MaxSAT-Based Large Neighborhood Search for High School Timetabling

Emir Demirović and Nysret Musliu

Vienna University of Technology
Institute of Information Systems
Databases and Artificial Intelligence Group
Favoritenstraße 9/184-2, 1040 Vienna, Austria
{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), times, 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 have been proven to be NP-complete. We propose a new algorithm for HSTT which combines local search with a novel maxSAT-based large neighborhood search. A local search algorithm is used to drive an initial solution into a local optimum and then more powerful large neighborhood search (LNS) techniques based on maxSAT are used to further improve the solution. We prove the effectiveness of our approach with experimental results on instances taken from the Third International Timetabling Competition 2011 and the XHSTT-2014 benchmark archive. We were able to model 27 out of 39 instances. The remaining 12 instances were not modeled because the currently used maxSAT formulation for XHSTT does not support resource assignments in general. For the instances which could be modeled, our algorithm shows good performance when compared to other XHSTT state-of-the-art solvers and for several instances new best known upper bounds have been computed.

Keywords: maxSAT, high school timetabling, large neighborhood search, local search

1 Introduction

The problem of high school timetabling (HSTT) is to coordinate resources (e.g. rooms, teachers, students) with times in order to fulfill certain goals (e.g. scheduling lectures). Every high school requires some form of timetabling which is a well known and wide spread problem. The difference between a good and a bad timetable can be significant, as timetables directly contribute to the quality of the educational system, satisfaction of students and staff, etc. Every timetable affects hundreds of students and teachers for prolonged amounts of time, since each timetable is typically used for at least a semester, making HSTT an extremely important and responsible task. However, constructing timetables by

hand can be time consuming, very difficult, and error prone. Thus, developing high quality algorithms which would generate automatically timetables is of great importance.

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 practical problems can be very large and have many different constraints, high school timetabling requirements vary from country to country. Due to this, many variations of the timetabling problem exist. A lot of research has been done and HSTT is still an active field of research, even having its own specific HSTT competition ITC 2011.

In order to standardize the formulation for HSTT, researches have recently proposed a general high school timetabling problem formulation [19] called XHSTT. This formulation has been endorsed by the Third International Timetabling Competition 2011 (ITC 2011) [18, 19] which attracted 17 competitors from across the globe. In this work, we consider the general HSTT problem formulation (XHSTT). We took all relevant XHSTT instances (see Section 6.1) and out of the pool of 39 instances we were able to model 27 of them. The remaining 12 instances were not modeled because the currently used maxSAT formulation for XHSTT does not support resource assignments in general. We have a specific modeling for resource assignments (Assign Resource Constraints and related constraints) for two instances, but our current model is not practical for other instances with resource assignments.

The main contributions of this paper are as follows:

- We present a new large neighborhood search (LNS) algorithm which exploits maxSAT to solve XHSTT. To this end, we propose a destroy operator with two neighborhood vectors and a novel insertion approach, for which we modified the open-source maxSAT solver Open-WBO [15] to support our exhaustive insertion strategy. The overall algorithm combines local search and LNS to solve XHSTT instances which are modeled as maxSAT.
- We experimentally compare our algorithm with other approaches in the literature on 27 out of 39 instances. Our approach outperforms the state-of-the-art solvers on many instances. Using our algorithm, we managed to compute four new best known upper bounds. The approach presented is a novel contribution to the state-of-the-art for XHSTT. Furthermore, to the best of our knowledge, it is the first time maxSAT is used within a LNS scheme.

The rest of the paper is organized as follows. In Section 2, we present the problem description followed by related work in Section 3. For completeness, the (max)SAT problem and its modeling for high school timetabling is briefly described in Section 4. The main section of this paper is Section 5, which describes our algorithm. Afterwards, Section 6 provides experimental results on benchmark instances. Finally, a conclusion is given in Section 7.

2 Problem description

High School Timetabling has been extensively studied in the past. However, a lot of work has been done in isolation, because each country has its own educational system which resulted in many different timetabling formulations. Thus, it was difficult to compare algorithms and the state-of-the-art was unclear. To solve this issue and encourage timetabling research, researchers have agreed on a standardized general timetabling formulation called XHSTT [19]. This formulation was general enough to be able to model education systems from different countries and was endorsed by the International Timetabling Competition 2011. This is the formulation used in this work.

The general High School Timetabling formulation specifies three main entities: times, resources, and events. Times refer to the available discrete time units, such as Monday 9:00-10:00 and Monday 10:00-11:00. 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 (who needs to be determined) and a specific student group (both the teacher and the student group are considered resources) and two units of time (two *times*). Events are to be scheduled into one or more *solution events* or *subevents*. For example, a mathematics lecture with a total duration of four hours can be split into two subevents with a duration of two hours each, but can also be scheduled as a single subevent with a duration of four hours (constraints may restrict the durations of subevents).

Constraints impose limits on what kind of assignments are desirable. They may state 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. It is important to differentiate between hard and soft constraints. The former are very important and are given precedence over the latter, in the sense that any single violation of a hard constraint is more important than all soft constraints violations combined. Thus, one aims to satisfy as many hard constraint violations as possible, and then optimize for the soft constraints. In the general formulation, any constraint may be declared hard or soft and no constraint is predefined as such, but rather left as a modeling option based on the specific timetabling needs. Additionally, each constraint has several parameters, such as the events or resources it applies to and to what extent (e.g. how many idle times are acceptable during the week), its weight, and other properties, allowing great flexibility.

We now give an informal overview of all the constraints in XHSTT (as reported in [19]). There is a total of 16 constraints (plus preassignments of times or resources to events, which are not listed).

Constraints related to events:

1. Assign Time Constraints - assign the specified number of times to specified events.
2. Split Events Constraints - limits the minimum and maximum duration of subevents and the amount of subevents that may be derived from specified

events. Distribute Split Events Constraints (below) gives further control on subevents.
3. Distribute Split Events Constraints - limits the number and duration of subevents for specified events.
4. Prefer Times Constraints - specified times are preferred over others for specified events.
5. Avoid Split Assignments Constraints - assign the same resource for all subevents derived from the same event.
6. Spread Events Constraints - specified events must be spread out during the week.
7. Link Events Constraints - specified events must take place simultaneously.
8. Order Events Constraints - specified events must be scheduled one after the other with a specified number of times in between.

