EasyChair Preprint

№ 2821

Minimal Perturbation in University Timetabling with Maximum Satisfiability

Alexandre Lemos, Pedro T. Monteiro and Inês Lynce

February 29, 2020

Minimal Perturbation in University Timetabling with Maximum Satisfiability*

Alexandre Lemos[0000−0002−3876−1011], Pedro T. Monteiro[0000−0002−7934−5495], and Inês Lynce[0000−0003−4868−415X]

Instituto Superior Técnico, Universidade de Lisboa
INESC-ID, Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{alexandre.lemos,pedro.tiago.monteiro,ines.lynce}@tecnico.ulisboa.pt

Abstract. Every new academic year, scheduling new timetables due to disruptions is a major problem for universities. However, computing a new timetable from scratch may be unnecessarily expensive. Furthermore, this process may produce a significantly different timetable which in many cases is undesirable for all parties involved. For this reason, we aim to find a new feasible timetable while minimizing the number of perturbations relative to the original disrupted timetable.
The contribution of this paper is a maximum satisfiability (MaxSAT) encoding to solve large and complex university timetabling problem instances which can be subject to disruptions. To validate the MaxSAT encoding, we evaluate university timetabling real-world instances from the International Timetabling Competition (ITC) 2019. We consider the originally found solutions as a starting point, to evaluate the capacity of the proposed MaxSAT encoding to find a new solution with minimal perturbation. Overall, our model is able to efficiently solve the disrupted instances.

Keywords: MaxSAT · University Course Timetabling · Minimal Perturbation

1 Introduction

Many real-life problems can be encoded as constraint optimization problems, being university timetabling problems a concrete example. Solving optimization problems is by itself a hard and complex computational task. When solving these problems, unexpected disruptions may cause the original solution to be no longer feasible. Therefore, one needs to solve the problem again subject to these unexpected disruptions. Universities, and in particular their timetables, are dynamical systems. Hence, it is natural that one often needs to solve new

* The authors would like to thank the reviewers for their helpful comments and suggestions that contributed to an improved manuscript. This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference SFRH/BD/143212/2019 (PhD grant), DSAIPA/AI/0033/2019 (project LAIfeBlood) and UIDB/50021/2020 (INESC-ID multi-annual funding).

timetables subject to disruptions. These types of real-world scenarios are still a significant research line [1,2].

The contribution of this paper is a MaxSAT encoding to solve university course timetabling problems which can be subject to different disruptions. We showcase the application of the MaxSAT encoding with the large data sets from the ITC-2019 benchmark[3]. Furthermore, these instances are subject to the most common disruptions in the literature.

This paper is organized as follows. Section 2 provides a concise background on university timetabling and minimal perturbation problems. Section 3 formally describes the problem of minimal perturbation in university timetabling and the MaxSAT encoding. Section 4 discusses the main computational results. Finally, Section 5 concludes the paper and discusses possible future directions.

2 Background

In this section, we provide an overview of university timetabling, followed by the background on the minimum perturbation problem and the MaxSAT problem.

2.1 University Timetabling

University timetabling problems [1,2] can be categorized as follows: examination timetabling [4], course timetabling [5] and student sectioning [6]. These problems are known to be NP-complete [7].

Examination timetabling is the problem of assigning exams to rooms subject to a set of constraints. Course timetabling can be informally defined as the problem of finding a feasible assignment for all the classes of all courses to a time slot and a room, subject to a set of constraints. Student sectioning is the problem of sectioning students, subject to capacity and schedule constraints, to all the classes required by the courses they are enrolled in. In the context of this paper, we only consider course timetabling and student sectioning problems. A formal and detailed description of both problems is given in Section 3.

In recent years, a significant improvement in solving university timetabling problems has been achieved [1,2]. In the literature, one can find distinct approaches to solve university timetabling problems, most notably: Constraint Programming (CP)[8,9], Answer Set Programming (ASP) [10], Boolean Satisfiability (SAT) [11], Maximum Satisfiability (MaxSAT) [12], Integer Linear Programming (ILP) [13,14,15] and local search [13,16].

The availability of benchmark data sets from previous competitions [5], based on data from Udine University, motivated the development of the above mentioned methods. However, a gap between theory and practice [1,3] still persists, given that the benchmark does not express the whole complexity and size of the worldwide university timetabling problem. Recently, to further reduce this gap, a new benchmark was made available as part of ITC-2019 [3].

2.2 Minimal Perturbation Problem

Consider a given problem, subject to a set of constraints, for which s is a feasible solution. A set of disruptions may imply a change in the set of constraints and/or a change in the set of variables of the problem. The disruptions cause the solution s to be no longer feasible.

This optimization problem can be described as a Minimal Perturbation Problem (MPP) [8,10,14,17,15] where the goal is to minimize the number of perturbations caused to s in order to find a new feasible solution. In this paper, we consider the MPP as a multi-objective optimization problem where we use the Hamming distance (HD) and the overall quality as the optimization criterion. This makes MPP cardinality minimal and so more restricted than subset minimal. The problem of finding similar/diverse solutions [18] has similarities to MPP. However, the task of finding similar/diverse solutions usually does not consider an infeasible solution as a starting point.

Example 1. Let us consider a course timetabling problem instance with two classes (c_i and c_j) that can be assigned to five different time slots denoted as $t_1 \dots t_5$. Time slots $t_3 \dots t_5$ have a penalty associated with both classes. Classes c_1 and c_2 have a *no overlap* constraint, to ensure that they are assigned to different time slots. Also, let us assume that the original solution s is optimal and consists in the assignment of c_i to the time slot t_1 and c_j to t_2. Now, if a disruption causes t_1 to be unavailable to class c_i, then solution s becomes infeasible, and needs to be modified. If one solves the problem instance from scratch, the optimal solution is the assignment of c_i to time slot t_2 and c_j to t_1, corresponding to a different solution. The solution with the smallest number of perturbations only implies changing c_i to time slot t_2 despite the fact that it causes a loss in the overall quality of the timetable.

