 & 1 & (Y_{e_1}) \\ \wedge & 0 & 1 & 0 & 0 & 0 & 0 & (Y_{e_2}) \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & (=0) \end{array} \tag{14}$$

The previous check ensures that there are no clashes for Y_{e_1} and Y_{e_2}.

$$\begin{array}{cccccccc} & 0 & 0 & 1 & 1 & 1 & 1 & (Y_{e_1}) \\ \wedge & 0 & 0 & 0 & 0 & 1 & 0 & (Y_{e_3}) \\ \hline & 0 & 0 & 0 & 0 & 1 & 0 & (\neq 0, violated) \end{array} \tag{15}$$

However, since a clash exists between Y_{e_1} and Y_{e_3}, the constraint is detected to be violated and some changes to the Y_{e_1}, Y_{e_2}, Y_{e_3} bitvectors must be made.

3.3.3 Avoid Unavailable Times Constraints

Specified resources are unavailable at certain times. For example, a teacher might be unable to work on Friday.

For each resource r, each unavailable time slot i and each Y_e vector of events from $E(r)$ we force the $i-th$ bit to be set to zero.

$$\bigwedge_{\forall r \in R_{spec} \forall e \in E(r) \forall i \in T_{spec}} (= (extract(Y_e, i), 0)) \tag{16}$$

If this constraint is used as a soft constraint, all of the above clauses would be assigned the given weight, as points of application are resources and deviations are calculated as the number of times a resource is assigned to an unavailable time.

For example, if time slots 1 and 4 are unavailable for resource r and event e requires r:

7	6	5	4	3	2	1	0	$(time\ slot)$
0	1	1	0	0	1	1	0	(Y_e)

(17)

Y_e would violate this constraint, as Y_e is taking place on time slot 1, which is a unavailable one, meaning that a different bitvector needs to be assigned to Y_e.

3.3.4 Split Events Constraints

This constraint has two parts.

The first part limits the number of starting times an event may have within certain time frames. For example, an event may have at most one starting time during each day, preventing it from being fragmented within days.

The second part limits the duration of the event for a single subevent. For example, if four time slot must be assigned to a Mathematics lecture, we may limit that the minimum and maximum duration of a subevent is equal to 2, thus ensuring that the lecture will take place as two blocks of two hours, forbidding having the lecture performed as one block of four hours.

This constraint specifies the minimum A_{min} and maximum A_{max} amount of starting times for the specified events:

$$\bigwedge_{\forall e \in E_{spec}} (atLeast\_A_{min}[S_{e,t}] \wedge atMost\_A_{max}[S_{e,t}) \tag{18}$$

In addition, this constraint also imposes the minimum d_{min} and maximum d_{max} duration for each subevent.

$$\bigwedge_{\forall e \in E_{spec} \forall d \in ili < d_{min} \vee i > d_{max}} (atMost\_0[K_{e,d}]) \tag{19}$$

If the constraint is specified as soft, then the soft cardinality encodings are used instead. Points of applications are events and deviations are calculated as the number of times an event has been assigned a duration which is less than d_{min} or greater than d_{max}, plus the amount by which the number of starting times for the event event falls short of A_{min} or exceeds A_{max}.

3.3.5 Spread Events Constraints

Certain events must be spread across the timetable, e.g. in order to avoid situations in which an event would completely be scheduled only in one day.

An event group eg is a set of events. Let vector Z_{eg} be a bit vector which has its $i-th$ bit set iff an event $e \in eg$ is taking place at time i. This is obtained by applying *or* to all of the appropriate Y_e vectors.

This constraint specifies event groups to which it applies, as well as a number of time groups (sets of times) and for each such time group the minimum and maximum number of starting times an event must have within times of that time group. Let TG_{spec} denote this set of sets of times and let $mask_{tg}$ be the bit vector which has its $i-th$ bit set iff i is a time slot of time group tg:

There must be at least d_i^{min} starting times within the given time groups (min is a subscript, not exponentiation):

$$\bigwedge_{\substack{\forall tg_i \in TG_{spec} \\ eg \in EG_{spec}}} (atLeast\_d_i^{min}[and(Z_{eg}, mask_{tg})]) \qquad (20)$$

A similar encoding to the one above is also used, but with $atMost\_d^{max}$.