Constraints related to resources:

1. Assign Resource Constraints - assign specified resources to specified events.
2. Prefer Resources Constraints - specified resources are preferred over others for specified events.
3. Avoid Clashes Constraints - specified resources cannot be used by two or more subevents at the same time.
4. Avoid Unavailable Times Constraints - specified resources cannot be used at specified times.
5. Limit Idle Times Constraints - specified resources within specified days must have their number of idle times lie between given values.
6. Cluster Busy Times Constraints - specified resources' activities must all take place within a minimum and maximum number of days.
7. Limit Busy Times Constraints - specified resources within specified days must have their number of busy times lie between given values.
8. Limit Workload Constraints - specified resources must have their workload lie between given values.

For more details regarding the problem formulation, see [17, 19].

3 Related work

For HSTT, both heuristic and exact methods have been proposed. Heuristic methods were historically the dominating approach, as they were able to provide good solutions in reasonable amounts of time even when dealing with large instances, albeit not being able to always obtain or prove optimality. Recently exact methods have been proposed and had success in obtaining good results and proving bounds, but require significantly more time (days or weeks).

All of the best algorithms in the International Timetabling Competition 2011 (ITC 2011) were algorithms based on heuristics. The winner was the group GOAL, followed by Lectio and HySST. In GOAL, an initial solution is generated, which is further improved with Simulated Annealing and Iterated Local

Search, using seven different neighborhoods [2]. Lectio uses an Adaptive Large Neighborhood Search [25] with nine insertion methods based on the greedy regret heuristics [26] and fourteen removal methods. HySST uses a Hyper-Heuristic Search [9].

Afterwards, the winning team of ITC 2011 has developed several new Variable Neighborhood Search (VNS) approaches [8]. All of the VNS approaches have a common search pattern: from one of the available neighborhoods, a random solution is chosen, after which a descent method is applied and the resulting solution is accepted if it is better than the previous best. Each iteration starts from the best solution. The most successful VSN algorithm was the Skewed Variable Neighborhood in which a relaxed rule is used to accept the new solution, taking into consideration the cost of the new solution as well as its distance from the best solution. A related approach is Late Acceptance Hill Climbing for XHSTT [7], in which a solution is accepted based on its comparison with the previous k solutions, where k is a parameter.

Kingston [10] introduced an efficient heuristic algorithm called KHE14 which directly focuses on repairing $defects$ (violations of constraints). Constraint violations are examined individually and specialized procedures are developed for most constraints to repair them. The algorithm is designed to provide high quality solutions in a small amount of time, but does not necessarily outperform other methods with respect to solution quality.

XHSTT has been modeled with Integer Programming (IP) in [12]. This exact approach is able to compute good (and in some cases optimal) solutions as well as lower bounds over longer periods of time using Gurobi (a commercial optimization solver). Additionally, a Large Neighborhood Search with IP has also been developed in [28] which is more efficient than pure IP when given limited time.

Another exact approach is the maxSAT approach proposed in [3]. Most of the XHSTT instances could be modeled as a Partial Weighted maxSAT problem and are then solved with a maxSAT solver. The approach can yield good (in some cases optimal) results, although it too requires longer running times. Satisfiability Modulo Theory (SMT) has also been investigated for XHSTT in [4], but this SMT method is still not mature enough to handle XHSTT instances efficiently.

Furthermore, several IP-based techniques have been introduced for similar HSTT problems which provide bounds and good solutions after long running times [20] [24] [27], including a fix-and-optimize IP-based hybrid approach reported in [5].

Even though significant work has been done for HSTT, many problems are still not solved efficiently or optimally. Therefore, calculating high quality solutions and providing new modeling approaches are important issues in this domain.

Our algorithm has several novel features compared to LNS approaches proposed in Lectio [25], the IP LNS in [28], and the fix-and-optimize approach in [5]. First, we use a novel exhaustive maxSAT insertion technique. Further, the

destroy operators are used in a different way and our approach is combined with a local search technique.

4 Modeling XHSTT as maxSAT

In this section we briefly describe the Satisfiability problem (SAT), its extention called maxSAT, and the modeling of XHSTT with maxSAT.

4.1 SAT and maxSAT

The Satisfiability problem (SAT) is a decision problem where it is asked if there exists an assignment of truth values to variables such that a propositional logic formula is true (that is, the formula is $satisfied$). A propositional logic formula is built from Boolean variables using logic operators (such as $\wedge$ AND, $\vee$ OR, and $\neg$ NOT) and parentheses. The formula is usually given as a conjuction of clauses (in Conjuctive Normal Form). A clause is a disjunction of literals, where a literal is a variable or its negation. For example, the formula $(X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_3)$ has three variables (X_1, X_2, and X_3), two clauses, and is said to be satisfiable because there exists an assignment, namely $(X_1, X_2, X_3) = (true, false, false)$, which satisfies the formula. However, had we inserted the clause $(\neg X_1 \vee X_2 \vee X_3)$ the same assignment would no longer satisfy the formula. Instead of writing $\neg X_1$ it is common to write $\overline{X}_1$ and this is the notation used in this work.

The extension of SAT considered in this work is Partial Weighted maxSAT, in which clauses are partitioned into two types: hard and soft clauses. Each soft clause is given a weight. The goal is to find an assignment which satisfies the hard clauses and minimizes the sum of the weights of the unsatisfied soft clauses. For more information about SAT and maxSAT, the interested reader is referred to [1].

4.2 XHSTT as maxSAT

We model XHSTT using the Partial Weighted maxSAT formulation given in [3]. Once a XHSTT instance has been modeled as maxSAT, any satisfiable assignment of cost c for the maxSAT formulation directly corresponds to a XHSTT solution of cost c. However, if the hard constraints cannot be satisfied, the model does not admit any solution, even though such solutions are possible in XHSTT (in which case the $infeasibility$ value would tell us how severe is the violation). The modeling revolves around Boolean variables $Y_{e,t}$, which are true iff a subevent of event e is taking place at time t (not just starting at t). A solution to XHSTT corresponds to assigning truth values to these variables. Their assignments are constrained by other variables and clauses which encode XHSTT constraints.

