Modeling High School Timetabling as Partial Weighted maxSAT

Emir Demirović, Nysret Musliu

Vienna University of Technology
Database and Artificial Intelliegence Group
{demirovic,musliu}@dbai.tuwien.ac.at

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 SAT, in which all constraints are treated as hard constraints. In order to take into account soft constraints, we extend the previous model to Partial Weighted maxSAT. In addition, we present experimental results and compare to other approaches, using both artificial and real-world instances, all of which were taken from the Third International Timetabling Competition 2011 benchmark repository. Our approach gives competitive results and in some cases outperforms the winning algorithm from the competition.

Keywords: High school timetabling, maxSAT, SAT encodings

1 Introduction

In this paper, we describe a modeling of the high school timetabling problem (HSTT) as a maximum propositional satisfiability problem (maxSAT). By doing so, we were able to find solutions for many instances which were proposed by the Internation Timetabling Competition 2011 [13], and in some cases managed to outperform the winning algorithm of the competition, GOAL.

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 as such 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 [6]. 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 researchers have 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 Iternational 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 [7]. Lectio uses an Adaptive Large Neighborhood Search [17], while HySST uses a Hyper-Heuristic Search [8]. Recently, [18] used Integer Programming (IP) in a Large Neighborhood Search algorithm and [16] introduced a two phase IP algorithm for a different timetabling problem, but have managed to adjust the method for a number of HSTT instances. All the best algorithms on the competition were heuristic algorithms and this is why introducing a new exact method (our approach) is important. Some advantages would include 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 maxSAT 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 maxSAT. There is a natural connection between timetabling and SAT. HSTT as itself has many logic based characteristics and as such some of its constraints can easily be encoded as SAT. This has motivated us to investigate how efficient can a SAT 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 Weighted Partial maxSAT 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 investigate empirically the performance 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 and the results show that our approach outperforms it in some cases.

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 maxSAT 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. 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. As for some examples, 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 do not describe constraints in this section, but will rather do so in the next section when we present the SAT 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, we have two parts: infeasibility value and objective value part. 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 as maxSAT

3.1 Cardinality Constraints

Cardinality constraints impose limits on the truth values assign to a set of literals. These are $atLeast\_k[x_i : x_i \in X]$, $atMost\_k[x_i : x_i \in X]$ and $exactly\_k[x_i, x_i \in X]$, which constraint that at least, at most or exactly k literals out of the specified ones must or may be assigned true. Many different encodings of these constraints exist (e.g. see [15]) and here we will describe only the ones used in our implementation.

Basic Encoding One way to encode the cardinality constraints is to simply enumerate all legal assignments. We refer to this as the *basic encoding*. Since the number of clauses grows exponentially, we use these encoding if $|X| < 50$ and $k < 4$. Otherwise, the bit adder encoding would be used. These values were determined ad hoc for the instances at hand.

Bit Adders The idea is to regard each variable as a 1-bit number, take the sum of all the chosen variables by using a series of adders which sum a binary number and a 1-bit number. The end result is a binary representation of the number of variables set. Appropriate clauses would then be created to forbid specified outputs. The number of clauses and auxiliary variables is $O(nlog(n))$.

Unary Representation We present an $exactly\_k$ encoding which was used in our modeling. The unary number representation as given in [4] is:

$$\bigwedge_{\forall i \in [1,n-1]} (u_i \Rightarrow u_{i-1}). \tag{1}$$

The interpretation is that the value assigned with this representation is equal to i, where i is the largest number such that $u_i = \top$. For the case $k = 1$, we create a variable that can take values from $[0, n]$ and do so in the unary representation as described above and add the following constraints:

$$\bigwedge_{\forall i \in [1,n-1]} (u_i \wedge \overline{u}_{i+1} \Leftrightarrow x_i) \wedge u_0 \tag{2}$$

