Journal of Scheduling (2022) 25:391–404
https://doi.org/10.1007/s10951-021-00715-5

A mixed-integer programming approach for solving university course timetabling problems

Efstratios Rappos[1] · Eric Thiémard[1] · Stephan Robert[1] · Jean-François Hêche[1]

Accepted: 2 November 2021 / Published online: 15 February 2022
© The Author(s) 2022

Abstract
This article presents a mixed-integer programming model for solving the university timetabling problem which considers the allocation of students to classes and the assignment of rooms and time periods to each class. The model was developed as part of our participation in the International Timetabling Competition 2019 and produced a ranking of second place at the competition. Modeling a timetabling problem as a mixed-integer program is not new. Our contribution rests on a number of innovative features adapted to this problem which allow for a reduction in the number of variables and constraints of the mixed-integer program to manageable levels achieving a reasonable computational performance. The proposed algorithm consists of a first-stage method to obtain an initial feasible solution and a second-stage local search procedure to iteratively improve the solution value, both of which involve the optimization of mixed-integer programming problems.

Keywords University course timetabling · ITC 2019 · Timetabling problems · Integer programming · Combinatorial optimization · Matheuristics

1 Introduction

The general university course timetabling problem considers the challenge of allocating university students to classes and assigning days and times to each class. The problem itself is not new, and much research has been devoted for its resolution (Schmidt and Ströhlein 1980; de Werra 1985) even before the advent of personal computers. In its simplest case, the problem considers only the planning of curricula without taking into account the students (Bettinelli et al. 2015; Bonutti et al. 2012), whereas the full problem also considers the allocation of students to individual classes and the production of a personalized timetable for each student. Recent surveys of the literature on timetabling problems, their mathematical modeling and solution approaches include (MirHassani and Habibi 2013; Pillay 2014; Qu et al. 2009; Schaerf 1999), which outline a number of operations research and optimization methods for their solution, and (Hwang et al. 2004; Kingston 2013; Lewis et al. 2007) where some variants of the problem are studied. The full timetabling problem has been the focus of recent research (Müller and Rudová 2016; Rudová et al. 2011) and used for the automated construction of real-life university timetables.

This article considers the university timetabling problem that is specified in the International Timetabling Competition (ITC) 2019 (Müller et al. 2018). The competition defined a timetabling problem where each class has to be assigned a time and, if needed, a room, while at the same time the students have to be grouped into classes. A number of additional distribution constrains are imposed when necessary, and the optimization goal is to minimize the total penalty associated with the solution. The competition consists of 30 real-life university timetabling instances. The problem is challenging as a typical timetable contains hundreds or thousands of classes which are interlinked because they must share the same, often limited, resources such as rooms, lecturers, students and time slots.

Our proposed solution approach consists of modeling the problem as a mixed-integer programming (MIP) model which is too large to solve directly. We then develop two algorithms to firstly obtain a feasible solution and subsequently improve the quality of the solution. The improvement method works by optimizing a sequence of smaller MIPs and as such is closely related to known matheuristics (Dor-

✉ Efstratios Rappos
efstratios.rappos@heig-vd.ch

[1] Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD), University of Applied Sciences of Western Switzerland (HES-SO), Yverdon-les-Bains, Switzerland

neles et al. 2014; Fonseca et al. 2016; Lindahl et al. 2018) and very large-scale neighborhood search algorithms (Ahuja et al. 2002; Burke et al. 2010; Kiefer et al. 2017; Pisinger and Ropke 2010) for timetabling, where a subset of the problem variables is fixed to reduce the problem size and the MIP solver is guided to iteratively find better solutions.

2 Problem definition

We begin by describing the general characteristics of the thirty problem instances of this competition and the requirements for feasibility. The complete problem definition can also be found in the description of the competition (Müller et al. 2018). Each timetable problem consists of rooms, classes with their course structure, distribution constraints, and students with their course demands. The aim is to place classes in the available times and rooms as well as to section students into classes based on the courses they require, respecting various constraints and preferences.

A *course* consists of a hierarchical structure of *classes* C, i.e., events to be scheduled. Each class $c \in C$ must be assigned a unique *time assignment*, chosen from a list of possible time assignments T_c. Some classes may also require a room to be assigned, chosen from a list of *room assignments* R_c. A class can meet several times a week during certain weeks of the semester. In that case, all meetings of the class start and end at the same time and are placed in the same room. Each time assignment consists of a start time, duration, day(s) of week and week(s) of the semester that the class meets, for example, 'between 9 and 10 am on Mondays and Wednesdays of weeks 1, 3, 5, 7, 9.' The solution must specify for each class one time and, if applicable, one room assignment. It is not permitted for two classes to share the same room at the same time, nor for a room to be used when it is not available—a list of unavailable slots is provided where applicable for each room.

Moreover, a set of hard *distribution constraints* is provided, each one imposing restrictions to the time and/or room choices that can be assigned to a subset of the classes. For example, a distribution constraint may require some classes to start at the same time (SameStart), take place on the same day (SameDay), use the same room (SameRoom), not take place at the same time (NotOverlap), not take place at the same time and allow for sufficient travel time (SameAttendees), impose an order on the starting time of the classes (Precedence), require a minimum amount of time separation between them (MinGap) or require the use of a maximum number of daily time slots (MaxDayLoad). Overall, there are 19 types of distribution constraints defined in Müller et al. (2018), although not all types apply to every problem instance.

A set of soft distribution constraints is also defined, each associated with a penalty, which are identical to the hard distribution constraints above but are allowed to be violated in the solution.

Each student $s \in S$ is assigned a list of courses $O_s \subset O$ which he must attend. Courses are linked to classes via a complex hierarchical structure. Each course $o \in O$ contains one or many course configurations $f \in F_o$. In turn, each course configuration defines a number of subparts $P_f \subset C$. Under this hierarchy, each student must attend one class of every subpart of a single configuration for each course from his or her list of courses (Müller et al. 2018).

This implies that if a particular class is assigned to a student, then this may forbid or may impose some choices of other classes in the course for this student. A class may also have a *parent* class defined which means that any student who attends the class must also attend its parent class.

It is possible that a student is allocated classes that result in a *clash* because he is unable to attend all of them: a clash is defined as the assignment of two classes to a student which overlap in time or the travel time between them is not sufficient; this situation is allowed but penalized.

The optimization goal is to find a solution which satisfies the hard constraints and minimizes the total penalty associated with the choice of time and room assignments, violated soft distribution constraints and student clashes.

3 The mixed-integer programming formulation

The mathematical model consists of a formulation of the timetabling problem as a large linear mixed-integer program. The program is, in general, too large to be solved directly and an incremental solution method was developed to obtain a feasible solution and progressively improve the objective function. This approach resembles other two-stage matheuristic algorithms of literature such as (Lindahl et al. 2018) employing a combination of a MIP model and a local search heuristic.

3.1 Decision variables

The formulation uses four types of binary 0–1 decision variables: x, y, z and Z, which represent, respectively, the class time assignment, the class room assignment, the allocation of students to classes and the allocation of students to course configurations. These variables are defined in eqs. (1), (2), (3) and (4). A feasible solution to the timetabling problem can be unambiguously represented by these variables.