The application of MPP to course timetabling has been studied in the literature [8,10,14,15]. The most common approach to measure the perturbations is to apply the HD [19].

Müller *et al.* [8] proposed the iterative forward search algorithm, a local search method that does not ensure completeness. Phillips *et al.* [15] proposed a neighborhood based integer programming algorithm to solve MPP in instances from the University of Auckland. In the worst case, the neighborhood will include the whole search space.

Recently, two different tools have been proposed to compute the Pareto front using ASP [10] and ILP [14]. The Pareto front is computed based on two objectives: (i) the minimization of the cost of unsatisfied soft constraints; and (ii) the minimization of the number of perturbations.

Another approach is to create a robust solution in order to resist predictable disruptions [16]. However, this approach will not be discussed in this paper.

2.3 MaxSAT

A literal l, is either a Boolean variable x (positive literal) or its negation $\neg x$ (negative literal). A clause is a disjunction of literals. A propositional formula

in Conjunctive Normal Form (CNF) is a conjunction of clauses. SAT is the problem of deciding whether a given formula has an assignment that satisfies all the clauses in the formula.

The MaxSAT problem is a generalization of SAT, where the objective is to find an assignment that maximizes the number of satisfied clauses. A weighted partial MaxSAT formula ($\varphi = \varphi_h \cup \varphi_s$) consists of hard clauses (φ_h), soft clauses (φ_s), and a function $w^{\varphi} : \varphi_s \rightarrow \mathbb{N}$ associating an integer cost to each soft clause. The goal in weighted partial MaxSAT is to find an assignment such that all hard clauses in φ_h are satisfied, while maximizing the weight of the satisfied soft clauses in φ_s.

In this paper, we will assume that all propositional formulas are in CNF. However, to simplify the writing of some constraints, we will use the definition of pseudo-Boolean (PB) constraints. PB constraints are commonly applied in pseudo-Boolean optimization [20], a related problem to weighted partial MaxSAT. PB constraints are linear constraints over Boolean variables, and can be generally written as follows: $\sum q_i x_i \ \text{OP} \ K$, where K and all q_i are integer constants, all x_i are Boolean variables, and $\text{OP} \in \{<, \leq, =, \geq, >\}$. This type of constraints can be translated into SAT [21].

3 MaxSAT encoding

In this section, we formally describe the university course timetabling problem [3] and its MaxSAT encoding. Consider a set of consecutive time slots of five minutes $T \in \{1, ..., 288\}$ corresponding to all possible time slots of a day and a set of sets of weekdays $\mathcal{D} \in \{0000000, \ldots, 1111111\}$. Each subset of days $Days \in \mathcal{D}$ has $|Days| = 7$. $Days^d$ corresponds to a particular weekday with $0 < d \leq |Days|$ ($Days^1$ corresponds to Monday, $Days^2$ to Tuesday, and so on). A set of sets of weeks of a semester is represented by $\mathcal{W}$. Each subset of weeks $Weeks \in \mathcal{W}$ has $|Weeks| = 16$. $Weeks^w$ corresponds to week w with $0 < w \leq |Weeks|$. A time period p is represented with a 4-tuple (W_p, D_p, h_p, len_p): a set of weeks ($W_p \subseteq \mathcal{W}$); a set of days ($D_p \subseteq \mathcal{D}$); an hour ($h_p \in T$); and its duration ($len_p > 1$).

Consider a set of courses Co. A course ($co \in Co$) is composed by a set of classes C_{co}. These classes are characterized by configurations ($Config_{co}$) and organized in parts ($Parts_{config}$). A student must attend the classes from a single configuration. A student enrolled in the course co and attending the configuration $config \in Config_{co}$ must attend *exactly-one* class from each part $Parts_{config}$. The set of classes belonging to $part \in Parts_{config}$ is represented by C_{part}.

The university has a set R of rooms where the classes of a course can be scheduled. The travel time, in slots, between two rooms $r_1 \in R$ and $r_2 \in R$ is represented as $travel_{r_2}^{r_1}$. Each room $r \in R$ has a set of unavailable periods P_r.

All university classes C (from different courses) must have a schedule assigned to them. Each class $c \in C$ has a set of possible periods (P_c) to be scheduled in. Each possible period $p \in P_c$ has an associated penalty. Furthermore, a class may need to be assigned to a room. A class has a hard limit on the number of students that can attend it (lim_c). A class may have a set of possible rooms

(R_c). Each room $r \in R_c$ has $capacity \geq lim_c$ and an associated penalty. Each class may also have parent-child relation with another class. The parent of class c is represented by $parent_c$.

The university has a set of students S. Each student $s \in S$ is enrolled in a set of courses Co_s. To reduce the number of similar variables and constraints, we create groups of students sharing the same curricular plan [22]. Furthermore, we limit the size of the group to the value of the greatest common divisor between the total number of students enrolled in a course and the smallest capacity limit of the classes of that course [23]. This process ensures that it is possible to find a feasible solution to a problem instance, since it is possible to combine all groups of students into classes. However, we may remove the optimal solution by not allowing the assignment of a single student to a given class. For this reason, we define $\mathcal{C}luster$ as a set of clusters of students. The number of students merged in the $id \in \mathcal{C}luster$ is represented by $|id|$.