For the case when $k > 1$, we encode k constraints of the form $exactly\_1$ and let $u_{i,j}$ represent the $j - th$ auxiliary variable used in the $i - th$ constraint. We then complete the encoding:

$$\bigwedge_{\substack{\forall i \in [0,k-2] \\ \forall j \in [0,n]}} (u_{i,j} \Rightarrow u_{i+1,j+1}) \wedge \bigwedge_{\substack{\forall i \in [0,k-2] \\ \forall j \in [k-i+1,n]}} (\overline{u}_{i,j}) \tag{3}$$

3.2 Soft Cardinality Constraints

soft cardinality constraints are similar to the previous ones, except that penalize violations of the constraint rather than forbidding it.

Basic Encoding We presented the encoding for the soft cardinality constraint $atLeast\_k[x_i : x_i \in X]$, while $atMost\_k[x_i : x_i \in X]$ is done in a similar fashion:

$$\bigwedge_{j \in P} (A_j \rightarrow atLeast\_j[x_i : x_i \in X]) \wedge \bigwedge_{j \in P} (w(j)(A_j)) \tag{4}$$

Where A_i are new auxiliary variable, P is a set of integers in the interval $[1, k]$ and w(j) is a weight function which depends on j, while the $atLeast$ in encoded by a basic encoding. The second equation is a series of soft unit clauses containing A_j and its weights are $w(j)$.

Bit Adders We use bit adder encoding described previously, but instead of forbidding certain outputs, we penalizes their assignments. It is important to note that the weights may be assigned to each undesired output completely independently, unlike in the basic encoding.

Special Cases A very important special case for $atLeast\_k[x_i : x_i \in X]$ is when $k = |X|$ (a similar case for $atMost\_k[x_i : x_i \in X]$ occurs when $k = 0$) and the weight function $w(j)$ is of the form $w(j) = c * j$, where c is some constraint. In this case, instead of using the previously described encoding, we encode the following soft clauses:

$$\bigwedge_{x_i \in X} ((w)(x_i)) \tag{5}$$

3.3 Constraints

In practice, some constraints are never used as soft constraint. Because of this, we only give the encodings for soft constraint where it is appropriate in order to avoid unnecessary technicalities.

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 (e.g. a cost function may simply be the sum of all deviations). Our approach supports cost functions of sums of (squares of) deviations. The HSTT specification allows for other cost functions as well, but we do not have an encoding for them currently. Fortunately, only two instances use nonsupported cost functions.

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.

In addition, E, T and R are sets of events, times and resources, respectively. Each constraint is applied to some subset of those three and within the text are notated as E_{spec}, T_{spec} and R_{spec}. These subsets are 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.

Due to space limitations, we present the encoding of only a subset of the constraints used in HSTT and provide a detailed longer version of this paper online (dbai.tuwien.ac.at/user/demir/sat2014long.pdf).

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.

We define decision variables $Y_{e,t}$ and other constraints rely on them heavily. For each $e \in E$ and $t \in T$, variable $Y_{e,t}$ indicates whether event e is taking place at time t. Each event must take place for a number of times equal to its duration d:

$$\bigwedge_{\forall e \in E} (exactly\_d[Y_{e,t} : t \in T]) \tag{6}$$

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.

In the formal specification of HSTT, there are no rules on what can be defined as a starting point. One would 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. Other constraints give more control over these kind of assignments.

For each event e, variable $S_{e,t}$ indicates whether event e has started taking place at time t. For example, if event e had *duration(e)=2* and its corresponding $Y_{e,t}$ were assigned at time slots t and $t+1$, then $S_{e,t} = true$, $S_{e,(t+1)} = false$. Formalities that are tied to starting times with regard to the specification are expressed as follows:

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

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ t \in T}} (Y_{e,t} \wedge \overline{Y}_{e,(t-1)} \Rightarrow S_{e,t}) \tag{7}$$

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

$$\bigwedge_{\substack{\forall e \in E \\ t \in T}} (S_{e,t} \Rightarrow Y_{e,t}) \tag{8}$$