If this constraint is used as a soft constraint, the soft cardinality constraint is used instead. Points of application are event groups (not events) and deviations are calculated as the number by which the events group falls short of the minimum or exceeds the maximum.

3.3.6 Distribute Split Events Constraint

This constraint specifies the minimal and maximum number of starting times of a specified duration. For example, if $duration(e) = 10$, we may impose that the lecture should be split so that at least two starting times must have duration three. Formally:

There must be at least d_{min} starting times with given duration d:

$$\bigwedge_{\forall e \in E_{spec}} (atLeast\_d_{min}[K_{e,d}] \wedge atMost\_d_{max}[K_{e,d}]) \qquad (21)$$

3.3.7 Limit Busy Times Constraints

This constraints imposes limits on the number of times a resource can become busy within certain a time group, if the resource is busy at all during that time group. For example, if a teacher teaches on Monday, he or she must teach at least for three hours. This is useful in preventing situations in which teachers or students would need to come to school for only to have a lesson or two.

A resource is busy at a time group tg iff it is busy in at least one of the time slots of the tg. Let TG_{spec} denote this set of sets of times:

$$\bigwedge_{\substack{\forall r \in R_{spec} \\ \forall tg \in TG_{spec}}} (or(atLeast\_b_{min}[and(Y_e, mask_{tg})], (= (and(Y_e, mask_{tg})), 0))) \quad (22)$$

A similar encoding to the one above is also used, but with $atMost\_b_{max}$. Note that in this case *or* represents *logical or*, rather than *bitvectoror*.

If this constraint is used as a soft constraint, the soft cardinality constraint is used instead, although special care must be given as this is a conditional cardinality constraint: if the calculated vector is different from zero then the cardinality constraints need to be fulfilled. Points of application are resources and each resource generates multiple deviations (one for each time group) which calculated as the number by which the events group falls short of the minimum or exceeds the maximum.

3.3.8 Limit Idle Times Constraints

This constraint specifies the minimal and maximum number of times in which a resource can be idle during the times in the specified time groups. For example, a typical constraint is to impose that teachers must not have any idle times.

A time slot t is idle with respect to time group tg (set of times) if it is not busy at time t, but is busy at an early time and at a later time of the time group tg. For example, if a teacher teaches classes Wednesdays at $Wed2$ and $Wed5$, he or she is idle at $Wed3$ and $Wed4$, but is not idle at $Wed1$ and $Wed6$. This constraint places limits on the number of idle times for each resource. Let vector $G_{e,tg}$ be the vector obtained by taking *or* of bitvectors $and(and(Y_e, mask_{tg}), rshift((Y_e, mask_{tg}), k))$ where $k = 1..n$ and n is the number of times in time group tg. Let vector $H_{e,tg}$ be similar, except using $lshift$ instead of $rshift$. We then encode the constraint as follows:

There must be at least $idle_{min}$ idle times during a time group:

$$\bigwedge_{\substack{\forall tg \in TG_{spec} \\ r \in R_{spec}}} (atLeast\_idle_{min}[and(inv(Y_e), and(H_{e,tg}, G_{e,tg}))]) \tag{23}$$

A similar encoding to the one above is also used, but with $atMost\_idle_{max}$. If this constraint is used as a soft constraint, the soft cardinality constraint is used instead.

3.3.9 Cluster Busy Times Constraints

This constraint specifies the minimal and maximum number of specified time groups in which a specified resource can be busy. For example, we may specify that a teacher must fulfill all of his or her duties in at most three days of the week.

We first define a helper bitvector B_r for each resource, in which $i-th$ bit is set iff the resource is busy at the $i-th$ time group. Therefore, $i-th$ bit in B_r is equal to the *or* operation on all of the $i-th$ bits of bitvectors in E(r). With this helper bitvector, we may now encode the constraint as:

There must be at least b_{tg}^{min} busy time groups:

$$\bigwedge_{\forall r \in R_{spec}} (atLeast\_b_{tg}^{min}[B_r]) \tag{24}$$

A similar encoding to the one above is also used, but with $atMost\_b_{max}$. If this constraint is used as a soft constraint, the soft cardinality constraint is used instead.