We do not give a complete overview of the modeling of XHSTT as Partial Weighted maxSAT, but instead give brief examples of the encoding of two XHSTT constraints, and refer the interested reader to [3] for full details.

Algorithm 1: XHSTT algorithm

```
begin
    S ⟵ initialSolutionAndLocalSearch()
    t_current ⟵ current time; t_vec ⟵ t_current
    t_default ⟵ 30 secs; t_max_active ⟵ 500 secs
    (V_r, V_d) = generateNeighborhoodVectors()
    active(V_r) ⟵ true; active(V_d) ⟵ false
    while there is time left do
        if condition for vector switch satisfied then
            switch active vector
            t_vec ⟵ t_current
            randomize orderings of active vector
        N ⟵ selectNeighborhood()                          ▷ Section 5.1
        S_destroyed ⟵ destroy(S, N)
        S_old ⟵ S
        S ⟵ insert(S_destroyed, allocatedTime(N))
        if insertion was successful then
            previouslySuccessful(N) ⟵ true                ▷ Used for selection;
            cost(N) ⟵ cost(S)                             ▷ Used for selection;
            allocatedTime(N) ⟵ t_default
        else
            previouslySuccessful(N) ⟵ false               ▷ Used for selection;
            allocatedTime(N) ⟵ allocatedTime(N) + 10
            S ⟵ S_old
```

An important part of modeling are cardinality constraints which impose limits on the truth values assigned 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 state that at least, at most, or exactly k literals out of the specified ones must be true. These constraints are used frequently when modeling XHSTT. One way to encode the cardinality constraints, which we refer to as the Combinatorial encoding, is to forbid all undesired assignments. For example, for $atMost\_2\{x_1, x_2, x_3, x_4\}$ we forbid every possible combination of three literals being simultaneously set to true with the following clauses: $(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$, $(\overline{x_1} \vee \overline{x_2} \vee \overline{x_4})$, $(\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$ and $(\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$.

There are many ways to encode cardinality constraints, each with its own advantages and disadvantages in terms of the number of auxiliary variables and clauses that are needed, propagation properties, etc. The type of encoding chosen impacts the solution process and is therefore very important. However, it is often difficult to assess which cardinality constraint would be appropriate for a particular application. In [3], experiments were made with a number of different cardinality constraints and maxSAT solvers in order to choose the best encoding for XHSTT. During experimentation it was concluded that Assign Time Constraints (ATC) are particularly important since changing their encoding independently from other constraints had significant impact during the search (using a "bad" encoding for ATC leads to noticeably worse solutions). We believe that a good encoding for this constraint is crucial, because ATC is fundamental for timetabling as it has (arguably) the most impact on other constraints. Therefore, ATC was given special treatment during the experimentation. Different combinations of cardinality encodings were considered for ATC and other constraints, and after extensive experimentation the best combination for cardinality con-

straints was the following: Sequential encoding [23] for Assign Time Constraints, Combinatorial encoding for other constraints, and Bit adder in cases where Combinatorial encoding is too large ($n \geq 50 \vee (n \geq 42 \wedge k \geq 5)$, where n is the number of variables and k is the cardinality constraint constant). We believe that these choices proved to be the best because of their simplicity over other variants when encoding relatively small cardinality constraint requirements which are encountered in XHSTT. We do not describe the cardinality constraints in more detail, but refer the interested reader to [3].

We now give two examples for encoding XHSTT constraints. As mentioned earlier, we frequently use cardinality constraints when modeling XHSTT constraints. For example, we use cardinality constraints in order to model Assign Time Constraints, which impose that each event $e \in E$ must be assigned a number of times equal to its duration d_e:

$$\bigwedge_{\forall e \in E} (exactly\_d_e[Y_{e,t} : t \in T]) \tag{1}$$

In order to model soft XHSTT constraints, $soft$ cardinality constraints are used which penalize undesirable assignments rather than forbidding them. The farther away from the imposed limits they are the more severe the penalty is. For example, for the soft cardinality constraint $atMost\_2\{x_1, x_2, x_3, x_4\}$, the assignment $(x_1, x_2, x_3, x_4) = (1, 0, 0, 0)$ would incur no penalty, while assignments $(x_1, x_2, x_3, x_4) = (1, 0, 1, 1)$ and $(x_1, x_2, x_3, x_4) = (1, 1, 1, 1)$ would incur a penalty of 1 and 2 (respectively) if a linear penalty function is used. An example of their usage would be to model the Avoid Unavailable Times Constraints which, if given as a soft constraints, try to minimize the occurrences where resources $r \in R_{spec}$ are busy during undesired times $T_{undesired}$ (both $T_{undesired}$ and R_{spec} are specified in the constraint). Let the variables $X_{t,r}$ be such that they are true iff a resource r is busy at time t. With these variables we encode the constraint as:

$$\bigwedge_{\forall r \in R_{spec}} (atMost_{soft}\_0[X_{t,r} : t \in T_{undesired}]) \tag{2}$$

The interested reader will find the full details about the maxSAT formulation in [3].

5 Algorithm description

We introduce a new algorithm for the XHSTT problem which combines local search and maxSAT-based large neighborhood search (LNS). LNS is a technique first introduced by Shaw [21] which has been used for many problems, including related timetabling problems [16], but never in a combination with maxSAT. The LNS algorithm is the main contribution of this paper and consists of two main components: destroy and insertion operations. The destroy operator unschedules certain subevents. Insertion is the opposite: it assigns times to the previously

unscheduled subevents. We proceed by explaining in detail the destroy operator, the insertion technique with maxSAT, the initial solution generation by local search, and finally we give a complete overview of the algorithm.

5.1 Destroy operator

The destroy operator selects a neighborhood from one of the two neighborhood vectors and destroys the solution with respect to the selected neighborhood. The two neighborhood vectors are based on resources and days. We first introduce these vectors and then explain how a neighborhood is selected from them.