There are four optimization criteria: (i) the cost of assigning a class to a room; (ii) the cost of assigning a class to a time slot; (iii) the number of student conflicts and (iv) a set of soft constraints. Each criterion has its weights. We solve university course timetabling in two sequential MaxSAT runs. First, we solve the course timetabling problem and then we solve the student sectioning problem. The sequential runs may result in the loss of the global optimum (*i.e.* it may remove the optimal solution in terms of student conflicts). Nevertheless, it produces a solution within the Pareto front if given enough time and memory resources. Moreover, it reduces the size of the global problem. Furthermore, allows us to tackle the MPP using only the first MaxSAT model.

3.1 Course timetabling

Our course timetabling encoding has four types of Boolean decision variables:

- $w_c^{Week_p}$ represents the assignment of class c to the set of weeks $Week_p$, with $c \in C$, $Week_p \in \mathcal{W}$ and $p \in P_c$;
- $d_c^{Day_p}$ represents the assignment of class c to the set of days Day_p, with $c \in C$, $Day_p \in \mathcal{D}$ and $p \in P_c$;
- $h_c^{hour_p}$ represents the assignment of class c to the hour $hour_p$, with $c \in C$ and $p \in P_c$;
- r_c^{room} represents the assignment of class c to the room $room$, with $c \in C$ and $room \in R_c$.

The scheduling possibilities of a class are usually just a small part of the complete set of possible combinations of weeks, days and hours. Consequently, we only define these variables for acceptable values of the class domain reducing the size of the problem. Furthermore, using four variables instead of one provides a more flexible approach when writing the associated constraints, reducing the size of the encoding. For example, one can write the constraints using only related variables (*e.g. SameDay* constraint uses only variable d).

To simplify the writing of the *exactly-one* constraints ($\sum \cdot = 1$) we define the auxiliary variable t, where t_c^{slot} represents the assignment of class c to the allocation slot $slot \in [0, \ldots, |P_c|]$.

Our encoding has the following constraints. If a class c takes place in the hour *hour* then all allocation slots including *hour* are assigned. If we consider that n allocation slots have the same *hour*, then the following equivalence is needed:

$$h_c^{hour} \iff \bigwedge_n t_c^{slot_n}. \tag{1}$$

This equivalence can be easily converted to SAT. Similarly, the same type of equivalence has to be written between the week/day variables and the t variables.

A class can only be taught in exactly one allocation slot. For each class $c \in C$:

$$\sum_{slot \in [0,\ldots,|P_c|]} t_c^{slot} = 1. \tag{2}$$

A class with $R_c \neq \emptyset$ can only be taught in exactly one room. For each $c \in C$:

$$\sum_{room \in R_c} r_c^{room} = 1. \tag{3}$$

We define the auxiliary variable $sd_{c_j}^{c_i}$ to represent two classes taught in the same day (*i.e.* with at least one day overlap). For each two classes c_i, c_j with $i \neq j$, where Day_0 to Day_n belong to the domain of class c_i, Day_{n+1} to Day_m belong to the domain of class c_j, with $0 < n < m$, and they overlap we add:

$$sd_{c_j}^{c_i} \iff (d_{c_i}^{Day_0} \vee \ldots \vee d_{c_i}^{Day_n}) \wedge (d_{c_j}^{Day_{n+1}} \vee \ldots \vee d_{c_j}^{Day_m}). \tag{4}$$

Similarly, one can define an auxiliary variable $sw_{c_j}^{c_i}$ to represent two classes overlapping in at least one week.

A class c with $R_c \neq \emptyset$ must be taught in a room not assigned to another class in the specific time slot. For each two classes c_i, c_j, where $room \in R_{c_i}$, $room \in R_{c_j}$, $hour_{p_i} + len_{p_i} > hour_{p_j}$ and $hour_{p_j} + len_{p_j} > hour_{p_i}$ with $p_i \in P_{c_i}$ and $p_j \in P_{c_j}$, we add clause:

$$\neg sd_{c_j}^{c_i} \vee \neg sw_{c_j}^{c_i} \vee \neg h_{c_i}^{hour_{p_i}} \vee \neg h_{c_j}^{hour_{p_j}} \vee \neg r_{c_i}^{room} \vee \neg r_{c_j}^{room}. \tag{5}$$

The clause above could have a smaller number of literals if we used the auxiliary variable t. However, it would require to generate more constraints. This trade-off was tested and fewer constraints proved to be more efficient.

The rooms may have unavailability time slots, where no class can be taught. To enforce this constraint we add the following clause for each class c, room r and unavailable period p:

$$\neg r_c^r \vee \neg t_c^p. \tag{6}$$

The next set of constraints can be hard or soft. These constraints involve always a pair of classes. In case of being soft, the penalty associated with each

constraint incurs for every pair of classes. Consider two classes c_i and c_j with $i \neq j$ and two time slots $p_i \in P_{c_i}$ and $p_j \in P_{c_j}$.

SameStart: The classes have to start at the same time. For each pair $hour_{p_i}$, $hour_{p_j}$ where $hour_{p_i} \neq hour_{p_j}$ we add a clause:

$$\neg h_{c_i}^{hour_{p_i}} \vee \neg h_{c_j}^{hour_{p_j}}. \tag{7}$$

DifferentTime (*SameTime*): The classes must be taught at a (the) different (same) hour. For each pair $hour_{p_i}$, $hour_{p_j}$ with (no) overlap in time, we add (7).

WorkDay(V): There must not be more than V time slots between the start time of the first class and the end time of the last class on any day. For each pair $hour_{p_i}$, $hour_{p_j}$ where $hour_{p_i} + len_{p_i} - hour_{p_j} \geq V$, we add clause (7).