For every class $c \in C$ and time assignment $t \in T_c$:

$$x_{c,t} = \begin{cases} 1 & \text{if the class } c \text{ takes place at time } t, \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

For every class $c \in C$ and potential room assignment $r \in R_c$:

$$y_{c,r} = \begin{cases} 1 & \text{if the class } c \text{ takes place in room } r, \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

For every student $s \in S$ and class $c \in C$:

$$z_{s,c} = \begin{cases} 1 & \text{if the student } s \text{ is assigned to class } c, \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

For every student $s \in S$ and course configuration $f \in F_o$ belonging to a course $o \in O_s$ that he may attend:

$$Z_{s,f} = \begin{cases} 1 & \text{if student } s \text{ attends the configuration } f, \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

The rest of the section presents the formulation of the linear MIP constraints and objective function using the above variables. We first describe the fundamental constraints representing the basic requirements among classes, rooms and students, and later on the formulation of the hard distribution constrains and the components of the objective function.

3.2 Fundamental constraints

The fundamental constraints correspond to the main requirements of what defines an acceptable solution to the time tabling problem. We describe eight types of fundamental constraints whose interpretation is straightforward.

C1: *Every class c must have a time assignment t*. For every class $c \in C$:

$$\sum_{t \in T_c} x_{c,t} = 1 \quad (5)$$

C2: *Every class c must be assigned a room r, where applicable.* For every class $c \in C$ requiring a room:

$$\sum_{r \in R_c} y_{c,r} = 1 \quad (6)$$

C3: *Every student s must attend exactly one class c from each subpart P_f of the selected course configuration f for each course o that he must attend.* For each student $s \in S$, for each course $o \in O_s$, for each course configuration $f \in F_o$ and for each $P_f \subset C$:

$$\sum_{c \in P_f} z_{s,c} = Z_{s,f} \quad (7)$$

C4: *If a class c has a parent class c' defined, whenever the class c is assigned to a student s then the parent class c' must also be assigned to that student.* For every student $s \in S$ and class $c \in C$ with a parent class $c' \in C$:

$$z_{s,c} \leq z_{s,c'} \quad (8)$$

C5: *Every student must be assigned a course configuration $f \in F_o$, for every course o that he attends.* For every student $s \in S$ and course $o \in O_s$:

$$\sum_{f \in F_o} Z_{s,f} = 1 \quad (9)$$

C6: *The capacity M_c of each class in terms of the number of students must be satisfied.* For every class $c \in C$:

$$\sum_{s \in S} z_{s,c} \leq M_c \quad (10)$$

C7: *A room cannot be used when it is unavailable.* For every class $c \in C$ and for every $t \in T_c$ and $r \in R_c$, if the time assignment t overlaps with a period of unavailability of the assigned room r, then both these assignments cannot be made:

$$x_{c,t} + y_{c,r} \leq 1 \quad (11)$$

C8: *Two classes cannot happen at the same time in the same room.* For every pair of classes $c_1, c_2 \in C$ and every potential common room assignment $r \in R_{c_1} \cap R_{c_2}$, and every time assignment $t_1 \in T_{c_1}$ and $t_2 \in T_{c_2}$ where t_1 and t_2 overlap, not all four assignments can be made simultaneously:

$$x_{c_1,t_1} + x_{c_2,t_2} + y_{c_1,r} + y_{c_2,r} \leq 3. \quad (12)$$

3.3 Distribution constraints

In addition to the fundamental constraints, further linear constraints are added to the MIP to represent the hard distribution constraints. Since these constraints involve only the classes and not the students, they only contain the x and y variables.

The inclusion of the hard distribution constraints D_H in the model is done in a similar way to $C8$ as follows: for every pair of classes $c_1, c_2 \in C_d$ of a hard distribution constraint $d \in D_H$, we calculate if a particular combination of time and room assignments results in a violated constraint. If that is the case, a constraint $C9$ is added as shown below.

C9: *For every pair of classes of a hard distribution constraint, forbid combinations of time and room assignments that result in a violated constraint.* For every

$c_1, c_2 \in C_d$ belonging to a hard distribution constraint $d \in D_H$ and every offending combination of $t_1 \in T_{c_1}$, $t_2 \in T_{c_2}$, $r_1 \in R_{c_1}$ and $r_2 \in R_{c_2}$:

$$x_{c_1,t_1} + y_{c_1,r_1} + x_{c_2,t_2} + y_{c_2,r_2} \leq 3. \quad (13)$$

For example, consider a SameDays constraint on the classes $\{c_1, c_2\}$ requiring the two classes to take place on the same day. For every pair of time assignments $t_1 \in T_{c_1}$ and $t_2 \in T_{c_2}$ representing different days, one constraint (13) is added, regardless of the choice of rooms (for any y_{c_1,r_1}, y_{c_2,r_2}).

The formulation $C9$ works for all distribution constraints that can be expressed as separate statements 'for each pair of classes' belonging to the distribution constraint. These constraints are the following 15 types of distribution constraint of Müller et al. (2018): SameStart, SameTime, DifferentTime, SameDays, DifferentDays, SameWeeks, DifferentWeeks, SameRoom, DifferentRoom, Overlap, NotOverlap, SameAttendees, Precedence, WorkDay and MinGap.

The remaining four types of distribution constraints, however, cannot be fully represented as a set of constraints among pairs of classes. These are the MaxDays, MaxDayLoad, MaxBreaks and MaxBlock types of constraints, which are referred to as 'special distribution constraints' in the description of the instances on the competition website[1].

For these special distribution constraints, it is necessary that the corresponding inequalities $C9$ are satisfied for every pair of classes in the constraint, but this is not sufficient. A simple counterexample is to consider a MaxDays constraint requiring three classes to take place over a total of 2 days or less: in that case, the constraint will always be satisfied for every pair of classes but, taken as a group of three, the classes will violate the constraint if they take place on three different days.

In our implementation, we begin by generating the necessary constraints $C9$ for each pair of classes in the special distribution constraint as usual and implement a check by means of a software lazy constraint callback to ensure feasibility. Each time a feasible solution is found during the solution of the MIP, we perform a check if the values of the solution violate any of the special distribution constraints. In the case that a special constraint is violated, we reject the current solution by adding the following constraint:

C10: *Every time an integer solution is found, check that all hard special distribution constraints present are satisfied, if any are not, then add the following inequality and continue the optimization.*

$$(x_{c_1,t_1} + y_{c_1,r_1}) + \cdots + (x_{c_N,t_N} + y_{c_N,r_N}) \leq 2N - 1 \quad (14)$$

where N is the number of classes contained in the specification of the special constraint which was found to be violated. This inequality forbids the current combination of x and y variables of the N classes. The constraints $C10$ are too numerous to include in the formulation from the start of the optimization, and with this approach, they are only added as needed.

3.4 Objective function

The objective function of the problem consists of four linear terms with instance-specific weights. The first two terms are simply the weighted sum of the time assignment x and room assignment y variables. The last two terms relate to violated soft distribution constraints and student clashes. In the rest of this section, we describe the inclusion of these two elements into the model, and in particular the student clashes which are the most complex part of the formulation. Despite the added complexity, the inclusion of all four terms in the objective function is crucial in order to obtain a solution of good quality.

The third term of the objective relates to penalties due to violated soft distribution constraints D_S. These penalties must be added to the objective for each pair of classes for which the soft distribution constraint $d \in D_S$ is violated. This is very convenient as we can use the same approach as for the hard distribution constraints (13) by adding an auxiliary indicator variable AUX denoting whether the constraint is violated (AUX = 1) or not (AUX = 0). The variable AUX is then added directly into the objective function with the appropriate weighting. This variable is created for every soft distribution constraint $d \in D_S$ and for every pair of classes $c_1, c_2 \in C_d$ in the description of that constraint (AUX_{d,c_1,c_2}). Several inequalities (15) may refer to the same AUX variable, as there are many ways that a given soft constraint can be violated by two classes, but we only account for this once in the objective. Similarly to the hard constraints, we generate $C11$ only for combinations of x and y that result in a violation.