3.3.10 Prefer Times Constraints

This constraint specified that certain events should be held at certain times. If an optional parameter d is given, then this constraint only applies to subevents assigned duration d. For example, a lesson of *duration 2* must be scheduled on Monday, excluding the last time slot on Monday.

Let P_e be the bitvector in which $i-th$ bit is set iff i is a preferred time. We then encode:

$$\bigwedge_{\forall e \in E_{spec}} (atMost\_0[and(\star, inv(P_e))]) \tag{25}$$

where $\star$ is either Y_e or $K_{e,d}$, depending on whether the optional parameter d is given. Note that this constraint is not the same when the optional parameter is not given and when $d = 1$.

3.3.11 Order Events Constraints

This constraint specifies that one event can start only after another one has finished. In addition to this, parameters B_{min} and B_{max} are given which define the minimum and maximal separations between the two events and are by default set to zero and the number of time slots, respectively. The constraint specifies a set of pairs of events to which it applies.

If the first event in a pair is taking place at time t, then the second event cannot take place at time $t + B_{min}$ nor at any previous times:

$$\bigwedge_{\forall (e_1,e_2) \in E^2_{spec}} (< (lshift(e_i, B_{min}), e_j)) \tag{26}$$

A similar encoding to the one above is also used, but with $>$ and B_{max}. Special care must be taken as overflows may happen during the shift operations.

3.3.12 Link Events Constraints

Certain events must be held at the same time. For example, physical education lessons for all classes of the same year must be held together. This constraint specifies a certain number of event groups and imposes that all events within an event group must be held simultaneously. Let EG_{spec} denote this set of sets of events. All events within an event group must be held at the same times:

$$\bigwedge_{\substack{\forall eg \in EG_{spec} \\ e_j \in eg}} (atMost\_0[and(Z_{eg}, inv(Y_{e_j}))]) \tag{27}$$

If the constraint is declared a soft one, the soft cardinality constraint is used instead. Points of application are event groups (not events) and deviations are calculated as the number of times in which at least one of the events within the event group is taking place, but not all of them.

3.3.13 Extending the Model

As mentioned in the beginning, we made the assumption that all resources have been already assigned to events, as it is easier to model, implement and present the formulation. This is a reasonable assumption, as most instances are of this form. Still, a significant part of the instances require assignments of resource to events. Our model is easy to extend with these requirements by introducing new bitvectors: for each event e and resource r, a bitvector is created in which $i-th$ bit is set iff resource r has been assigned to event e at time i. With these bitvectors, the other resource assigning constraints (we direct interested readers to [14]) can be encoded in a similar fashion as the ones already presented, along with certain modifications need to be made to Assign Time and Avoid Clash constraints.

However, special care needs to be given when doing so to concrete instances, as requirements for resource assignments can be diverse. For example, in instance $SpainInstance$ given in the ITC repository, assignments consist of assigning one gym room out of two available. For instance $EnglashStPaul$, room need to be assigned and many symmetries appear because all rooms are identical. Hence, it might be a better idea to restrict the number of events at each time to the number of rooms, rather than assigning rooms directly to events.

In addition, it may be of interest to simplify the $K_{e,d}$ and S_e encodings which would simply state that if an event has three consecutive bits set it is treated as a subevent of duration 3 rather than of the complicated formulation given or that only the first constraint regarding S_e should be used. The reason the encoding is so complicated is because of the way

the general formulation specifies starting times, but this is not necessary for all instances.

3.3.14 Other Theories

We have done some initial experiments with linear integer arithmetic theories in two different formulations. One of the formulations had variables which were restricted to binary values and the encodings were similar to a pure SAT formulation, except that the cardinality constraints could be encoded more elegantly. The second encoding took more advantage of the integer arithmetic in which for each event we create a number of variables equal to its duration. The value of the variable determines which time slot the event takes place. This reduced the number of variables significantly when compared to the binary version, but some constraints were harder to encode. However, regardless of that, both modeling options failed to produce any solutions to problem instances, even when only Assign Time, Avoid Clashes and Prefer Times constraints were used. Therefore, we did not continue with these modelings and continued with the bitvector formulation, which performed better in these initial experiments.