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

$$\bigwedge_{\forall e \in E_{spec}} (atLeast\_A_{min}[S_{e,t} : t \in T] \wedge atMost\_A_{max}[S_{e,t} : t \in T]) \tag{9}$$

In addition, this constraint also imposes the minimum d_{min} and maximum d_{max} duration for each subevent. For each specified event $e \in E_{spec}$ and duration d, variable $K_{e,t,d}$ indicates event e has a starting time at time t of duration d. Formally:

If time t has been set as a starting time, associate a duration with it:

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ \forall t \in T}} (S_{e,t} \Rightarrow \bigvee_{d_{min} \leq d \leq d_{max}} K_{e,t,d}) \tag{10}$$

Remark: We could had encoded that exactly one of the right hand sides variables must be chosen, but this is handled in the later parts of this encoding.

When $K_{e,t,d}$ is set, the event in question must take place during this specified time (where set D is the set of integers from the interval $[d_{min}, d_{max}]$):

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ \forall t \in T \\ d \in D}} K_{e,t,d} \Rightarrow \bigwedge_{i \in [0,d-1]} Y_{e,(t+i)} \tag{11}$$

If a duration has been specified for time t, make sure that other appropriate $K_{e,t,d}$ variables must be false:

$$\bigwedge_{\substack{\forall e \in E_{spec} \\ \forall t \in T \\ d \in D}} (K_{e,t,d} \Rightarrow \bigwedge_{d_{min} \leq g \leq d_{max}} \bigwedge_{i \in [0,d-1] \wedge (i \neq 0 \vee g \neq d)} \overline{K}_{e,t+i,g}) \tag{12}$$

If an subevent of duration d has been assigned and immediately after the event is still taking place, then assign that time as a starting time:

$$K_{e,t,d} \wedge Y_{e,t+d} \Rightarrow S_{e,t+d} \tag{13}$$

Assign Resource Constraints Each event requires a certain amount of resources in order to be scheduled. These resources can be teachers, classes, rooms, etc. For example, in order for a math lesson to take place a math teacher, a room and a projector are needed. It might also be the case that two teachers are needed, e.g. one lecturer and one as an assistant. This has been implemented into the general HSTT specification as follows:

Each event has a number of *roles*. To each of these *roles* exactly one resource of a specific resource type must be assigned. The *role* names within an event must be unique, but different events may have the same *roles* requiring different types of resources. For example, a resource might require the following roles with the appropriate resource types given in parenthesis: 'Teacher' (teacher), 'Assistant' (teacher), 'Class' (class), 'Seminar room' (room). This constraint merely requires that a resource of a given type must be assigned. For the given *role*, a variable $M^{role}_{e,t,r}$ is created, which indicates whether event e at time t is using resource r to fulfill the given *role*. The constraint is encoded as follows:

If an event is taking place, it's specified *role* must be fulfilled:

$$\bigwedge_{\substack{e \in E_{spec} \\ t \in T}} (Y_{e,t} \rightarrow exactly\_1[M^{role}_{e,t,r} : r \in R_{spec\_resource\_type}]) \tag{14}$$

If a resource has been chosen to fulfill an event's role as some time, mark that resource as used by the event at that time:

$$\bigwedge_{\substack{e \in E_{spec} \\ t \in T \\ r \in R_{spec\_resource\_type}}} (M^{role}_{e,t,r} \rightarrow Y_{e,t,r}) \tag{15}$$

The previous two encodings hold individually for each Assign Resource Constraint. The next encoding is done after all constraint of type Assign Resource Constraints and is in a sense a global constraint:

A resource may fulfill at most one role at any given time:

$$\bigwedge_{\substack{e \in E_{spec} \\ t \in T \\ r \in R}} (atMost\_1[M^{role}_{e,t,r} : role \in ARC_{roles}]) \tag{16}$$