C11: For every pair of classes $c_1, c_2 \in C_d$ of a soft distribution constraint $d \in D_S$, create a new 0-1 variable AUX_{d,c_1,c_2}. Then for every combination of $t_1 \in T_{c_1}$, $t_2 \in T_{c_2}$, $r_1 \in R_{c_1}$ and $r_2 \in R_{c_2}$ which results in a violation of the constraint, add the inequality:

$$x_{c_1,t_1} + y_{c_1,r_1} + x_{c_2,t_2} + y_{c_2,r_2} - \text{AUX}_{d,c_1,c_2} \leq 3. \quad (15)$$

Inequalities (15) force the variable AUX to take the value 1 whenever the soft constraint is violated. There is no need to require AUX = 0 when this does not happen, as the objective function (a minimization problem) will ensure this. For the same reason, it is not necessary to specify the AUX variables as integer; instead, they are added to the model as continuous 0–1 variables.

[1] International timetabling competition 2019. https://www.itc2019.org

Just as in the case of the hard constraints, consideration should be given to the soft special constraints (MaxDays, MaxDayLoad, MaxBreaks, MaxBlock) which are not fully represented by $C11$. They can be represented using inequalities similar to (15) with a larger number of terms:

$$(x_{c_1,t_1} + y_{c_1,r_1}) + \cdots + (x_{c_N,t_N} + y_{c_N,r_N}) - \text{AUX}_{d,c_1,\ldots,c_N} \leq 2N - 1 \quad (16)$$

where N is the number of classes in the constraint d and $\text{AUX}_{d,c_1,\ldots,c_N}$ is a continuous 0–1 variable, but their number is very large.

Of course, one could completely ignore these terms from the objective, but this would lead to poor quality solutions for the instances with a large number of such constraints. Instead, as our solution method consists of a sequence of optimization runs, we adopted a simple approach where we maintain a log of any violated soft special distribution constraints detected during the branch-and-bound, which are then added to the model during the next run. The optimization begins without any of these soft constraints included in the objective function. During each optimization run, whenever a feasible solution is produced we check for violated soft special distribution constraints. If any are found, they are added into a file, which is read at the beginning of the next optimization run, when the associated constraints (16) are added to the model and the AUX_{d,c_1,c_2} variables included in the objective function as described later. In the long term, this external file will contain a subset of soft special distribution constraints that merit to be added to the objective, having been found to be violated in the past runs. In practice, we observed that the size of this file tends to stabilize over time, suggesting that eventually only a few of these special soft constrains are the most useful to include in the objective function.

The fourth and last term of the objective relates to the penalties associated with student clashes. We distinguish between two types of student clash:

- A student cannot attend two classes because they overlap;
- A student cannot attend two classes because there is not sufficient time to travel between them.

The first type is simpler because we do not need to consider the room assignments: if two classes overlap, a clash exists regardless of the room assignment. For the second type, we need to consider the rooms allocated to the classes, as it is possible that a clash exists for some room assignments but not for others.

For the first type, for each student s and for each pair of classes c_1, c_2 he may attend and for each combination of their time assignments x_{c_1,t_1} and x_{c_2,t_2} which overlap, an auxiliary variable aux can be added denoting whether a clash is present for this student or not. The corresponding constraint is similar to (15):

$$x_{c_1,t_1} + x_{c_2,t_2} + z_{s,c_1} + z_{s,c_2} - aux_{c_1,t_1,c_2,t_2,s} \leq 3, \quad (17)$$

where $0 \leq aux \leq 1$ is a continuous variable. Let $T'_{c_2} \subseteq T_{c_2}$ be the set of values of $\{t_2\}$ for which the inequalities (17) are generated. Since the class c_2 can only have one time assignment (constraint C1) and therefore only one of the variables x_{c_2,t_2} can take the value 1, we can aggregate the inequalities (17) corresponding to different values of $t_2 \in T'_{c_2}$ into a single inequality (18):

$$x_{c_1,t_1} + \sum_{t_2 \in T'_{c_2}} x_{c_2,t_2} + z_{s,c_1} + z_{s,c_2} - aux_{c_1,t_1,c_2,s} \leq 3, \quad (18)$$

where $c_1, c_2 \in C$, $t_1 \in T_{c_1}$, $s \in S$. Equivalently, the aggregation can be done by the class c_1 instead of c_2 (in the implementation we used the variant that produced the smallest number of constraints—see also the discussion on constraint aggregation later on).

The number of constraints (18) is still too numerous to handle efficiently in the model because each student and each possible pair of classes are considered individually. We can further reduce the number of constraints significantly by grouping equations (18) among students who follow the same courses. To do this, we need to introduce the auxiliary variables w_{s,c_1,c_2} which denote whether a student s attends both classes c_1 and c_2 or not:

$$w_{s,c_1,c_2} = \begin{cases} 1, & \text{if } z_{s,c_1} = 1 \text{ and } z_{s,c_2} = 1, \\ 0, & \text{otherwise} \end{cases} \quad (19)$$

with the necessary constraints $0 \leq w \leq 1$ and $w_{s,c_1,c_2} \geq z_{s,c_1} + z_{s,c_2} - 1$. Note that as a further simplification we only create the w variables when necessary, because in many cases they can be substituted by one of the corresponding z variables or even their value may be fixed. This will happen when a student is obliged to attend a particular class ($z_{s,c_1} = 1$) therefore $w_{s,c_1,c_2} = z_{s,c_2}$, or alternatively if he can never attend a class ($z_{s,c_1} = 0$) in which case $w_{s,c_1,c_2} = 0$. Using the variables w, the inequalities (18) can be grouped for all N students who may attend the two classes c_1 and c_2 to produce:

$$Nx_{c_1,t_1} + N \sum_{t_2 \in T'_{c_2}} x_{c_2,t_2} + \\ +(w_{s_1,c_1,c_2} + \cdots + w_{s_N,c_1,c_2}) - \mathrm{AUX}_{c_1,t_1,c_2} \leq 2N \quad (20)$$

where $\mathrm{AUX}_{c_1,t_1,c_2}$ is a new continuous variable between 0 and N, equal to the sum of the original variables $aux_{c_1,t_1,c_2,s}$, denoting the number of students who cannot attend both classes c_1 and c_2 simultaneously.

For the second type of student clash, where two classes do not overlap but the time between them is not sufficient to allow the travel between the assigned rooms, we begin by considering the following special case. For each pair of classes, consider all the possible rooms that can be assigned to them, and calculate the minimum travel distance $M \geq 0$ among any possible rooms of the two classes. The values of M are specific to each problem instance and can be precalculated for efficiency.

If the time assignment x_{c_1,t_1} and x_{c_2,t_2} of two classes do not overlap but still have a gap between them of less than M periods, we can still use the inequalities (20) of the previous case, because a clash will exist regardless of the room allocation. In other words, if two classes do not overlap, but their gap in time is smaller than the time that is necessary under the best room assignment, then we do not need to consider the room assignments and this case is dealt in the same way as the previous one.