4 Computational Results

In our current experiments we evaluated our approach on some benchmark instances from HSTT which can be found on the repository of the International Timetabling Competition 2011 [1]. A subset of instances which were suggested by the competition as test beds, as well as the ones used in the competition have been chosen (these two sets intersect). All tests were performed on (Intel Core i3-2120 CPU @ 3.30GHz with 4 GB RAM) and each instance was given a single core. We restricted the computational time to 24 hours per instance.

In the instances, the number of time slots ranges from 25 to 142, number of resources from 8 to 99, number of events from 21 to about 350 (exception is the Italy4 instance with

748 events) with total event duration from 75 to around 1000. These numbers are approximations and vary heavily from instance to instance. We do not provide detailed information, but direct the interested reader to [12] [1] [14].

In the tables below, we denote the objective function cost by (x, y), where x is the infeasibility value and y is the objective value. If a dash is used, that means that the solver failed to produce a solution. If an x is used, that means that we had not provided an objective value, as in initial solution has been generate in which only hard constraints had been considered and the resulting objective value is essentially random.

We experimented with the SMT solver Yices 1.0.40 (released December 4, 2013) [6]. It was chosen because it is the only SMT solver to our knowledge which directly supports optimization, rather than just checking for satisfiability.

4.1 Comparisons of Results

We compare results we had obtained with our approach and GOAL (the winning team of the competition). GOAL's algorithm first generates an initial solution using KHE [10] and then performs its heuristic search algorithm. Note that the initial solution generated can be unfeasible and in some cases the algorithm fails to improve this solution to a feasible one.

In the table below we present the computational results. To make our comparison fair, we ran our approach and GOAL on the same computer platform and each solver was restricted to 24 hours and was given one core. The source code of GOAL was provided by their authors [4]. The time to convert an instance from xHSTT to a SMT instance is negligible (a few seconds at most) when compared to the SMT solution process.

As we see in the table, we provide experimental results for 11 instances. Other instances were not included because the current implementation does not support them. The reasons for this are either that we did not yet implement constraints which allow resource assignments (e.g. Assign Resource Constraints), use the square of sums (we currently do not have a

model for this cost function) or the sum of squares cost function (we have a model for this cost function but is not yet implemented).

The abbreviations used in the columns are as follows: OA is our approach, GOAL is the winning team algorithm:

Name	OA	GOAL
BrazilianInstance1	**(0, 47)**	(0, 54)
BrazilianInstance2	**(0, 60)**	(1, 42)
BrazilianInstance4	**(0, x)**	(16, 95)
BrazilianInstance5	**(0, x)**	(4, 121)
BrazilianInstance6	**(0, x)**	(4, 195)
BrazilianInstance7	**(0, x)**	(11, 230)
SouthAfricaLewitt2009	(0, x)	**(0, 18)**
SouthAfricaWoodlands	(-, -)	**(2, 13)**
GreeceHighSchool	**(0, 0)**	**(0, 0)**
ItalyInstance1	(-, -)	**(0, 19)**
ItalyInstance4	(-, -)	**(0, 57)**

Table 1 Results obtained after 24 hours.

There might be differences in the results obtained by GOAL in the competition and obtained by our 24 hour runs, because in the competition competitors in the final phase were given one month to use whatever available resources to provide the best results. We focus here on the comparison with the winner of ITC competition, because we think that this gives a good idea how good our approach performs in a limited amount of time compare to one of best existing approaches for this problem. For some of the instances, better upper bounds were obtained after the competition by GOAL and other approaches without time or resource limitations.

It is interesting to note that Yices found an initial solution for all instance except three quickly (within 10 minutes for all but SouthAfricaLewitt which took several hours), but had not managed to perform any optimization for most instances within the given time limit. Even so, as we can see from Table 1, from the examples in which our encoding was

done successfully, our approach could find feasible solutions in which GOAL could not in seven instances within the given time limit. A feasible solution has been found for all except for three instances. In the case of the Italy4 instance, the program ran out of memory.

Overall, we conclude that the SMT approach can provide feasible solutions in short time for several instances. Further research is needed to successfully apply this technique for the optimization variant. In general, it seems that SMT strengths are in satisfiabiliy rather than optimization, while GOAL could be used for optimizing solutions which are (near) feasibility.