DifferentDays (*SameDays*): The classes must be taught in different days (the same subset of days). For each pair Day_{p_i}, Day_{p_j} where $Day_{p_i} \wedge Day_{p_j} = \emptyset$ ($Day_{p_i} \subseteq Day_{p_j}$), we add a clause:

$$\neg d_{c_i}^{Day_{p_i}} \vee \neg d_{c_j}^{Day_{p_j}}. \tag{8}$$

DifferentWeeks (*SameWeeks*): The classes must be taught in different weeks (the same subset of weeks). For each pair $Week_{p_i}$, $Week_{p_j}$ where $Week_{p_i} \wedge Week_{p_j} = \emptyset$ ($Week_{p_i} \subseteq Week_{p_j}$), we add a clause:

$$\neg w_{c_i}^{Week_{p_i}} \vee \neg w_{c_j}^{Week_{p_j}}. \tag{9}$$

DifferentRoom (*SameRoom*): The classes must be taught in different rooms (the same room). For each pair $room_i \in R_{c_i}$, $room_j \in R_{c_j}$ where $room_i = room_j$ ($room_i \neq room_j$), we add a clause:

$$\neg r_{c_i}^{room_i} \vee \neg r_{c_j}^{room_j}. \tag{10}$$

SameAttendees: The classes cannot overlap in time, days and weeks. Furthermore, the attendees must have sufficient time to travel between the rooms corresponding to consecutive classes. For each pair of hours $hour_{p_i}$, $hour_{p_j}$ and rooms $room_i$, $room_j$ where and $hour_{p_i} + len_{p_i} + travel_{room_j}^{room_i} > hour_{p_j}$, we add:

$$\neg sd_{c_j}^{c_i} \vee \neg sw_{c_j}^{c_i} \vee \neg h_{c_i}^{hour_{p_i}} \vee \neg h_{c_j}^{hour_{p_j}} \vee \neg r_{c_i}^{room_i} \vee \neg r_{c_j}^{room_j}. \tag{11}$$

Overlap (*NotOverlap*): The classes must (not) overlap in time, day and week. For each pair p_i, p_j with (no) overlaps in time, we add a clause:

$$\neg t_{c_i}^{p_i} \vee \neg t_{c_j}^{p_j}. \tag{12}$$

Precedence: The first meeting of a class in a week must be before the first meeting of another class. For each pair p_i, p_j where p_j preceeds p_i, we add (12).

MinGap(V): The classes that are taught on the same day and on the same set of weeks must be at least V slots apart. For each pair p_i, p_j that is taught in the same week, day and $hour_{p_i} + len_{p_i} + V \geq hour_{p_j}$, we add (12).

The next set of constraints involve a set of classes. In these cases, the penalty depends on the distance between the solution and the unsatisfied constraint.

MaxDays(V): The classes cannot be taught in more than V different days. When the constraint is soft, the penalty is multiplied by the number of days that exceed V. For this reason, we define an auxiliary variable $dayofweek_d^{const}$, where *const* is the identifier of the constraint *MaxDays* and $d \in \{1, \ldots, |Days|\}$. This variable corresponds to having at least one class, of this constraint, assigned to weekday d. Consider $Day_{p_1}, \ldots, Day_{p_n}$, $Day_{p_{n+1}}, \ldots, Day_{p_m}$ where $p_1, \ldots, p_n \in P_{c_i}$, $p_{n+1}, \ldots, p_m \in P_{c_j}$ and $Day_1^d = 1, \ldots, Day_m^d = 1$ we add:

$$dayofweek_d^{const} \iff d_{c_i}^{Day_{p_1}} \vee \ldots \vee d_{c_i}^{Day_{p_n}} \vee d_{c_j}^{Day_{p_{n+1}}} \vee \ldots \vee d_{c_j}^{Day_{p_m}}. \quad (13)$$

Now, we only need to ensure that:

$$\sum_{c \in C} \sum_{p \in P_c} \sum_{d \in [1,\ldots,|Day_p|]} dayofweek_d^{const} \leq V. \quad (14)$$

MaxDayLoad(V): The classes must (should, if the constraint is soft) be spread over the days in a way that there is no more than a given number of occupied V time slots on each day. When the constraint is soft, the penalty is multiplied by the division of the sum of the number of slots that exceed V for each day by the number of weeks. Hence, we only need to ensure that:

$$\sum_{d=1}^{7} \sum_{c \in C} \sum_{\substack{p \in P_c, \\ Day_p^d = 1}} d_c^{Day_p} \times len_p \leq V. \quad (15)$$

MaxBreaks(V_1,V_2): There are at most V_1 breaks throughout a day between a set of classes in this constraint. A break between two classes is a gap larger than V_2 time slots. When the constraint is soft, the penalty is multiplied by the number of new breaks. For every class c_1 to c_n assigned to a period ($p_1 \in P_{c_1}$ to $p_n \in P_{c_n}$) in such a way that it forms a block of classes that breaks this constraint, we add the clause:

$$\neg l_{c_1}^{p_1} \vee \ldots \vee \neg l_{c_n}^{p_n}. \quad (16)$$

MaxBlock(V_1,V_2): There are at most V_1 consecutive slots throughout a day between a set of classes in this constraint. Two classes are considered to be consecutive if the gap between them is less than V_2 time slots. When the constraint is soft, the penalty is multiplied by the number of new blocks of classes. For every class c_1 to c_n assigned to a period ($p_1 \in P_{c_1}$ to $p_n \in P_{c_n}$) in such a way that it forms a block of classes that breaks this constraint, we add (16).

3.2 Student Sectioning

To solve student sectioning our encoding is extended with one decision variable s_{id}^c, where $c \in C$ and $id \in [1, \ldots, |\mathcal{C}luster|]$. To ensure a student can only be

sectioned to a single course configuration, we define an auxiliary variable for each pair configuration-cluster of students. The variable is denoted as $conf_{id}^{config}$, where $id \in [1, \ldots, |\mathcal{C}luster|]$, $config \in Config_{co}$ and $co \in Co$.