For the remaining cases of the second type of student clash, we must include the terms relating to the room allocation y:

$$x_{c_1,t_1} + \sum_{t_2 \in T'_{c_2}} x_{c_2,t_2} + z_{s,c_1} + z_{s,c_2} + y_{c_1,r_1} + y_{c_2,r_2} - \\ -aux_{c_1,t_1,r_1,c_2,r_2,s} \leq 5. \quad (21)$$

As before, we can group together all N students $s_i \in S$ who may attend classes c_1 and c_2 (20) to obtain:

$$Nx_{c_1,t_1} + N \sum_{t_2 \in T'_{c_2}} x_{c_2,t_2} + Ny_{c_1,r_1} + Ny_{c_2,r_2} + \\ +(w_{s_1,c_1,c_2} + \cdots + w_{s_N,c_1,c_2}) - \mathrm{AUX}_{c_1,t_1,r_1,c_2,r_2} \leq 4N. \quad (22)$$

The inequality (22) is generated for any $c_1, c_2 \in C, t_1 \in T_{c_1}$, $r_1 \in R_{c_1}, r_2 \in R_{c_2}$. The variable AUX is a new continuous variable between 0 and N denoting the number of students who cannot attend both classes c_1 and c_2 because their time and room assignments results in insufficient travel time.

The inequalities (20) and (22) fully describe for the student clashes and the corresponding variables AUX can be directly added into the objective function with the appropriate weights. In the implementation, we always generated (20) and for small problems we were able to generate all of the constraints (22). For larger instances, we used a method similar to the one used for the soft constraints: at the end of each optimization run, we check for those student clashes that were omitted from the objective and record the corresponding constraints (22) in a log file, to be used during the next optimization run.

It is worth noting that constraints (22) are not student-specific and correctly account for any number of students having the same clash. This means that adding a constraint during the next optimization run will not result in simply exchanging the affected student with another.

The full objective function containing all four cost elements to minimize, namely time allocation, room allocation, distribution constraints, student clashes, is therefore as follows:

$$\begin{aligned} \mathrm{obj} \; = \; & W_{\mathrm{T}} \sum_{c \in C} \sum_{t \in T_c} p_{c,t} x_{c,t} + W_R \sum_{c \in C} \sum_{r \in R_c} p_{c,r} y_{c,t} + \\ & + W_{\mathrm{D}} \sum_{d \in D_S} p_d \left(\sum_{c_1,c_2 \in C_d} \mathrm{AUX}_{d,c_1,c_2} + \sum_{\mathrm{file}} \mathrm{AUX}_{d,c_1,\ldots,c_N} \right) \\ & + W_{\mathrm{S}} \left(\sum_{c_1,c_2 \in C} \sum_{t_1 \in T_{c_1}} \mathrm{AUX}_{c_1,t_1,c_2} + \sum_{\mathrm{file}} \mathrm{AUX}_{c_1,t_1,r_1,c_2,r_2} \right) \end{aligned} \quad (23)$$

where W_{T}, W_{R}, W_{D} and W_{S} are the weights corresponding to the time penalty, room penalty, distribution penalty and student clashes, the constants p represent the weights attributed to each possible choice of time, room and violated soft constraint and the last summation denotes a sum over all the variables $\mathrm{AUX}_{c_1,t_1,r_1,c_2,r_2}$ of the specific type of student clashes which were read from an external log file.

3.5 Variable and constraint simplification strategies

The presented model accurately represents the timetabling problem in the sense that any feasible solution of the MIP corresponds to an acceptable solution for the timetabling problem, and, conversely, no solution to the real timetabling problem is infeasible for the MIP.

Nonetheless, it is possible to reduce the size of the MIP formulation by implementing a number of strategies for variable elimination and reduction of the number of constraints. This approach is more efficient that relying to the commercial MIP solver to detect and eliminate them. The strategies used in our computational implementation are described in the next sections.

3.5.1 Elimination of variables whose value is fixed

The first step is to consider decision variables whose values can be deduced; they are fixed to 0 or 1 accordingly and eliminated from the formulation. We use the following strategies:

- Variables x, y, z and Z where only one choice is possible; the corresponding variable is set to 1. These are in the constraints $C1$, $C2$, $C3$, $C5$ with only one term in the left-hand side.
- Variables that can be set to zero because another mutually exclusive variable is fixed to 1. This means that if $\sum x_i = 1$ and $x_k = 1$ is fixed to one, then all the other variables can be set to zero ($C1$, $C2$, $C3$, $C5$).
- If a class has both variables x and y fixed, and another class has its variable y fixed to the same room, we can then set to zero all variables x of the second class which overlap with the known time of the first class. In other words, if two classes must take place in the same room, and one class has its time fixed, we exclude all times from the other class that overlap with the first as no two classes can share the same room at the same time ($C8$).
- Some variables z do not need to be generated, as they can be replaced by the corresponding variable Z. This is the case where a course subpart contains only one possible class (constraint $C3$ with only one term in the left-hand side).
- The auxiliary variables w used to model student clashes need to be created only in certain situations as explained in (19). In the other cases, these variables will be either fixed or substituted by one of the corresponding z variables (which, in turn, may be replaced by a Z variable as described in the previous point).

3.5.2 Elimination of constraints that are always satisfied

Once the number of variables has been reduced, we proceed to identify sets of constraints are always satisfied and can be removed from the formulation. We concentrate on constraints of the form $\sum a_i x_i \leq c$ with $a_i > 0$. These constraints are always satisfied and therefore redundant if $\sum a_i \leq c$. We did not consider elimination of equality constraints because they are relatively few, although the commercial solvers used were able to detect and eliminate them as needed.

3.5.3 Reducing the number of constraints by aggregation

We can reduce the number of constraints in the formulation by taking advantage of the fact that exactly one time and/or room assignment is possible and group similar constraints together. This reduction was briefly described in the aggregation of constraints (17) to yield (18) in the objective function, but can also be applied to numerous constraints. For example, if we consider constraint $C7$ where combinations of x and y are excluded to forbid the use of a room when it is unavailable, we can aggregate a set of K constraints ($1 \leq i \leq K$) of the form:

$$x_{c_1,t_1} + y_{c_1,r_i} \leq 1$$

with a single equivalent constraint:

$$x_{c_1,t_1} + \sum_{1 \leq i \leq K} y_{c_1,r_i} \leq 1. \tag{24}$$

Alternatively, one can aggregate the same constraints over the x variables instead ($1 \leq i \leq L$), to produce:

$$\sum_{1 \leq i \leq L} x_{c_1,t_i} + y_{c_1,r_1} \leq 1. \tag{25}$$

As these two choices are equivalent, we select the one that yields the smallest number of aggregated constraints. In the above example, if $K \geq L$ we select the aggregation (24) to 'exclude several rooms for a given time,' otherwise we opt for (25) to 'exclude several times for a given room.' Aggregating these constraints has no impact on the feasibility of the problem.

4 Computational implementation

The mixed-integer programming model to solve the university timetabling problem was programmed in Java using the commercial software CPLEX[2] and Gurobi[3] as solvers.

The algorithm consists of two stages: the first stage is to obtain a feasible solution (a valid timetable with all the hard constraints satisfied), and the second stage is to iteratively improve the solution quality using a local search method while maintaining feasibility. As such, the solution improvement procedure can be repeated as long as desired although eventually better solutions become more and more rare.