Neighborhood vector based on resources. This vector consists of all possible combinations of two resources (e.g. rooms, teachers, etc). When a neighborhood from this vector is used, every subevent which uses at least one of these two resources is unscheduled (unassigned from each time unit), as well as every subevent which is linked to any of the unassigned subevents via Link Event Constraints. A similar idea of unscheduling resources has been presented in [11].

Using events that share one of the selected resources and linked events are both important for the insertion step. For example, students in HSTT have compact schedules and attempting to assign a subevent which requires class C to a new time without previously unassigning other subevents which also require class C will most likely result in a clash. Additionally, since linked events should take place simultaneously, unassigning one subevent requires that its linked subevents also get unassigned in order to be able to schedule all subevents at a different time.

Neighborhood vector based on days. This vector consists of all possible combinations of days. For example, if we are considering a timetable with three days {Mon, Tu, Wed}, the corresponding day vector would be: { {Mon}, {Tu}, {Wed}, {Mon, Tu}, {Mon, Wed}, {Tu, We}, {Mon, Tu, Wed} }. When a neighborhood from this vector is used, subevents assigned to times pertaining to the days of the selected neighborhood are unscheduled.

This neighborhood vector is used to make better rearrangements within each individual day (e.g. for Limit Idle Times Constraint), as well as to be able to move, merge, or split subevent throughout different days (e.g. for Spread and Cluster Events Constraints).

Neighborhood selection. Only one of the two neighborhood vectors are considered active and neighborhoods are selected from the active one. The other neighborhood vector will become active after either a timeout occurs (equal to half of the total running time) or if all neighborhoods from the active vector have been visited exactly twice. A neighborhood can be visited a second time only after all other neighborhoods in the vector have been visited once, and no neighborhood will be visited more than twice within a single activation of its neighborhood vector.

After using a neighborhood to destroy the solution and applying the insertion operator based on a maxSAT solver, we record the objective value and whether the insertion operator has successfully exhaustively explored the neighborhood. If the maxSAT solver did not show progress within a certain amount of time

(did not find a better solution nor proved that one does not exist), we stop further exploration, label the attempt as unsuccessful, and move on to the next neighborhood. Initially we set the objective value of a neighborhood to be a large number and label it as if an insertion operation has successfully terminated. Note that if the recorded objective value for a neighborhood is the same as the current upper bound then it means that the solution has not changed since the last time the neighborhood was visited.

The order of the neighborhoods in a neighborhood vector are reset randomly when a neighborhood vector becomes active or when all of its members have been visited once. Neighborhoods are then visited in order, with the exception of the day neighborhood vector where smaller neighborhoods are visited first before proceeding to larger ones (e.g. single-day neighborhoods are visited before two-day neighborhoods).

In some cases a neighborhood will be skipped rather than examined. This is done in order to avoid spending time with neighborhoods that are likely to be unsuccessful. A neighborhood is skipped if its recorded objective value (previously explained in this section) is equal to the current upper bound, and any of the following conditions hold:

- Its previous insertion attempt was successful. The neighborhood is surely of no use as it has already been thoroughly explored.
- It is being visited for the second time since the current neighborhood vector has been set active. In this case we heuristically choose to skip the neighborhood, because it has already been unsuccessfully explored, so the chances are that it will be unsuccessful again when given the same amount of time. It is better to allocate time to other neighborhoods.
- The neighborhood is an element of the day neighborhood vector, and since the last time the neighborhood vector became active, another neighborhood which is a subset of the currently considered one has not successfully been explored. For example, the three-day neighborhood {Mon, Tu, Wed} will be skipped if the single-day neighborhood $\{Mon\}$ has been labeled unsuccessful.

Each neighborhood is allocated a specified amount of time. For each subsequent run on a neighborhood that has not successfully terminated, more time is allocated. After successfully terminating, the next run with that neighborhood will be given the default amount of time.

We now further comment on our motivation for choosing this neighborhood selection strategy. Our strategy was devised through heuristic reasoning and experimentation, leaving the possibility that we might have missed other better strategies. Nevertheless, experiments in Section 6 show that our approach is effective. The two main reasons for our strategy are:

First, the idea is to examine smaller neighborhoods before going on to larger ones. Accordingly, local search (see Section 5.3) takes place before the LNS algorithm, the resource based neighborhood vector precede the day neighborhood vector, and smaller day neighborhoods are examined before larger ones. The reason is that smaller neighborhoods are much easier to explore and often provide

improvements quickly. Once they are no longer useful, we go to larger ones, which will now be explored faster because of the clauses learned and bounds obtained (see Section 5.2) in the process of solving smaller neighborhoods (as variant LNS.v2 confirms, see Section 6.7).

Second, we want to thoroughly examine all neighborhoods in the hope of finding good ones. We wish to avoid situations where a useful neighborhood is getting overlooked because other neighborhoods are getting selected repeatedly. Therefore, we introduce fairness in the sense that a neighborhood will be examined a second time only after all other neighborhoods have been examined at least once. We chose to explore every neighborhood exactly twice before moving on to the other neighborhood vector. Considering the same neighborhood again after a series of other neighborhoods have been examined might be useful, but we opted not to examine it more than twice in order to avoid getting stuck in a search space which consumes too much time. The neighborhood skipping mechanism directs the search by filtering out neighborhoods which are unlikely to contain better solutions.

5.2 Insert operation

The insertion operation repairs the solution by trying to find the best possible insertion for the unscheduled events using an exhaustive search based on a maxSAT formulation. The idea is to call a maxSAT solver to solve the maxSAT formula which represents a XHSTT instance while fixing the assignments of all subevents which were *not* unassigned in the destroying phase. In principle, any exhaustive search technique could be used for the insertion operator. In Section 6.6 we provide a comparison of maxSAT with Integer Programming and further elaborate on our decision to choose maxSAT.