5 Conclusion

High school timetabling is a wide spread and important problem and because of this, developing algorithms to solve the problem are of great importance.

In this paper, we have shown that the general HSTT problem [14] can indeed be modeled as a SMT problem, despite the generality of the specification, with the exception of not being able to model the square of sums of deviation cost function. We presented a complete and detailed encoding using theory of bitvectors in the general sense as required by the specification under the assumption that resources had been preassigned to events, but have sketched how the model can be extended and discussed some important special cases.

We implemented and evaluated our approach on a subset of benchmark instances suggested and used by the Third International Timetabling Competition 2011 and compared our results with GOAL, the winning team of the Third International Timetabling Competition 2011. For some of the tested instance, our approach managed to find feasible solutions within a given time limit and there is space for further improvements. Generated encodings solve practical problems and as such can be used as benchmarks for the evaluation of SMT solvers.

Furthermore, we plan on to investigate hybridization of our approach with heuristic techniques (e.g. develop a large neighborhood search algorithm) that will utilize SMT.

Acknowledgements The work was supported by the Vienna PhD School of Informatics and the Austrian Science Fund (FWF): P24814-N23.

References

1. International timetabling competition 2011. http://www.utwente.nl/ctit/hstt/itc2011/welcome/. Accessed: 2014-1-30
2. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: Version 2.0. In: Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England), vol. 13, p. 14 (2010)
3. Bofill, M., Suy, J., Villaret, M.: A system for solving constraint satisfaction problems with SMT. In: Theory and Applications of Satisfiability Testing–SAT 2010, pp. 300–305. Springer (2010)
4. Brito, S.S., Fonseca, G.H.G., Toffolo, T.A.M., Santos, H.G., Souza, M.J.F.: A SA-VNS approach for the high school timetabling problem. Electronic Notes in Discrete Mathematics **39**, 169–176 (2012)
5. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer (2008)
6. Dutertre, B., De Moura, L.: The Yices SMT solver. Tool paper at http://yices. csl. sri. com/tool-paper. pdf **2**, 2 (2006)
7. Even, S., Itai, A., Shamir, A.: On the complexity of time table and multi-commodity flow problems. In: Foundations of Computer Science, 1975., 16th Annual Symposium on, pp. 184–193. IEEE (1975)
8. Fonseca G. H. G., S.H.G.T.T.A.M.B.S.S.S.M.J.F.: A SA-ILS approach for the high school timetabling problem. In: In Proceedings of the ninth international conference on the practice and theory of automated timetabling, PATAT (2012)
9. Kheiri, A., Ozcan, E., Parkes, A.J.: HySST: hyper-heuristic search strategies and timetabling. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012) (2012)
10. Kingston, J.H.: The KHE high school timetabling engine (2010)
11. Nieuwenhuis, R., Oliveras, A.: On SAT modulo theories and optimization problems. In: Theory and Applications of Satisfiability Testing-SAT 2006, pp. 156–169. Springer (2006)
12. Post, G., Ahmadi, S., Daskalaki, S., Kingston, J., Kyngas, J., Nurmi, C., Ranson, D.: An XML format for benchmarks in high school timetabling. Annals of Operations Research **194**(1), 385–397 (2012). DOI 10.1007/s10479-010-0699-9. URL http://dx.doi.org/10.1007/s10479-010-0699-9
13. Post, G., Di Gaspero, L., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. Annals of Operations Research pp. 1–7 (2012)
14. Post, G., Kingston, J.H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., et al.: XHSTT: an XML archive for high school timetabling problems in different countries. Annals of Operations Research pp. 1–7 (2011)
15. Sørensen, M., Dahms, F.H.: A two-stage decomposition of high school timetabling applied to cases in Denmark. Computers & Operations Research **43**, 36–49 (2014)
16. Sørensen, M., Kristiansen, S., Stidsen, T.R.: International timetabling competition 2011: An adaptive large neighborhood search algorithm. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), p. 489 (2012)
17. Sørensen, M., Stidsen, T.R.: Comparing solution approaches for a complete model of high school timetabling. Tech. rep., DTU Management Engineering (2013)