4.1 Obtaining an initial solution

The first stage considers the production of a feasible solution with all the hard constraints satisfied. We begin with a small set of constraints, typically the equalities $C1$, $C2$, $C3$, $C5$, and use the MIP solver to optimize the model. The objective function is not relevant for this stage and can be omitted. Once a solution has been obtained, we iteratively 'inject' the remaining constraints into the model formulation, using the following strategy:

[2] IBM CPLEX. https://www.cplex.com

[3] Gurobi Optimization. https://www.gurobi.com

– We select one or a group of constraints to be added to the formulation.
– The solution obtained in the previous run is provided to the solver as a suggested starting, feasible solution to assist the optimization.
– If the solution obtained in the previous run satisfies, by coincidence, any of the new constraint inequalities, we simply add them to the MIP formulation.
– The remaining constraints, who are violated by the solution available from the previous run, are added to the formulation with an artificial slack variable, for example $a^T x - \text{slack} \leq c$.
– An objective function is added which aims to minimize the sum of the artificial slack variables.
– As the size of the MIP may be large, a local variable and constraint reduction strategy is used to reduce the number of variables and columns (local search heuristic).

At the start of the procedure, the slack variables and the objective function will be positive, whereas, at the end of the optimization they will become negative indicating that all the additional constraints are now satisfied and the slack variables can be removed. This will happen of course as long as the original problem was feasible.

In practice, it is not possible to directly minimize the objective in one step because of the size of the MIP, so the local search heuristic aims to reduce the number of variables in the formulation: in the simplest case, we randomly fix a percentage of the x, y, z, Z variables to the values they have in the solution of the previous run (the exact fixing strategy varies depending on the constraint and will be discussed later). The fixed variables are eliminated from the model, resulting in a smaller MIP that can be solved more quickly. The drawback is that as some variables are fixed there may still be positive slack variables at the end of the run and so the fixing procedure needs to be repeated.

For example, assume that a feasible solution has been obtained consisting of only the constraints $C1$, $C2$, $C3$, $C5$ and we wish to add constraints $C4$ to the model. We add the constraints $C4$ to the formulation using slack variables as necessary. We then consider fixing some variables to their current values: in this example, we can fix all variables x and y as they are irrelevant for the constraints $C4$ which involve the student allocations z.

Depending on the number of students in the instance, we can then proceed to fix a percentage of the z variables, typically 50–80% worked well in practice, so some assignments of students to classes will be fixed. We then optimize the MIP and any improvement to the objective function translates to a reduction of the number of constraints $C4$ that remain violated. By iterating the procedure (which can be automated and run unattended), we eventually obtain a solution where all constraints $C4$ are satisfied.

The strategy for fixing variables is important and automated fine tuning is necessary: the aim is to fix sufficient variables to allow the MIP to be solved in a reasonable time (we imposed a time limit of 15 min for all except the largest instances) but, on the other hand, fixing too many variables results in a slow convergence. The strategies used to reduce the number of variables were the following:

– Fixing a percentage of the variables (x or y or z or Z) at random.
– Fixing the x and/or y variables of a percentage of classes.
– Fixing the z and/or Z variables of a percentage of students.
– Fixing all variables except the x and/or y variables in the definition of a distribution constraint.

When a number of iterations was reached and there were still violated constraints, we would switch to the next strategy. Besides these strategies, we also fix any variables that are not relevant for the constraint: for example, when adding constraints relating to the student allocations we can fix the x, y variables, or conversely when adding the distribution constraints we can fix all z and Z variables.

4.2 Solution improvement and optimization

Once a feasible solution has been obtained and all the hard constraints are satisfied, an improvement heuristic is applied iteratively to increase the solution quality by reducing the solution penalty. The penalty consists of four terms which are approximated by the objective function (23).

The improvement heuristic is quite simple: we carry out a local search procedure by solving the MIP with a number of its decision variables fixed to the solution values obtained in the previous iteration. At each iteration, the feasibility is maintained and the optimization will maintain or improve the objective function. This procedure can be stopped at any time and the best obtained solution can be retrieved.

Regarding the strategies used to fix variables, we use the same methods as those described in the previous section for obtaining an initial solution, plus the additional two strategies below:

– Fix all variables except variables (x, z) or (y, z) relating to a percentage of students.
– Only include a percentage of students in the objective function term relating to student clashes.

The improvement heuristic is left to run unattended, switching to a different local search variable fixing strategy after a number of iterations is reached and no improvement in the objective was observed in recent runs.

Table 1 Characteristics of the competition instances

Instance	Classes	Rooms	Students	Distr. constr.		Optim.
				Hard	Soft	emphasis
1. agh-fis-spr17	1239	80	1641	820	400	Soft distr.
2. agh-ggis-spr17	1852	44	2116	2202	488	Soft distr.
3. bet-fal17	983	62	3018	861	390	Time
4. iku-fal17	2641	214	0	2237	665	Soft distr.
5. mary-spr17	882	90	3666	3151	796	Students
6. muni-fi-spr16	575	35	1543	645	95	Soft distr.
7. muni-fsps-spr17	561	44	865	331	69	Students
8. muni-pdf-spr16c	2526	70	2938	1456	570	Students
9. pu-llr-spr17	1001	75	27,018	416	218	Soft distr.
10. tg-fal17	711	15	0	459	42	Soft distr.
11. agh-ggos-spr17	1144	84	2254	1181	507	Soft distr.
12. agh-h-spr17	460	39	1988	288	111	Soft distr.
13. lums-spr18	487	73	0	449	69	Soft distr.
14. muni-fi-spr17	516	35	1469	639	60	Soft distr.
15. muni-fsps-spr17c	650	29	395	562	147	Students
16. muni-pdf-spr16	1515	83	3443	579	433	Students
17. nbi-spr18	782	67	2293	585	11	Soft distr.
18. pu-d5-spr17	1061	84	13,497	1262	273	Soft distr.
19. pu-proj-fal19	8813	768	38,437	6399	1398	Soft distr.
20. yach-fal17	417	28	821	529	116	Soft distr.
21. agh-fal17	5081	327	6925	4836	2318	Soft distr.
22. bet-spr18	1083	63	2921	1004	414	Time
23. iku-spr18	2782	208	0	2833	655	Soft distr.
24. lums-fal17	502	73	0	521	76	Soft distr.
25. mary-fal18	951	93	5051	349	164	Students
26. muni-fi-fal17	535	36	1685	635	152	Soft distr.
27. muni-fspsx-fal17	1623	33	1152	1070	289	Students
28. muni-pdfx-fal17	3717	86	5651	2433	1068	Students
29. pu-d9-fal19	2798	224	35,213	2039	707	Students
30. tg-spr18	676	18	0	375	50	soft distr.

We conclude this section by presenting some technical enhancements that we discovered during the computational experimentation and proved to be useful during our participation in the competition.

- *Data structure reuse and sharing* For example, even though the number of classes × possible time assignments is extremely large, a large number of time assignments is the same among classes. In the code, we made sure that each time pattern is only defined once and shared among the classes to avoid duplication.
- *Data pre-calculation* Some necessary information for the MIP is instance-specific and does not change between runs. We pre-calculated the maximum amount of such information and stored it in an external file which is used during each run. The pre-calculated information included the class-to-class minimum travel time matrix (distances), and a set of matrices indicating whether two time assignments overlap or not, whether they take place on the same day or their minimum gap.
- *Use of lazy constraints* In some cases, it is efficient to add some constrains into the MIP as 'lazy constraints,' a concept supported by the commercial solvers, where the constraint is not added in the model from the start but instead only each time a feasible solution is found, as necessary. This is necessary for the 'heavy' constraints $C8$ and $C9$, which can otherwise expand to hundreds of millions of rows for the larger instances.
- *Switching solvers* in our computational experimentation we produced code that can be solved with either the CPLEX or the Gurobi commercial solvers. In our experience, both solvers achieved similar performance in

their ability to optimize MIPs. We generally solved all instances with one solver until no further improvement was obtained in the solution value, determined by a limit on the number of runs or running time. We then noticed that if, at this stage, we switched to the other solver we sometimes obtained further (albeit small) improvements in the solution value. We suspect this behavior is due to the different default parameters or default strategies used by the two solvers, helping to overcome problems associated with one solver being stuck in a local optimum. Due to the competition time deadlines, we did not investigate the impact of changing the default strategies of each solver, but given that the ranking in the competition was determined by the solution value the small gains obtained by using both solvers were of course welcome.