Each cluster of students id must be enrolled in *exactly-one* configuration of each course, $co \in Co_s$, and thus we add the clause:

$$\sum_{config \in Config_{co}} conf_{id}^{config} = 1. \tag{17}$$

To ensure that the class capacity is not exceed, we add for each class c:

$$\sum_{id \in [1,\ldots,|\mathcal{C}luster|]} |id| \times s_{id}^{c} \leq lim_c. \tag{18}$$

A cluster of students id enrolled in a class c must be enrolled in class $parent_c$:

$$\neg s_{id}^{c} \vee s_{id}^{parent_c}. \tag{19}$$

Finally, we need to ensure that a cluster of students id is enrolled in *exactly-one* class of each part of a single configuration of the course co. Therefore, for each cluster of student id and for each pair of two classes c_i, c_j in the same $c_i, c_j \in C_{part}$ where $part \in Parts_{config}$, we add:

$$\neg conf_{id}^{config} \vee \neg s_{id}^{c_i} \vee \neg s_{id}^{c_j}. \tag{20}$$

For each cluster of students and for each $part \in Parts_{config}$ we add:

$$\neg conf_{id}^{config} \vee s_{id}^{c_i} \vee \ldots \vee s_{id}^{c_{|C_{part}|}}. \tag{21}$$

The conflicting schedule of classes attended by the same cluster of students is represented by a set of weighted soft clauses. For each cluster of students id enrolled in two classes c_i, c_j with overlapping time:

$$\neg s_{id}^{c_i} \vee \neg s_{id}^{c_j} \vee \neg sw_{c_j}^{c_i} \vee \neg sd_{c_j}^{c_i} \vee \neg h_{c_i}^{hour_{c_i}} \vee \neg h_{c_j}^{hour_{c_j}}. \tag{22}$$

3.3 Disruptions

In this work we consider the following disruptions: *invalid time* and *invalid room*. These disruptions reduce the domain of a specific class c in terms of available time slots or rooms. Disruptions in the students enrollments would only cause changes in the student sectioning part. The problem definition has the underlining assumption that all the rooms in the domain of class have enough capacity for the students attending. As our original solutions are sub-optimal we do not consider disruptions in the enrollments.

Invalid Time: The time slot t is no longer available for class c:

$$\neg t_c^t. \tag{23}$$

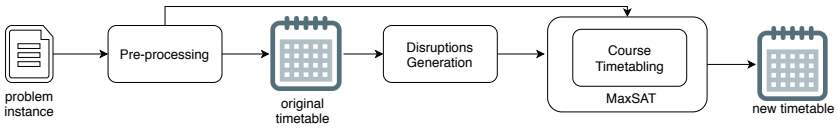


Fig. 1: Algorithm schema to solve university timetabling problems subject to disruptions.

Invalid Room: The room r is no longer available for class c:

$$\neg r_c^r. \tag{24}$$

When recovering from disruptions we apply lexicographic optimization with two criteria: (i) the HD and (ii) the overall quality of the solution (computed based on the four criteria defined above). This way we can take advantage of the disruption to improve the quality of the solution.

4 Experimental Evaluation

In this section, we discuss the main computational results obtained. First, we describe the setup used to validate our approach. Next, we discuss our results for both university timetabling problems and MPP.

4.1 Experimental setup

The evaluation was performed using the *runsolver* tool [24] with a time out of **6,000** seconds. *Runsolver* was run on a computer with Fedora 14, with 32 CPUs at 2.6 GHz and 126 Gb of RAM. To validate our approach, we used the benchmark obtained from ITC-2019 [3], which is divided into three groups (early, middle, late). The goal of the competition was to find the best solution for these instances with no time or memory limits. The organizers provided an validation tool[1], which we used to validate the correctness of our approach.

The proposed solution was implemented in C++, using the *TT-Open-WBO-Inc* [25,26][2] MaxSAT solver. The solver was configured to use linear search with the clusters algorithm [27]. Moreover, a lexicographic optimization criterion [28] was used. *Exactly-one* constraints were encoded into CNF through the ladder encoding [29]. PB constraints were encoded to CNF using the adder encoding [30]. Our implementation is available at github.com/ADDALemos/MPPTimetables[3].

Table 1 shows the different characteristics of the instances. One can see that the instances are distinct from each other. Instances from *iku** are the largest in terms of classes. However, they do not have students or *MaxBlock/MaxBreak*.

[1] `https://www.itc2019.org/validator`

[2] *TT-Open-WBO-Inc* won the Weighted Incomplete category at *MaxSAT Evaluation 2019*. The results are available at `https://maxsat-evaluations.github.io/2019`.

[3] We use the *RAPIDXML* parser which is available at rapidxml.sourceforge.net/

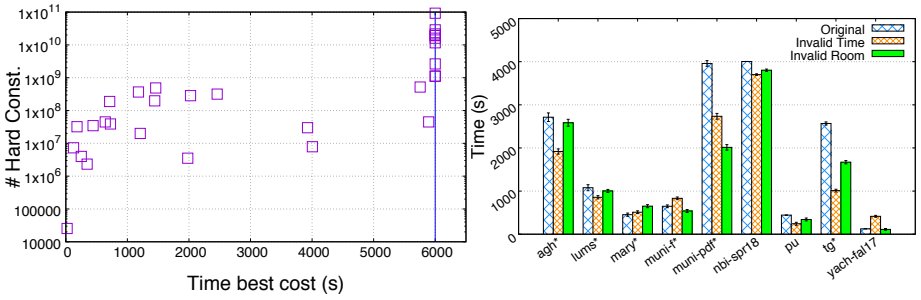
Table 1: Data sets per university (instances sorted by # of variables).