With regard to the traditional decision problem, the problem of solving a SAT instance while fixing certain variables is known as "solving under assumptions". This can be done by having the solver first "branch" on the fixed variables and then continue doing a regular SAT search. However, this kind of technique cannot be directly used for maxSAT because the underlying formula is being changed during the solution process. We elaborate on this further below.

We use the Linear maxSAT algorithm (Algorithm 2) [13] which makes repeated calls to a SAT solver and after each call adds constraints which ask for a better solution than the previous one (in Algorithm 2, K is the set of soft constraints). The optimal solution is obtained when the SAT solver returns $false$.

Another maxSAT algorithm is based on *unsatisfiable cores*. An unsatisfiable core is a set of clauses whose conjuction is unsatisfiable. Initially one may consider all soft clauses as hard and then attempt to solve the formulation. If the solver reports that the formulation is unsatisfiable (not all soft clauses can be satisfied), an unsatisfiable core is computed and used to relax the SAT formulation and the process is repeated iteratively.

We opted to use the Linear algorithm because better results were reported in [3] when compared to core guided solvers for XHSTT instances. In this algorithm, the original maxSAT formula gets changed because bounds are added at each

Algorithm 2: Linear algorithm for maxSAT

```
begin
    P ⟵ maxSAT formula
    c = ∞
    bestAssignment = ∅
    while isSatisfiable(P) do
        bestAssignment = satisfiableAssignment(P)
        c ⟵ cost(P, bestAssignment)
        P = P ∪ (∑_{i∈K} softConstraint(i) < c)
```

iteration, in addition to *learned* clauses which are added to direct the search (see [22] for clause learning). It is not straightforward to remove the added clauses at later stages of the algorithm, because clauses are learned with respect to other clauses (including other learned clauses) and removing some clauses may therefore invalidate previously learned clauses. To the best of our knowledge, no maxSAT solver supports this kind of search. An alternative is to restart the solver after each call, losing possibly valuable learned clauses and bounds. This motivated us to investigate a different approach: instead of restarting between calls, we keep the modified formula intact. Thus, each call to the solver depends on all previous calls due to the bounds and learned clauses. When querying the solver with a new set of assumptions, it will attempt to report the best possible solution, but only if it is better than all of the previously computed solutions. To this end, we modified the linear algorithm in the open-source maxSAT solver Open-WBO [15]. Keeping the solver state between runs proved to be important and this is discussed in more detail in Section 6.7. A different approach related to ours is presented in [14] for lower bounding maxSAT algorithms.

5.3 Initial solution

We use two approaches for generating an initial solution. The first approach is to use the KHE14 algorithm [10] to obtain an initial solution. We chose KHE14 because it is a publicly available state-of-the-art solver designed to produce high quality solutions very quickly. If KHE14 does not succeed in generating a feasible solution, we generate an initial solution by ignoring all soft constraints and solving the corresponding XHSTT as a pure SAT instance, with the exception of Split Events Constraints which are treated as hard constraints even if they are given as soft ones. This solution is then improved with a local search procedure. Split Events Constraints are treated as hard constraints because the following local search algorithm does not split or merge subevents, making it very difficult to find a good solution if the constraint is not satisfied initially. The local search procedure is based on simulated annealing (SA) and it uses two neighborhoods: swap (the times of two subevents are exchanged) and block-swap (similar to a swap, with the exception that if a swap move causes two subevents to overlap, an appropriate time is assigned to the second one so that they appear one after

the other). Similar neighborhoods were previously used in [8]. In our algorithm we only apply feasible moves to subevents that share resources. The importance of using subevents with shared resources has been discussed in Section 5.1. The following parameter values are used in the SA: the initial temperature $T_{initial}$ is set to 0.1 and the temperature is multiplied by $\alpha = 0.99$ every 10 iterations. If the last five improvements are made at the initial temperature, then the algorithm starts accepting only improving moves. The probabilities of selecting the swap or block-swap neighborhoods at each iteration are set to 2/3 and 1/3, respectively. When a neighborhood is selected, two subevents are randomly selected until they both share a resource (this is attempted 100000 times). The goal is to quickly improve the solution through simple moves leaving more complicated moves to LNS. We note that the generation of the initial solution takes only a small amount of time when compared to the maxSAT-based LNS part. However, it is still important and we discuss in Section 6.7 the impact of using a non optimized initial solution (c.f., $LNS.v1$ variant).

5.4 Algorithm summary

The pseudo-code in Algorithm 1 summarizes our problem-solving approach. An initial solution is generated and improved with SA, after which the LNS provides further improvements. LNS consists of two parts: the destroy operator which unschedules subevents based on a neighborhood chosen using either the resource or day vector (described in Section 5.1), and the insertion operator which reschedules the previously selected subevents and records the performance of the chosen neighborhood. The algorithm iterates until a timeout occurs (in our experiments, 1000 seconds).

6 Experimental results

In this section we introduce the problem instances, experimentally evaluate our algorithm including different variants, and compare with existing state-of-the-art solutions.

6.1 Instances

We evaluated our approach on XHSTT benchmark instances which can be found in the repository of the International Timetabling Competition 2011 (ITC 2011) [1]. We used the XHSTT-2014 benchmark set which contains instances that were carefully selected by the ITC 2011 over the years and are meant to be interesting test beds for researchers. Additionally, we included every instance used in the competition (these two sets of instances overlap). This way we took into consideration all relevant XHSTT instances to the best of our knowledge.

Overall, we can efficiently model 27 out of 39 (70%) selected instances with maxSAT. The remaining 12 instances were not modeled because the currently

[1] http://www.utwente.nl/ctit/hstt/itc2011/welcome/

used maxSAT formulation for XHSTT does not support resource assignments in general. We have a specific modeling for resource assignments (Assign Resource Constraints and related constraints) for two instances (FinlandArtificialSchool, EnglandStPaul), but our current model is not practical for other instances with resource assignments. Unfortunately, in this case the number of variables and clauses is very large, and until now we could not come up with a more efficient encoding for these constraints. Thus, for the remaining 12 instances, we currently do not have an appropriate model and could not experiment with them.