4.3 Computational results

The computational results focused on solving the problem instances of the ITC competition. We were able to solve 29 of the 30 instances; the instance 'agh-fal17' was not solved in time for the competition deadline. The problem instances were solved on virtual machines running Linux equipped with 32GB RAM and 4 cores CPU Xeon E5 2.3 GHz, with the software CPLEX 12.8 and Gurobi 8.11 installed.

Table 1 presents the key characteristics of the problem instances, each instance representing a real university timetabling problem, and grouped in the order they appeared on the competition (early, middle and late instances). These are the number of classes, the number of rooms and the number of hard and soft distribution constraints.

We observe that the problem instances show a large variation in the number of classes, rooms, students and number of distribution constraints, with some instances having no students at all (the students are included in the problem via the SameAttendees constraints) while others have up to 38,000 students. We also note that the complexity is only partially related to the size of the instances. In some cases, in spite of the small number of classes, it was challenging to obtain even one feasible solution due to the high occupancy of rooms and time slots: it appeared that too few feasible configurations exist to begin with.

Table 1 also describes the emphasis of the optimization of each problem instance, represented by the largest relative weight among the four penalty elements. In some instances, the aim is to reduce the student clashes, in others to provide the best time assignments, whereas in others it is to satisfy the most soft distribution constraints.

Table 2 presents the results of our computational experimentation. The first four columns correspond to the first stage of the algorithm, the production of a feasible solution. We report the number of times the MIP was optimized (runs), the total clock time taken by all runs and the average number of rows and columns of the MIP per run (in thousands). For the second stage of improving the solution value, we further separate the time taken for setting up the MIP t_{setup} (data structures, reading/writing files, elimination of constraints, etc.) and the MIP optimization time t_{opt} taken by CPLEX or Gurobi. The twelfth column presents the ratio of integer variables divided by the total number of variables in the MIP. The last four columns relate to the values of the solution value (penalty): the first one of these is the solution value of the first feasible solution obtained (i.e., at the end of stage 1), followed by the solution value (A) corresponding to the best solution found (i.e., at the end of the algorithm). The last two columns present the value of the current best known solution (B) of each instance[4] at the conclusion of the competition and the calculated % gap of our solution relative to the best-known solution.

Due to the small number of instances and the disparity among their characteristics, it is difficult to generalize our conclusions; nonetheless, the following observations can be made:

- Our approach worked well for small and medium problems but poorly when the number of student was large. It is clear that the method for including the student clashes in the formulation becomes inefficient when dealing with tens of thousands of students, mainly due to equations (19) which are generated per student, and therefore some grouping of the students is necessary.
- In most cases the first stage to obtain a feasible solution concludes quite quickly: in half the instances this was done in under 1 hour and in two-thirds of the instances in under 2 hours. However, there were a few instances where we struggled to obtain a feasible solution, in one instance requiring over 10 days of calculations. These were the instances with the largest number of distribution constraints combined with a large number of classes. This is expected, as some distribution constraints require all the classes they involve to be able to change their time and room assignments and therefore would take longer for our class fixing/variable reduction method to arrive at fixing a good combination of classes.
- The number of columns in the MIPs solved in the second stage ranged from 5000 to 1 million and the corresponding rows from 30,000 to 1.6 million, after elimination of variables and constraints as described earlier. This resulted in short optimization times under typically under 15 min per run but allowed more runs in a given amount of time. This approach was deliberate because we noted that it is more beneficial to have a larger number of runs with smaller MIPs compared to solving fewer, larger MIPs. However, further investigation is necessary to calculate

[4] Obtained from https://www.itc2019.org

Table 2 Computational results for 29 instances

Instance	Stage 1				Stage 2										
	Time (hrs)	Runs #	Avg # k rows	Avg # k cols	Time (hrs)	t_{setup}(hrs)	t_{opt}(hrs)	Runs #	Avg # k rows	Avg # k cols	Int cols/cols	Sol val start	Sol val end (A)	Sol best known (B)	Gap % $_{(A/B)-1}$
1. agh-fis-spr17	1.7	20	67.6	27.4	11.2	5.3	5.9	479	66.2	67.3	0.3	11,480	4557	3039	50
2. agh-ggis-spr17	0.8	49	36.6	24.8	2.5	1.4	1.0	100	186.4	171.7	0.2	72,796	36,616	34285	7
3. bet-fal17	1.8	97	21.1	16.7	153.9	42.9	110.9	1895	199.7	210.7	0.2	372,785	295,427	289,965	2
4. iku-fal17	1.2	82	22.1	15.1	239.9	17.0	222.0	3635	37.0	21.4	0.9	52,313	26,840	18,968	42
5. mary-spr17	0.4	32	19.5	14.3	26.5	11.7	14.8	1842	39.5	38.8	0.3	22,395	15,021	14,910	1
6. muni-fi-spr16	1.8	25	268.9	304.6	150.4	24.8	125.6	4242	141.4	158.9	0.1	15,669	3844	3756	2
7. muni-fsps-spr17	1.0	18	62.8	63.6	47.1	1.6	45.6	790	61.5	60.7	0.2	13,589	883	868	2
8. muni-pdf-spr16c	4.4	328	45.3	45.6	100.2	63.1	37.1	2769	129.3	130.5	0.2	81,319	37,487	33,724	11
9. pu-llr-spr17	0.9	23	136.4	139.2	84.5	40.4	44.1	1421	146.6	138.8	0.3	34532	13,385	10,038	33
10. tg-fal17	0.1	5	84.7	22.6	0.7	0.2	0.5	32	98.4	18.4	0.5	4215	4215	4215	0
11. agh-ggos-spr17	8.4	656	52.0	18.3	33.9	27.2	6.7	3227	29.4	28.4	0.4	61,449	6320	2864	121
12. agh-h-spr17	0.0	557	0.0	0.0	50.3	28.7	21.6	810	631.5	95.7	0.2	43,207	26,159	22,175	18
13. lums-spr18	0.4	19	22.4	12.0	61.7	6.2	55.5	1170	939.8	4.8	0.8	167	114	95	20
14. muni-fi-spr17	0.8	36	102.2	122.5	94.7	35.8	58.9	1415	319.5	354.5	0.1	16,108	4,289	3825	12
15. muni-fsps-spr17c	0.8	61	93.3	21.5	131.7	72.8	58.9	2149	258.9	178.0	0.1	236,572	3303	2596	27
16. muni-pdf-spr16	1.4	50	28.7	31.0	64.7	22.7	42.0	696	350.0	193.6	0.4	244,453	24318	17,208	41
17. nbi-spr18	0.4	31	51.4	20.6	65.4	24.4	41.1	2044	296.3	88.4	0.1	65,490	19,055	18,014	6
18. pu-d5-spr17	1.1	30	60.6	70.5	17.0	8.5	8.6	332	133.6	156.4	0.6	54,230	18,813	15,204	24
19. pu-proj-fal19	262.2	1845	318.9	1542.2	86.0	64.1	21.9	159	288.6	213.2	0.9	926,972	561,194	117,425	378
20. yach-fal17	0.5	60	24.9	32.6	90.2	19.9	70.3	1562	158.2	142.8	0.1	29,715	1844	1074	72
22. bet-spr18	1.5	50	37.7	117.2	58.3	24.5	33.8	1033	75.9	138.1	0.6	480,397	360,057	348,524	3
23. iku-spr18	5.3	223	21.7	10.6	127.2	55.6	71.6	5086	171.6	23.1	0.9	138,065	36,711	25,868	42
24. lums-fal17	0.9	59	11.1	6.0	54.5	14.9	39.6	868	1620.6	6.8	0.8	1019	386	349	11
25. mary-fal18	0.7	28	3.1	82.2	70.4	32.8	37.6	1364	446.3	143.5	0.6	48,055	5637	4422	27
26. muni-fi-fal17	0.3	7	13.6	21.9	91.5	41.3	50.2	3643	291.5	329.5	0.2	16,675	3794	2999	27
27. muni-fspsx-fal17	12.7	844	85.9	58.0	45.4	37.0	8.4	2198	110.9	90.9	0.4	755,496	33,001	10,123	226
28. muni-pdfx-fal17	7.2	214	94.8	213.8	102.9	56.4	46.5	1353	83.2	195.5	0.8	906,140	151,464	98,373	54
29. pu-d9-fal19	21.6	429	105.7	147.5	4.6	3.9	0.7	47	237.7	1071.5	0.9	612,311	134,009	39,942	236
30. tg-spr18	0.2	10	172.8	30.2	7.2	3.2	4.0	309	178.9	48.2	0.2	15,948	12,856	12,704	1