		$\|C\|$	Avg. $\|R_c\|$	Avg. $\|P_c\|$	$\|S\|$ (k)	#MaxBreak	#MaxBlock	# Var. (k)
yach-fal17		417	4	43	1	0	0	19
nbi-spr18		782	4	38	2	0	0	35
tg*	Avg.	693	11	24	0	0	0	42
	Median	693	11	24	0	0	0	42
mun-f*	Avg.	743	4	44	1	0	3	45
	Med.	700	4	30	1	0	2.5	38
mary*	Avg.	916	14	12	4	0	0	47
	Med.	916	14	12	4	0	0	47
lums*	Avg.	494	26	43	0	0	0	82
	Med.	494	26	43	0	0	0	82
bet*	Avg.	1,033	25	23	3	24	19	140
	Med.	1,033	25	23	3	24	19	140
pu*	Avg.	3,418	12	33	28	16	0	196
	Med.	1,929	12	30	31	17	0	125
muni-pdf*	Avg.	2,586	15	53	4	0	13	374
	Med.	2,526	17	56	3	0	10	373
agh*	Avg.	1,955	34	89	3	15	0	380
	Med.	1,239	10	75	2	14	0	340
iku*	Avg.	2,711	25	34	0	0	0	1,050
	Med.	2,711	25	34	0	0	0	1,050

They have one order of magnitude more variables than the next largest instance (despite not having students). The *muni-f** instances have a particular small search space in terms of possible rooms per class (only 4).

Figure 1 illustrates the process of solving the university timetabling problem subject to disruptions. The process starts with a problem instance and a timetable, and ends when a new feasible timetable is found. Each problem instance is pre-processed before generating the encoding. Our approach relies on two pre-processing methods: (i) identification of independent sub-sets in terms of courses; and (ii) merging students with exactly the same course enrollment plan. Method (i) divides the problem into self-contained sub-problems, while not removing any solution. Method (ii) was already discussed before (Section 3) and it may remove the optimal solution by not allowing the assignment of an individual student to a given class.

For each instance, we generated 50 different disruption instances. As our space is limited, we only show the results for the disruptions that are more likely to occur [14,15]. The disruptions were randomly generated following a uniform distribution with a probability of 21% and 25% for *invalid time* and *invalid room*, respectively. These percentages represent the probability of an assignment being invalid. These values were obtained by the academic office of our university, and applied to the ITC-2019 benchmark instances.



(a) (b)

Fig. 2: (a) A comparison of the number of hard constraints versus CPU time, in seconds (log scale), for each instance. The dotted line represents the time limit. (b) A comparison of the CPU time per disruption scenario and university.

4.2 Computational results

First, we discuss the results for the university course timetabling without subjecting it to disruptions. Next, we discuss the results for the MPP.

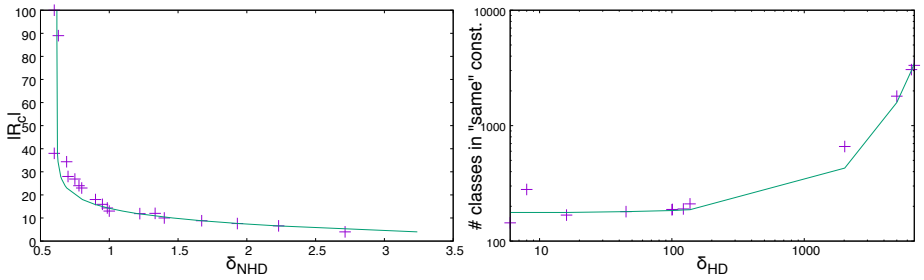
University Course Timetabling The success of our approach is attested by having been ranked among the five finalists of the ITC-2019 competition. The creation of clusters has a significant impact on the number of variables. On average, one can reduce the number of variables relating to students up to 15%.

We are able to find a solution within limit in 20 out of 30 instances. However, the solver was not able to prove optimality within the time limit, on any of the instances considered in this paper. Note that, for most instances, the solver requires only a short amount of time to produce the best solution.

Figure 2(a) compares the number of hard constraints generated by the CNF encoding and the CPU time needed to find the best solution for each instance. One can see that most of these instances have a larger number of hard constraints. Most of them actually exceed in two orders of magnitude more constraints than the others (top right corner of Figure 2(a)). Most of these constraints result from the *MaxBlock* and *MaxBreak* constraints.

A large contributing factor for these results is the size of $|P_c|$ and $|R_c|$ (see Table 1). In most cases, a larger size of these sets causes the instance to be harder to solve. In these cases, more options do not always mean more solutions. The *lums** instances are an exception.

Minimal Perturbation in University Timetabling Our MaxSAT approach was compared with a modified integer programming approach based on [14]. The results showed that the integer programming approach is able to find the optimal solution for the MPP but only to a subset of instances compared to those solved by the MaxSAT approach. Furthermore, the MaxSAT approach is much faster.



(a) (b)

Fig. 3: (a) Room domain size (R_c) versus the normalized number of perturbations (δ_{NHD}) for the room disruptions. (b) Number of classes involved on constraints of type *same* (log scale) versus the number of perturbations (HD) for the time disruptions (log scale). Data points represent the results and the line the best fit function.

Our approach is able to find feasible solutions to all disruptions tested. Moreover, the solver is able to find an optimal solution for all disrupted instances. Despite the fact that the disruptions only add new constraints, one can occasionally improve the cost of the solution. This can be explained by the fact that our original solutions are sub-optimal. Otherwise, the new solution could only, in the best case scenario, be as good as the original one.

The results for the disrupted instances with *invalid room* and *invalid time* are shown in Tables 2 and 3, respectively. The tables show the average and median required CPU time to find an optimal solution, as well as the distance between the two solutions (δ_{HD}) and the change in the global cost (δ_{cost}). It is important to take into consideration that the value of δ_{HD} is directly linked to the size of the instance due to the process of generating disrupted instances.