In the test instances, the number of times ranges from 25 to 125, the number of resources from 8 to 99, the number of events from 21 to 809 with a total duration between 75 to 1912. These numbers vary heavily from instance to instance. We do not provide detailed information for the instances, but refer the interested reader to [17, **?**].

6.2 Computer Specification and Computational Time Limit

All tests were performed on an Intel Core i3-2120 CPU @ 3.30GHz with 4 GB of RAM and each instance was given a single core. To determine the computational time we used the ITC's benchmark tool which is designed to test how fast a machine performs operations relevant for timetabling. The tool suggested a computational time of 1007 seconds, which is a similar to the one used in the second round of ITC 2011 (1000 seconds).

6.3 Solvers

We compared our approach (abbreviated by LNS) with VNS [8], KHE14 [10], Matheuristic [28], and a pure maxSAT approach, all using the time limit of 1000 seconds. Additionally, we ran our solver for longer running times and compared its results to the best known upper bounds.

VNS [8] was developed by the winners of ITC 2011 after the competition. KHE14 [10] is a competitive solver which is also used in our approach for generating the initial solution, as described in Section 5.3. Matheuristic [28] is an Integer Programming-based LNS algorithm. These solvers were chosen because they are state-of-the-art XHSTT algorithms which can generate good solutions in the time limit set by the competition. The pure maxSAT approach runs the maxSAT solver used in our algorithm (open-WBO) on the same maxSAT model of the XHSTT instance as LNS, but without using any of our LNS techniques.

6.4 Results

The algorithms were run for the same amount of time (1000 seconds) on the same machine, except Matheuristic because it is not available to the public. In this case, we compare with the results reported in [28]. We believe the comparison is fair because the authors used the same benchmarking tool as we did to determine the computational time. When no result was reported with Matheuristic for a

given instance, we put '-' in the table. Since KHE14 was designed to run for shorter durations, we ran the algorithm multiple times during the time limit and report the best solution found.

We denote the objective function cost as a pair (x, y), where x and y are the hard and soft constraint costs. If the hard cost is zero, we only present the soft cost. The algorithms include some forms of randomness during their execution and we present the average values of five runs. For BrazilInstance5, we included the best solution computed out of the five runs, since the solution is the best solution known so far for this instance.

The comparison is given in Table 1. Instances noted above the bold horizontal line are part of the XHSTT-2014 benchmark set, while the ones below are not but have been used in the competition.

Name	LNS	VNS	KHE14	maxSAT	Math.
Brazil2	**5.4***	(1, 44.4)	14	57	6
Brazil4	61.4	(17.2, 94.8)	$-^c$	214	**58**
Brazil6	**50.6**	(4, 223.6)	124	352	57
GreecePatras10	**0***	**0***	**0***	2329^m	-
GreeceUni4	**5**	6.2	8	141	12
GreeceHSchool	**0***	**0***	**0***	**0***	-
Italy4	**35**	178	40	16979^m	48
SAfricaWood	1.2*	(2, 6.2)	(3, 0)	**0***	(2, 429)
SAfricaLewitt	$-^m$	**8**	$-^c$	1039^m	-
FinlandHSchool	**9.8**	36.6	29	812	-
FinlandCollege	54.6	(2.8, 25)	**20**	1309	-
FinlandSSchool	95.2	(0.4, 93)	**90**	504	-
KosovaInst	$-^m$	14	(8 , 6)	29946^m	-
EngStPaul	-	(92, 1739.4)	(26, 764)	$-^m$	-
Brazil1	**39**	52.2	54	**39**	-
Brazil3	**23***	107.8	116	75	-
Brazil5	**19.4 (17)**	(4, 138.4)	(1, 179)	224	-
Brazil7	**136.2**	(11.6, 234.6)	179	603	-
Italy1	**12**^{p}	21.2	31	**12**^{p}	-
FinlandSSchool2	**0.2***	**0.2***	2	3523	-
FinlandESchool	**3**^{p}	**3**	4	**3**^{p}	-
FinlandASchool	**0***	(5.4, 4.2)	(4, 6)	12	-
GreecePreveza	38.2	**2***	**2**	5617	-
GreeceUni3	7	**5**	7	7	6
GreeceUni5	**0***	**0***	**0***	**0***	**0***
GreeceAigio	368	(0.2, 6.2)	**6**	4582	180
NetherGEPRO	$-^m$	(3, 7518)	$-^c$	$-^m$	-

Table 1. Comparison of results with different methods using a time limit of 1000 seconds. The best results are in bold. Legend: * optimality was found within the five runs; ^{m} 'out of memory'; ^{p} proof of optimality; ^{c} program crash (a bug); and " $-$ " solution has not been produced within the time limit .

Our approach outperforms the VNS solver for 16 instances out of 27, while in five cases it gives the same result. For KHE14, we get a better result in 15 cases and a tie in four cases. When compared to the pure maxSAT approach, our algorithm has a clear advantage given the computational limit. As for the Matheuristic algorithm, we get a better result for five out of nine instances and a tie for one instance based on the published results. Overall, our solver is better than or equal any other solver in 16 cases.

We note that a new best known upper bound has been computed for BrazilInstance5 within the time limit. Moreover, our solver found and proved optimality within the time limit in two cases, while the other solvers were not able to generate proofs of optimality. Our method is able to prove optimality for these instances because larger and larger neighborhoods are explored over time until the whole problem is solved, that is, the complete solution is destroyed and optimally reassigned. Further improvements to the best known solutions with other time limits are discussed in Section 6.5.

Overall, the results obtained are very encouraging, as our approach outperformed the state-of-the-art solvers on many instances that were modeled with our maxSAT approach.

6.5 Longer runs and additional improvements to the best known solutions

With more computational time, it was possible to produce three new best known solutions (in addition to the ones from the previous section). For the KosovaInstance, we ran our algorithm using the previous best known solution as a starting point. In Table 2, we present the results in column $LNS(time)$ together with the time used in brackets, but only when it was possible to produce better solutions with longer running times. In column $'Best'$ we give the previously best known solution values. The other instances (including the ones that we have previously computed optimally) are marked with $'-'$. We note that after providing our results for the Brazilian instances to the ITC 2011's repository, the instance modelers decided to slightly modify them. After reviewing our solution they decided to make the Prefer Times Constraints hard constraints, instead of soft ones. Nevertheless, we provide these results as no other method could compute those values.