Avoid Split Assignments Constraint Events are frequently broken down into subevents, each of which has the same resource requirements. This constraint imposes that for the specified *role*, only one resource should be used across all of the subevents. For example, a lecture should always take place in the same lecture room, regardless of which room is chosen. This constraint applies to the specified *role* and to a specified resource type. We create auxiliary variables $V^{role}_{e,r}$ which indicate whether an event e is using a resource r to fulfill its *role* at some point in time:

$$\bigwedge_{\substack{e \in E_{spec} \\ R_{spec\_resource\_type} \\ t \in T}} (M^{role}_{e,t,r} \rightarrow V^{role}_{e,r}) \tag{17}$$

The constraint is now encoded as:

$$\bigwedge_{e \in E_{spec}} (atMost\_0[V^{role}_{e,r} : r \in R_{spec\_resource\_type}]) \tag{18}$$

If this constraint is used as a soft constraint, the soft cardinality constraint is used instead.

Special Cases In this section we look into important special cases which may simplify the encodings significantly.

If an Assign Resource Constraint is given and all of the resources it references behave the same, then instead of encoding assign resource and avoid clashes constraints for those resource, we may use the following encoding:

$$\bigwedge_{t \in T} (atMost\_hY_{e,t} : e \in E_{spec}) \tag{19}$$

Where E_{spec} are events that require the mentioned resources and h is number of resources of the described kind. This case arises in EnglandStPaul and FinlandArtificialSchool instance and without this case and another case described further would not have been possible to encode within reasonable amounts of memory.

If there is only one *role* per resource type specified in the requirements of an event, then the encoding of the auxiliary variables in ARC may be avoided.

If the resources specified in Assign Resource Constraints are not subjected to Limit Idle Constraints and assigning more than one resource to an event may be legal, then a simpler encoding may be used for ARC, in which $atLeast\_1$ is used instead of using $exactly\_1$. This case happens typically in instances which require the assignment of rooms. If two rooms are assigned to an event, in the solution we would simply pick only one. However, this cannot be applied in general e.g with teachers.

Another problem with ARC is that certain symmetries may arise, increasing the solution time. For example, if we have two ARCs, each with their specified *role*. If these two roles both use the same resource type and no further constraints are imposed on these resources, then we may swap their assignments of resources and still get the same un(feasible) solution, which is undesirable. Therefore, encoding a sorting is very useful and can be done efficiently since the unary representation is used.

In some cases, by knowing the semantics of each constraint, simpler encoding can be produced. This is encountered in SpainInstance in which a large amount of Spread Events Constraints are encoded which are the state that lessons can have at most one starting point in two consecutive days. However, this is not trivial to specify in the general HSTT specification and will produce a large number of clauses, which could be avoid if a special encoding for much a constraint is encoded.

Another interesting case is the encodings of $K_{e,t,d}$. These are created in order to comply with the formal specification of HSTT. In some cases, it suffices to encode $K_{e,t,d}$ as (i is an integer):

$$K_{e,t,d} \leftrightarrow (\bigwedge_{i \in [0..d-1]} Y_{e,t+i}) \wedge \overline{Y}_{e,t+d} \tag{20}$$

This encoding is much more desirable when it is possible and we can use it e.g. in the ItalyInstance1, where the general encoding took around 50 hours to compute the optimal solution, while with the change shown above took around 10 hours. If the encoding is used, other constraints might be affected, such as Split Events Constraint and need to be changed appropriately. However, in our current implementation this situation needs to be done by hand.

4 Computational Results

We had conducted experimental evaluations on benchmark instances from HSTT which can be found on the repository of the International Timetabling Competition 2011 [2]. 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. We made our generated maxSAT instances available online (dbai.tuwien.ac.at/user/demir/xHSTTtoSAT_instances.tar.gz).

In the instances, number of time slots range from 25 to 60, number of resources from 20 to 120, number of events from 200 to 1000 with total event duration from 300 to 1500. These numbers are approximations and vary heavily from instance to instance. Due to space limitation, we do not provide detailed information, but direct the interested reader to [12] [2] [14].