Figure 2(b) shows the CPU time, per university, for the instances with and without disruptions. In most cases, less time is needed to solve a problem instance subject to small disruptions than to solve the original problem instance. If the disruptions cause no perturbations in the original solution, then almost no time is needed (only parsing time). However, our disrupted instances were subject to significant disruptions. In most cases, the solver is able to find the optimal solution taking around the same time it took to find the best solution without disruptions. The time spent to find a solution increases with the number of perturbations required.

As one can see in Figure 2(b), the *invalid room* disruptions are, in most cases, easier to sort out than *invalid time* disruptions. The CPU time is smaller since fewer perturbations are needed. The reduction in time can be also explained by the fact that a smaller number of hard constraints are, in fact, related to rooms. The solutions found are, usually, closer to the original one. This can be explained by the fact that most instances have fewer rooms than time slots available.

Table 2: Results for the *Invalid Room* disruption. δ_{HD} measures the number of perturbations and δ_{cost} measures the change in the global quality of the solution.

		Invalid Room					
		Avg. Time (s)	Med. Time (s)	Avg. δ_{HD}	Med. δ_{HD}	Avg. δ_{cost}	Med. δ_{cost}
Early	agh-fis-spr17	1,460.4	1,612.7	22	29	42	39
	agh-ggis-spr17	2,321.2	2,210.8	11	8	0	1
	mary-spr17	231.5	253	25	29	54	55
	muni-fi-spr16	2,133.2	2,317.9	15	18	4	6
	muni-fsps-spr17	812	999.1	13	18	0	0
	muni-pdf-spr16c	4,114.8	4,101.2	42	38	26	21
	pu-llr-spr17	142.5	143	35	36	6	6
	tg-fal17	1,208.8	1,247	100	112	18	19
Middle	agh-ggos-spr17	3,212.6	3,212.9	40	40	640	639
	agh-h-spr17	679.9	699.9	19	20	57	60
	lums-spr18	913.9	921.8	18	18	0	0
	muni-fi-spr17	80.8	99	9	13	36	37
	muni-fsps-spr17c	888.4	977.3	39	44	20	20
	muni-pdf-spr16	1,354.5	1,444.1	89	94	1,335	1,336
	nbi-spr18	3,701.7	3,781	14	13	33	35
	yach-fal17	415.56	420	56	66	84	86
Late	lums-fal17	999.9	1,000.2	20	20	0	0
	mary-fal18	788.9	812.1	20	24	40	42
	tg-spr18	813.8	888	5	8	100	100
	muni-fi-fal17	248.9	250.1	9	10	36	30

The *muni-f** instances are, in most cases, the most difficult instances to solve after *invalid room* disruptions. This can be explained by the fact that these instances are very *tight* in terms of room space. On average, these instances only have 4 possible rooms by class versus an average of 14 in the other instances.

To evaluate the quality of the fittings the following metrics were defined. Root mean square error (RMSE) has a range from 0 to ∞, where the best fit model has a value closer to zero. Coefficient of determination (CD) has a range between 0 and 1, where the best fit model has a value closer to 1. To perform the fitting, we used the Microsoft Excel Solver [31].

Figure 3(a) shows the relation between the room domain size on δ_{HD}. The RMSE of the fit function is 0.04. The CD is 0.95. Note that, for fairness we normalized the value of δ_{HD}. The normalization simply takes into account the number of disruptions generated to the instance (δ_{NHD}).

The *lums** instances are the ones that have the largest δ_{HD} when tested subject to *invalid time* disruptions (see Table 3). This fact can be explained by the large number of constraints forcing the classes to be in the same allocation slot (*SameWeek*, *SameTime*, *SameDay* and *SameStart*). These constraints force a chain of perturbations for a single disruption. Figure 3(b) shows the relation

Table 3: Results for the *Invalid Time* disruption. δ_{HD} measures the number of perturbations and δ_{cost} measures the change in the global quality of the solution.

		Invalid Time					
		Avg. Time (s)	Med. Time (s)	Avg. δ_{HD}	Med. δ_{HD}	Avg. δ_{cost}	Med. δ_{cost}
Early	agh-fis-spr17	1596.22	1711.1	5001	5003	4	6
	agh-ggis-spr17	2358.2	2100.4	4	3	0	3
	mary-spr17	381.2	380.1	0	4	0	6
	muni-fi-spr16	1784.2	1794.2	16	18	0	0
	muni-fsps-spr17	212.4	218.4	45	46	0	0
	muni-pdf-spr16c	2992.1	3001.2	6	6	4	4
	pu-llr-spr17	342.6	356	122	126	10	10
	tg-fal17	1408.7	1484	2021	2070	25	25
Middle	agh-ggos-spr17	5465.8	5466.1	92	93	276	139
	agh-h-spr17	919.1	920.9	97	98	290	289
	lums-spr18	961	978.8	6446	6436	0	0
	muni-fi-spr17	40.12	39	144	140	433	423
	muni-fsps-spr17c	500.3	498.8	137	136	0	0
	muni-pdf-spr16	1035.3	1030	636	630	6363	6364
	nbi-spr18	3803.8	3991.1	164	186	3284	3289
	yach-fal17	112.56	111	100	100	0	4
Late	lums-fal17	1085.58	1100.1	6777	6787	0	0
	mary-fal18	800.12	812.1	269	270	807	900
	tg-spr18	933.2	934	568	559	1704	1705
	muni-fi-fal17	149.2	140.2	101	108	50	51

between of the number of classes involved in constraints of type *Same* on the δ_{HD}. The RMSE of the fit function is 131.8. The CD is 0.11.