6.6 MaxSAT vs Integer Programming

We now provide a comparison of maxSAT with Integer Programming and further elaborate on our decision to choose maxSAT for the insertion operator.

We compare Integer Programming[2] [12] and pure maxSAT for XHSTT. Both solvers were run for 30 seconds (the initial time used in this paper for each neighborhood) and 600 seconds using a single thread. The results are given in Table 3. In most instances the pure maxSAT approach produces better results.

[2] We thank the authors for providing their solver

The maxSAT results can easily be reproduced by running our maxSAT instances [3] with open-WBO (parameter: -algorithm=1). Based on these results, we believe that maxSAT is an appropriate exact method to be used as part of LNS.

In addition, we note two other very important points for our maxSAT choice: this is the first time, to the best of our knowledge, that maxSAT has been used in a LNS algorithm. Second, Gurobi (the commercial IP solver used in [12]) is a highly engineered piece of software, while open-WBO (the maxSAT solver that we modified) is open source and not so heavily engineered, but still provides competitive results.

6.7 Variants of the algorithm

We experimented with three variants of our algorithm and compared them with our standard algorithm ('LNS' column), still using 1000 seconds of computational time. The results of all three variants are given in Table 2. Each variant is aimed at testing a certain aspect of our algorithm.

Variant LNS.v1 is similar to our standard algorithm, except that it uses a non optimized initial solution as a starting point, generated by ignoring all soft constrains and solving the problem as a SAT problem. Based on the results obtained, we conclude that using optimized initial solutions is better overall, since the local search quickly eliminates simple improvements, leaving more time for the maxSAT solver to identify the more challenging ones.

The second variant LNS.v2 consists of restarting the maxSAT solver between two consecutive calls. In the original version, the maxSAT solver is not restarted from scratch, thus all learned clauses and previous bounds are kept (see Section 5.2). The lower quality of results indicate that restarting the maxSAT solver after every insertion operation is detrimental. We note that the total time required for restarting the solver was not very significant (typically under 50 seconds).

The third variant LNS.v3(R) and LNS.v3(D) consists of using only one of the two proposed neighborhood sets (resource or day vector) in order to analyze whether the combination of neighborhoods is beneficial. Overall these variants seem to be worse than the standard version. We believe that the resource neighborhood vector can provide quick improvements and leave the more complicated ones to the day neighborhood vector. Thus, both neighborhoods complement each other.

7 Conclusion

We introduced an algorithm which uses local search and a novel maxSAT-based LNS approach to solve XHSTT problems. The LNS part consists of two main components: the destroy and insertion operations. The first operator destroys the part of the solution based on a neighborhood selection from one of the two neighborhood vectors, while the second operator repairs the solution using exhaustive

[3] http://www.dbai.tuwien.ac.at/user/demir/WCNF_SNA.rar

search in the form of maxSAT. During the course of the algorithm, information regarding the neighborhoods is kept in order to avoid selecting neighborhoods which are most likely to be of no use.

The proposed approach outperforms state-of-the-art solvers on most instances which we were able to model with maxSAT. In addition, new upper bounds have been obtained on four instances. We have also discussed variants of our algorithm to gain further insights into its inner workings. Our approach is highly effective and we believe that it is a good direction for further research on XHSTT. Finally, we think that our maxSAT-based framework can be used in other domains as well. However, problem specific neighborhood vectors need to be introduced and a particular maxSAT formulation for the problem at hand has to be developed.

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

Name	LNS(1000s)	LNS(time)	Best	LNS.v1	LNS.v2	LNS.v3(R)	LNS.v3(D)
Brazil2	5	-	5	5.4	5	12.6	5
Brazil4	61.4	53 (4000)	51	60.6	69	91	88.2
Brazil6	50.6	**34 (27000)**	35	99	75	56.8	74.6
GreecePatras10	0	-	0	57.8	6	0.5	0
GreeceUni4	5	-	5	38.8	6	13	5
GreeceHSchool	0	-	0	0	0	0	0
Italy4	35	35 (3500)	34	$-^m$	52	48	55
SAfricaWood	1.2	-	0	1.2	0.4	27.8	0
SAfricaLewitt	-	-	0	-	-	-	-
FinlandHSchool	9.8	8 (3200)	1	37.2	15.8	14.6	23.2
FinlandCollege	54.6	23 (4000)	0	43.4	84.8	50	643.8
FinlandSSchool	95.2	82 (9000)	77	92.2	104	117	126
KosovaInst	-	**0 (300)**	3	-	-	-	-
Brazil1	39	38 (10000)	38	38.6	38	41.8	38.8
Brazil3	23	-	23	23.6	23	61	23.2
Brazil5	19.4	**17 (20000)**	20	23.4	27.6	43.6	27.6
Brazil7	136.2	**57 (14000)**	67	282	176	114.4	217.6
GreecePreveza	38.2	0 (1200)	0	39.6	69	168.8	5.5
GreeceUni3	7	-	5	8	6	10	7.2
GreeceUni5	0	-	0	0	0	1.2	0
GreeceAigio	368	97 (5300)	0	$-^m$	442	461.2	200
Italy1	12	-	12	12.2	12.2	14	12
FinlandSSchool2	0.2	-	0	76.8	1	5	0
FinlandESchool	3	-	3	3	3	3	3
FinlandASchool	0	-	0	0	0	30	0

Table 2. Comparisons of results for longer running times, different variants of our algorithm, and previously best known upper bounds. The new best known upper bounds computed are in bold.