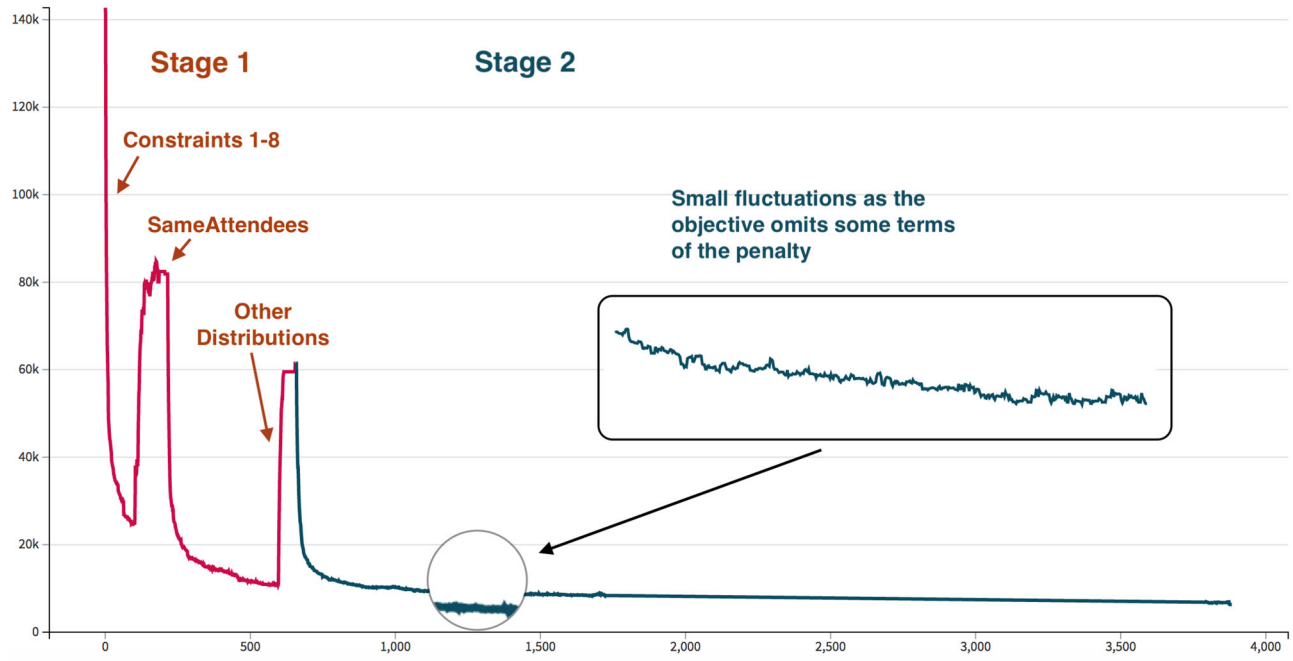

Fig. 1 Evolution of the solution quality over the course of the algorithm: total penalty versus number of iterations

the correct balance between the number of variables to fix at each run and the number of runs to use for each problem instance.

– The final gap was calculated by comparing our solutions with the best solutions published at the end of the competition. Our method performed worse for the larger instances, and it is evident that our approach does not scale well for very large instances and perhaps an aggregation or grouping step would be beneficial to reduce the dimensionality in these cases.

Figure 1 shows the evolution of the solution penalty for one instance (agh-ggos-spr17) against the number of runs. In the first stage, we observe two peaks corresponding to the increase of the penalty as we inject first the SameAttendees constraints and then the remaining distribution constraints. The objective function in this stage focuses heavily in reducing the number of violated constraints and with a smaller weight to improving the time and room assignments. The downward trend between the peaks corresponds to runs where not all violated constraints could be added, but we were still able to improve the time and room allocations.

During the second stage of optimization, the feasibility is maintained and the optimization aims to improve the objective function value. As mentioned earlier, the objective function of our model is a close approximation to the true penalty of the solution. The difference is due to the treatment of a subset of the student clashes (those related to when a student does not have enough travel time between two classes although the classes do not overlap) and those related to soft special distribution constraints: in both these situations, an external log file stores any missing objective function terms identified in previous runs to partially mitigate this, as explained earlier in the paper.

The instance of Fig. 1 contains both student clashes and MaxDays soft special constraints which cause the jagged appearance of the solution penalty; the actual objective function never increased; instead, it was missing some of the penalty terms.

Figure 2 presents the convergence patterns of a selection of four further instances. The first chart (bet-fal17) relates to a relatively small instance where the focus is to achieve good time assignments to each class. A feasible solution is obtained very quickly and most of the time is devoted to improve the objective function. The optimization initially aggressively optimizes the time and room assignments and distribution constraints while ignoring the student clashes. These are added at a later step and produce a temporary jump in the penalty, before reaching a good quality solution (2% gap). Note that it is meaningful to calculate the solution penalty only once the constraints $C1$–$C7$ are satisfied and each class is assigned a time and room and the first few runs are missing from the penalty charts of Fig. 2.