5 Conclusion and Future Work

This paper discusses the real-world problem of solving university course timetabling problems which can be subject to disruptions. We propose a MaxSAT encoding to solve course timetabling and student sectioning problems. To validate our approach, we used the ITC-2019 benchmark. The approach is able to solve two thirds of the benchmark instances within the time limit of 6,000 seconds. Moreover, the proposed solution is able to efficiently solve them after the occurrence of the most common disruptions reported in the literature.

As future work, we recommend extending this work to explore the incremental nature of MPP. The application of an incremental algorithm would, in theory, reduce CPU time bypassing the repetition of decisions during the search for a feasible solution. Furthermore, one can study the performance of this implementation using different SAT solvers.

References

1. McCollum, B.: University timetabling: Bridging the gap between research and practice. In: 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 15–35. Springer (2006)
2. Vrielink, R.A.O., Jansen, E.A., Hans, E.W., van Hillegersberg, J.: Practices in timetabling in higher education institutions: a systematic review. Annals of Operations Research **275**(1), 145–160 (2019). https://doi.org/10.1007/s10479-017-2688-8
3. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and international timetabling competition 2019. In: Proceedings of 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT). p. 27 (2018)
4. Müller, T.: ITC-2007 solver description: a hybrid approach. Annals of Operations Research **172**(1), 429 (2009). https://doi.org/10.1007/s10479-009-0644-y
5. Di Gaspero, L., Schaerf, A., McCollum, B.: The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Tech. rep., Queen's University (2007)
6. Laporte, G., Desroches, S.: The problem of assigning students to course sections in a large engineering school. Computers & Operations Research **13**(4), 387 – 394 (1986)
7. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. Society for Industrial and Applied Mathematics Journal on Computing **5**(4), 691–703 (1976). https://doi.org/10.1137/0205048
8. Müller, T., Rudová, H., Barták, R.: Minimal perturbation problem in course timetabling. In: 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 126–146 (2004)
9. Atsuta, M., Nonobe, K., Ibaraki, T.: ITC-2007 track 2: an approach using a general CSP solver. In: 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 19–22 (2008)
10. Banbara, M., Inoue, K., Kaufmann, B., Okimoto, T., Schaub, T., Soh, T., Tamura, N., Wanko, P.: *teaspoon* : Solving the curriculum-based course timetabling problems with Answer Set Programming. Annals of Operations Research **275**(1), 3–37 (2019)
11. Bittner, P.M., Thum, T., Schaefer, I.: SAT encodings of the at-most-k constraint - A case study on configuring university courses. In: Proceedings of the Software Engineering and Formal Methods (SEFM). pp. 127–144 (2019)
12. Achá, R.J.A., Nieuwenhuis, R.: Curriculum-based course timetabling with SAT and MaxSAT. Annals of Operations Research **218**(1), 71–91 (2014)
13. Lemos, A., Melo, F.S., Monteiro, P.T., Lynce, I.: Room usage optimization in timetabling: A case study at Universidade de Lisboa. Operations Research Perspectives **6**, 100092 (2019). https://doi.org/10.1016/j.orp.2018.100092
14. Lindahl, M., Stidsen, T., Sørensen, M.: Quality recovering of university timetables. European Journal of Operational Research **276**(2), 422 – 435 (2019)
15. Phillips, A.E., Walker, C.G., Ehrgott, M., Ryan, D.M.: Integer programming for minimal perturbation problems in university course timetabling. Annals of Operations Research **252**(2), 283–304 (2017). https://doi.org/10.1007/s10479-015-2094-z
16. Gülcü, A., Akkan, C.: Robust university course timetabling problem subject to single and multiple disruptions. European Journal of Operational Research **283**(2), 630 – 646 (2020)

17. Zivan, R., Grubshtein, A., Meisels, A.: Hybrid search for minimal perturbation in dynamic CSPs. Constraints **16**(3), 228–249 (2011). https://doi.org/10.1007/s10601-011-9108-5
18. Hebrard, E., Hnich, B., O'Sullivan, B., Walsh, T.: Finding diverse and similar solutions in constraint programming. In: 20th National Conference on Artificial Intelligence and 17th Innovative Applications of Artificial Intelligence. pp. 372–377 (2005)
19. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal **29**(2), 147–160 (1950). https://doi.org/10.1002/j.1538-7305.1950.tb00463.x
20. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS press (2009)
21. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation **2**(1-4), 1–26 (2006)
22. Carter, M.W.: A comprehensive course timetabling and student scheduling system at the University of Waterloo. In: 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 64–84 (2000)
23. Schindl, D.: Optimal student sectioning on mandatory courses with various sections numbers. Annals of Operations Research **275**(1), 209–221 (2019)
24. Roussel, O.: Controlling a solver execution with the runsolver tool. Journal on Satisfiability, Boolean Modelling and Computation **7**(4), 139–144 (2011)
25. Nadel, A.: Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In: Proceedings of the 19th Conference on Formal Methods in Computer Aided Design (FMCAD) (2019)
26. Nadel, A.: TT-Open-WBO-Inc: Tuning polarity and variable selection for anytime SAT-based optimization. In: Proceedings of the MaxSAT Evaluations (2019)
27. Joshi, S., Kumar, P., Martins, R., Rao, S.: Approximation strategies for incomplete MaxSAT. In: Principles and Practice of Constraint Programming (CP). pp. 219–228 (2018)
28. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Annals of Mathematics and Artificial Intelligence **62**(3-4), 317–343 (2011)
29. Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables into problems with boolean variables. In: Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT). vol. 3542, p. 1–15 (2004)
30. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Information Processing Letters **68**(2), 63–69 (1998)
31. Fylstra, D.H., Lasdon, L.S., Watson, J., Waren, A.D.: Design and use of the Microsoft Excel solver. Interfaces **28**(5), 29–55 (1998), `https://doi.org/10.1287/inte.28.5.29`