In the tables below, we shall note the objective function cost as (x, y), where x is the infeasibility value and y is the objective value.

We experimented with different maxSAT solvers in order to solve encodings obtained by our approach. We considered the following maxSAT solvers, all of which had been used in the recent maxSAT Competition [1]: sat4j-maxsat [10], maxsatz [11] and optimax. After preliminary experiments, which consisted of solving three smaller instances (each instance given 4 hours), sat4j performed the best and was chosen as the solver we would use for the longer experimental results. In the next table we give the results ("oom" stands for "out of memory", "to" stands for "timeout" with no solution provided):

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 [9] and then performs its heuristic search algorithm. The initial solution generated can be unfeasible and in some cases the algorithm fails to

Name	sat4j-maxsat	maxsatz	optimax
BrazilianInstance1	(0, 39)	(0, 196)	"to"
ItalyInstance1	(0, 58)	"to"	(0, 1118)
SouthAfricaLewitt2009	(0, 17)	"oom"	(0, 0)

Table 1. Preliminary results with maxSAT solvers on small instances.

improve this solution to a feasible one. Because of this we had investigated the hybridization of our approach and GOAL, in which our method provides the initial feasible solution which GOAL then optimizes. The initial solution generation is typically done very quickly in most cases, ranging from a few seconds to a few minutes with the only exception being FinlandArtificialInstance which took 30 minutes.

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 [5]. Encoding of instances times are negligible compared to the maxSAT solution process. Three Denmark, one Spanish and four Netherlands instances could not be solved by our approach and were not included in the table, but are discussed later on. Values marked with an asterisk * are known to be optimal (further information on how the optimality proof was obtained can be found on ITC's web page on individual instances [2]). The abbreviations used in the columns are as follows: OA is our approach, GOAL is the winning team algorithm, OA+GOAL is the previously described method in which we generate an initial solution with our approach which is then improved further with GOAL:

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 an idea of how good our approach performs in a limited amount of time compared 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.

As we can see from Table 2, from the examples in which our encoding was done successfully, our approach outperformed GOAL in 12 instances and in five cases was the best method. Three of the instance had been solved to optimality, namely BrazilianInstance1 and African instances. For the others, part to the success on the Brazilian instances can be attributed to the fact that our approach quickly finds a feasible solution, while GOAL in these cases does not and its heuristics could not escape infeasibility. Even though our encoding cannot capture the square of sums cost function for soft constraints of EnglandStPaul and

Name	OA	GOAL	$OA + GOAL$
BrazilianInstance1	**(0, 38)***	(0, 54)	(0, 48)
BrazilianInstance2	**(0, 32)**	(1, 42)	(0, 37)
BrazilianInstance4	(0, 205)	(16, 95)	**(0, 142)**
BrazilianInstance5	(0, 117)	(4, 121)	**(0, 106)**
BrazilianInstance6	(0, 230)	(4, 195)	**(0, 171)**
BrazilianInstance7	(0, 400)	(11, 230)	**(0, 210)**
AfricaLewitt2009	**(0, 0)***	(0, 18)	(0, 470)
AfricaWoodlands	**(0, 0)***	(2, 13)	(0, 71)
FinlandCollege	(0, 1523)	(1, 5)	**(0, 14)**
FinlandHighSchool	(0, 289)	**(0, 14)**	(0, 15)
FinlandSecondarySchool	(0, 252)	**(0, 83)**	(0, 85)
FinlandArtificialSchool	(0, 47)	(3, 6)	**(0, 12)**
GreecePatras2010	(0, 331)	**(0, 0)***	**(0, 0)***
GreeceWesternUniversity4	(0, 121)	(0, 5)	**(0, 4)**
GreeceHighSchool	**(0, 0)***	**(0, 0)***	**(0, 0)***
KosovaInstance	(0, x)	(0, 5)	(0, 1059)
EnglandStPaul	(0, x)	(3, 48)	**(0, 138)**
ItalyInstance1	(0, 17)	(0, 19)	**(0, 13)**
ItalyInstance4	(0, 12825)	**(0, 57)**	(0, 59)

Table 2. Results obtained after 24 hours.