Name	maxSAT(30s)	IP(30s)	maxSAT (600s)	IP (600s)
Brazil2	**98**	(16, 293)	**55**	87
Brazil4	**261**	(55, 266)	**199**	241
Brazil6	**505**	(162, 580)	**341**	609
GreecePatras	**3219**	-	3219	**25**
GreeceUni4	**186**	193	141	**55**
Italy4	**18671**	(2387, 26756)	**18671**	21228
SAWoodlands	**4028**	-	**811**	-
SALewitt	**1189**	-	1189	**802**
FinlandHSchool	**1231**	(331, 981)	501	**351**
FinlandCollege	**1056**	-	**1056**	(546, 1185)
FinlandSSchool	**765**	(30, 1053)	575	**166**
Brazil1	**44**	59	41	41
Brazil3	**171**	(77, 407)	**65**	93
Brazil5	**406**	(118, 487)	**225**	585
Brazil7	**819**	(240, 887)	**771**	(157, 908)
Italy1	2039	**141**	27	**17**
FinlandSSchool2	**2464**	-	**2464**	(279, 3483)
FinlandESchool	**3**	-	**3**	4
FinlandASchool	**4004**	(416, 125)	**43**	(202, 111)
GreecePreveza	**5617**	-	**5617**	(18, 3145)
GreeceUni3	161	183	122	**37**
GreeceUni5	**100**	52	**32**	37
GreeceAigio	**4582**	-	**4582**	(6609, 195)

Table 3. Comparison of Integer Programming [12] and pure maxSAT for XHSTT. The hypen in the table indicates that the solver was not able to produce a solution within the specified time limit.

References

1. Biere, A., Heule, M., van Maaren, H., and Walsh, T., Eds. *Handbook of Satisfiability* (2009), vol. 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press.
2. Brito, S. S., Fonseca, G. H. G., Toffolo, T. A. M., Santos, H. G., and Souza, M. J. F. A SA-ILS approach for the high school timetabling problem. *Electronic Notes in Discrete Mathematics 39* (2012), 169–176.
3. Demirović, E., and Musliu, N. Modeling high school timetabling as partial-weighted maxsat. *LaSh 2014: The 4th Workshop on Logic and Search (a SAT / ICLP workshop at FLoC 2014)* (2014).
4. Demirović, E., and Musliu, N. Modeling High School Timetabling with Bitvectors. *Annals of Operations Research, accepted for publication* (2016).
5. Dorneles, Á. P., de Araujo, O. C. B., and Buriol, L. S. A fix-and-optimize heuristic for the high school timetabling problem. *Computers & OR 52* (2014), 29–38.
6. Even, S., Itai, A., and Shamir, A. On the complexity of time table and multi-commodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on* (1975), IEEE, pp. 184–193.
7. Fonseca, G. H., Santos, H. G., and Carrano, E. G. Late acceptance hill-climbing for high school timetabling. *Journal of Scheduling* (2015), 1–13.
8. Fonseca, G. H. G., and Santos, H. G. Variable neighborhood search based algorithms for high school timetabling. *Computers & OR 52* (2014), 203–208.
9. Kheiri, A., Ozcan, E., and 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. KHE14: An algorithm for high school timetabling. In *Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling, E. Ozcan, E. K. Burke, B. McCollum (Eds.)* (2014), pp. 498–501.
11. Kingston, J. H. Timetable construction: the algorithms and complexity perspective. *Annals of Operations Research* (2012), 1–11.
12. Kristiansen, S., Sørensen, M., and Stidsen, T. R. Integer programming for the generalized high school timetabling problem. *J. Scheduling 18*, 4 (2015), 377–392.
13. Le Berre, D., and Parrain, A. The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation 7* (2010), 59–64.
14. Martins, R., Joshi, S., Manquinho, V. M., and Lynce, I. Incremental cardinality constraints for maxsat. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings* (2014), B. O'Sullivan, Ed., vol. 8656 of *Lecture Notes in Computer Science*, Springer, pp. 531–548.
15. Martins, R., Manquinho, V., and Lynce, I. Open-wbo: a modular maxsat solver. In *Theory and Applications of Satisfiability Testing–SAT 2014*. Springer, 2014, pp. 438–445.
16. Meyers, C., and Orlin, J. B. Very large-scale neighborhood search techniques in timetabling problems. In *Practice and Theory of Automated Timetabling VI*. Springer, 2007, pp. 24–39.
17. Post, G., Ahmadi, S., Daskalaki, S., Kingston, J., Kyngas, J., Nurmi, C., and Ranson, D. An XML format for benchmarks in high school timetabling. *Annals of Operations Research 194*, 1 (2012), 385–397.

18. Post, G., Di Gaspero, L., Kingston, J., McCollum, B., and Schaerf, A. The third international timetabling competition. *Annals of Operations Research* (2013), 1–7.
19. Post, G., Kingston, J. H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngäs, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., and Schaerf, A. XHSTT: an XML archive for high school timetabling problems in different countries. *Annals OR 218*, 1 (2014), 295–301.
20. Santos, H. G., Uchoa, E., Ochi, L. S., and Maculan, N. Strong bounds with cut and column generation for class-teacher timetabling. *Annals of Operations Research 194*, 1 (2012), 399–412.
21. Shaw, P. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings* (1998), M. J. Maher and J. Puget, Eds., vol. 1520 of *Lecture Notes in Computer Science*, Springer, pp. 417–431.
22. Silva, J. P. M., Lynce, I., and Malik, S. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*. 2009, pp. 131–153.
23. Sinz, C. Towards an optimal CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, P. van Beek, Ed., vol. 3709 of *Lecture Notes in Computer Science*. Springer, 2005, pp. 827–831.
24. Sørensen, M., and Dahms, F. H. A two-stage decomposition of high school timetabling applied to cases in Denmark. *Computers & Operations Research 43* (2014), 36–49.
25. Sørensen, M., Kristiansen, S., and 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)* (2012), p. 489.
26. Sørensen, M., and Stidsen, T. R. High School Timetabling: Modeling and solving a large number of cases in Denmark. In *9th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)* (2012), pp. 359–364.
27. Sørensen, M., and Stidsen, T. R. Comparing solution approaches for a complete model of high school timetabling. Tech. rep., DTU Management Engineering, 2013.
28. Sørensen, M., and Stidsen, T. R. Hybridizing integer programming and metaheuristics for solving high school timetabling. In *Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling, E. Ozcan, E. K. Burke, B. McCollum (Eds.)* (2014), pp. 557–560.