KosovaInstance, it does not affect our findings of feasible solutions, which gives immediately a better solution than the unfeasible solution for EnglandStPaul of GOAL. For these instances we did not provide the objective function obtained, since only hard constraints had been considered and the resulting objective value is essentially random. In 4 cases, our approach performs better than both other approaches.

GOAL outperformed our method in six instances, in five cases manages to exceed the results of the combination of the two approaches ($OA+GOAL$) and in four instances was the best method. However, in the cases where GOAL outperformed the combined approach (FinlandHighSchool, FinlandSeconarySchool, ItalyInstance4), the difference in results was marginal. In the two cases were the difference is significant (KosovaInstance and AfricaLewitt2009), an interesting situation arises. GOAL takes an initial solution provided by our method, but it heavily fails to escape the local optima, even though much better results are known to exist. This leads us to the conclusion that GOAL may be heavily influenced by the initial solution it gets before starting to optimize. This can help us

by further examining these situation in order to learn what exactly leads GOAL to fail to escape local optima and possibly improve the algorithm.

When we use the combination of our approach and GOAL, it manages to outperform GOAL in 11 cases, 12 cases our method and outperformes both methods in 8 cases. However, for GreeceWesternUniversity4, the improvement over GOAL is marginal, similar to the situation before. In cases where it does perform better than GOAL, GOAL could not find a feasible by its own and our initial feasible solution was of benefit. For two cases, it got quickly stuck in local optima, as explained previously.

In a previous iteration of our approach, we used a less general encoding which was still sufficient for ItalyInstance1 (see the notes in the previous section regarding $S_{e,t}$ and $K_{e,t,d}$) and have calculated the optimal solution of the instance (objective value 12) in around 10 hours. However, this is not automatized and must be done by hand currently, but it is an interesting way to point out that the more general the encoding is, the harder it is to solve, since the layers of abstraction add additional complexity. With the general encoding, it took around 50 hours to obtain the optimal solution in contrast to the previous 10 hours.

We note that Australian instances were not included in Table 2 because two of them were proven to be unsatisfiable by our approach, while the third (SAHS96) was not encoded due to implementation reasons. Because of the proof of infeasibility, a lower bound on the infeasibility value can be derived for TES99, as a solution with unfeasibility value 1 is known.

We now discuss the cases in which an encoding has not been generated.

KosovaInstance and EnglandStPaul contain the cost function SquareSum for its soft constraints, which is currently not supported, as discussed in the previous section. However, this does not affect the generation of a feasible solution, which was done successfully in both cases.

The Spanish instance has been encoded, but the resulting maxSAT file was around 2 GB large, which was too large for our computers with 32-bit operating systems. After closer examination, we concluded that this was dominated by Spread Event Constraints, because of the way the constraints and its accompanying time groups had been given in the specification. A drastic reduction of variables and clauses could had be gained if the encoding of the special case regarding these constraints had been implemented, as discussed in the previous section.

Instances from Netherlands and Denmark had not been solved, as the current implementation for the maxSAT encoder runs out of memory due to inefficient memory management (the number of resources are unexpectedly high: over 800, while other instances typically range from 20 to 120 resources). Regardless of this, in general AssignResourceConstraints and related constraints seem to add more complexity than other constraints.

Overall, we conclude that our approach is competitive. Our approach can be used to provide competitive solutions when compared to GOAL and in the case of smaller instances optimal solution can be calculated. Larger instances can be problematic, but combining our method with GOAL provides satisfac-

tory results. The results obtained in the cases were encodings could have been constructed are promising and we believe that further research in this direction will prove to be fruitful.

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 weighted partial maxSAT problem, despite the generality of the specification. We presented a complete and detailed encoding in the general sense as required by the specification, but also presented several alternative encodings for special cases.

We implemented and evaluated our approach on 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. Our approach gives competitive results and there is space for further improvements. Generated encodings are generally quite large and solve practical problems and as such can be used as benchmarks for the evaluation of maxSAT solvers.

The encodings can be optimized better to suit a particular instance. A few special cases are handled automatically, but for some we have to do them by hand by looking at the characteristics of the instance (such as previously discussed for ItalyInstance1, SpainInstance and EnglandStPaul). Introducing mechanisms that would automatize these process would be very valuable to both decrease the size of instances and improves the solution process. Additionally, different encodings for cardinality constraints as in [15] and sorting networks [3] might offer improvements.

For a number of instances, significant amount of clauses are generated to encode basic timetabling requirement, such as Avoid Clashes Constraint. We believe that improvements might be achieved if these constraints are handled in a special manner within the solver, rather than encoding them as clauses. SMT solvers offer such possibilities and might be a good choice for such problems.

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

6 Acknowledgements

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

References

1. Eighth max-sat evaluation. http://maxsat.ia.udl.cat:81/13/introduction/index.html. Accessed: 2014-1-30.

2. International timetabling competition 2011. http://www.utwente.nl/ctit/hstt/itc2011/welcome/. Accessed: 2014-1-30.
3. Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In *Theory and Applications of Satisfiability Testing-SAT 2009*, pages 167–180. Springer, 2009.
4. Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming–CP 2003*, pages 108–122. Springer, 2003.
5. Samuel S. Brito, George H. G. Fonseca, Túlio A. M. Toffolo, Haroldo G. Santos, and Marcone J. F. Souza. A SA-VNS approach for the high school timetabling problem. *Electronic Notes in Discrete Mathematics*, 39:169–176, 2012.
6. Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multi-commodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 184–193. IEEE, 1975.
7. Santos H. G. Toffolo T. A. M. Brito S. S. Souza M. J. F. Fonseca, G. H. G. 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.
8. Ahmed Kheiri, Ender Ozcan, and Andrew J Parkes. 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.
9. Jeffrey H Kingston. The khe high school timetabling engine. 2010.
10. Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
11. Chu Min Li, Felip Manyà, Nouredine Mohamedou, and Jordi Planes. Exploiting cycle structures in max-sat. In *Theory and Applications of Satisfiability Testing-SAT 2009*, pages 467–480. Springer, 2009.
12. Gerhard Post, Samad Ahmadi, Sophia Daskalaki, JeffreyH. Kingston, Jari Kyngas, Cimmo Nurmi, and David Ranson. An xml format for benchmarks in high school timetabling. *Annals of Operations Research*, 194(1):385–397, 2012.
13. Gerhard Post, Luca Di Gaspero, Jeffrey H Kingston, Barry McCollum, and Andrea Schaerf. The third international timetabling competition. *Annals of Operations Research*, pages 1–7, 2012.
14. Gerhard Post, Jeffrey H Kingston, Samad Ahmadi, Sophia Daskalaki, Christos Gogos, Jari Kyngas, Cimmo Nurmi, Nysret Musliu, Nelishia Pillay, Haroldo Santos, et al. Xhstt: an xml archive for high school timetabling problems in different countries. *Annals of Operations Research*, pages 1–7, 2011.
15. Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming-CP 2005*, pages 827–831. Springer, 2005.
16. Matias Sørensen and Florian HW Dahms. A two-stage decomposition of high school timetabling applied to cases in Denmark. *Computers & Operations Research*, 43:36–49, 2014.
17. Matias Sørensen, Simon Kristiansen, and Thomas R Stidsen. 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)*, page 489, 2012.
18. Matias Sørensen and Thomas Riis Stidsen. Comparing solution approaches for a complete model of high school timetabling. Technical report, DTU Management Engineering, 2013.