The second chart (muni-fsps-spr17c) considers a problem with many classes and distribution constraints where the optimization focusses on reducing the student clashes. The optimization is able to reduce the penalty quite rapidly but thereafter does not produce any big gains. The third instance (pu-proj-fal19) is the largest instance with over 8800 classes and 38,000 students, and the optimization emphasis is on the

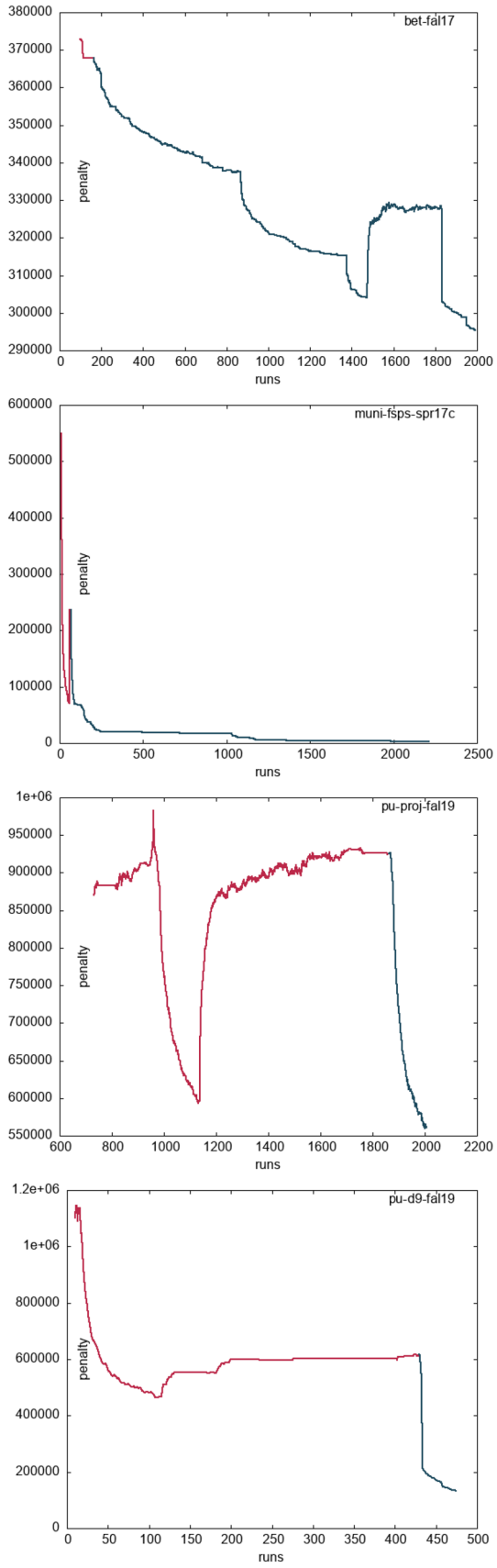

Fig. 2 Convergence of the solution penalty for a selection of instances

soft distribution constraints. For that instance, our algorithm had difficulty in obtain a feasible solution and most of the time was spent in Stage 1. The overall pattern is similar to the one of Fig. 1, and the upward trend between runs 1200 and 1800 suggests that the optimization is continuously able to reduce the infeasibility at each run, but the large number of constraints that need to be added results in a large computational time. The final reduction in penalty corresponds to Stage 2 where the solution is optimized, although the quality of the solution is very poor compared to the best known solution for this instance.

The last chart corresponds to the second largest instance where the focus was on reducing the student clashes. Just as in the third example, our algorithm took a long time to obtain a feasible solution but, in contrast to the previous case, we observe no continuous improvement during the last part of Stage 1 and therefore the challenge here was not due to the large number of violated constraints but instead that a few of them were very difficult to satisfy. A feasible solution is obtained eventually, and the small number of runs in the second stage resulted in a solution with a large gap compared to the best known solution.

In summary, we believe that the presented two-stage algorithm worked reasonably well for producing a feasible solution and iteratively improving on its quality for the small- and medium-sized instances of the competition. Further work is necessary to evaluate the methodology outside the restrictions of the competition, and the production of efficient variations of the algorithm which will depend on the optimization emphasis, problem size and other characteristics of each instance.

5 Conclusion and future work

This article presented an MIP model and solution methodology adapted to the instances of the International Timetabling Competition 2019. The proposed method consisted of a first stage which aimed at finding a feasible solution satisfying all the hard constraints and a second stage which used a local search approach to improve the objective function. The core model itself is not new; however, several adaptations and reductions, specific for timetabling problems, of the MIP formulation have produced an efficient approach for its solution and optimization, even in the presence of a large number of students and classes.

The future work will firstly attempt to combine the two stages together by formulating the model as a hierarchical multi-objective MIP which will improve performance. Moreover, a grouping algorithm will be beneficial in reducing the dimensionality for instances with a large number of students. Furthermore, an improved local search method is being developed to reduce the chance of getting stuck in local

optima, especially for problems with a large number of students, which was sometimes observed in the larger instances.

Finally, an adaptation of the methodology is in progress for the solution of more general timetabling and scheduling problems beyond the field of university timetabling.

Funding Open access funding provided by University of Applied Sciences and Arts Western Switzerland (HES-SO)

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Ahuja, R. K., Ergun, Ö., Orlin, J. B., & Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics, 123*(1), 75–102.

Bettinelli, A., Cacchiani, V., Roberti, R., & Toth, P. (2015). An overview of curriculum-based course timetabling. *TOP, 23*(2), 313–349.

Bonutti, A., De Cesco, F., Di Gaspero, L., & Schaerf, A. (2012). Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research, 194*(1), 59–70.

Burke, E., Eckersley, A., McCollum, B., Petrovic, S., & Qu, R. (2010). Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research, 206*(1), 46–53.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research, 19*(2), 151–162.

Dorneles, Á. P., de Araújo, O. C., & Buriol, L. S. (2014). A fix-and-optimize heuristic for the high school timetabling problem. *Computers and Operations Research, 52*, 29–38.

Fonseca, G. H., Santos, H. G., & Carrano, E. G. (2016). Integrating matheuristics and metaheuristics for timetabling. *Computers and Operations Research, 74*, 108–117.

Hwang, K.S., Lee, K.M., Jeon, J. (2004) A practical timetabling algorithm for college lecture-timetable scheduling. In: *Lecture notes in computer science*, pp. 817–825. Springer.

Kiefer, A., Hartl, R. F., & Schnell, A. (2017). Adaptive large neighborhood search for the curriculum-based course timetabling problem. *Annals of Operations Research, 252*, 255–282.

Kingston, J. H. (2013). Educational timetabling. In A. Uyar & E. U. N. Ozcan (Eds.), *Studies in computational intelligence* (pp. 91–108). Springer.

Lewis, R., Paechter, B., & Rossi-Doria, O. (2007). Metaheuristics for university course timetabling. In K. P. Dahal, K. C. Tan, & P. I. Cowling (Eds.), *Evolutionary scheduling* (pp. 237–272). Berlin Heidelberg: Springer.

Lindahl, M., Sørensen, M., & Stidsen, T. R. (2018). A fix-and-optimize matheuristic for university timetabling. *Journal of Heuristics, 24*(4), 645–665.

MirHassani, S. A., & Habibi, F. (2013). Solution approaches to the course timetabling problem. *Artificial Intelligence Review, 39*(2), 133–149.

Müller, T., & Rudová, H. (2016). Real-life curriculum-based timetabling with elective courses and course sections. *Annals of Operations Research, 239*(1), 153–170.

Müller, T., Rudová, H., Müllerová, Z. (2018) University course timetabling and international timetabling competition 2019. In: *PATAT 2018—Proceedings of the 12th international conference on the practice and theory of automated timetabling (PATAT 2018)*.

Pillay, N. (2014). A survey of school timetabling research. *Annals of Operations Research, 218*(1), 261–293.

Pisinger, D., & Ropke, S. (2010). Large neighborhood search. In M. Gendreau & J. Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 399–419). US, Boston, MA: Springer.

Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling, 12*(1), 55–89.

Rudová, H., Müller, T., & Murray, K. (2011). Complex university course timetabling. *Journal of Scheduling, 14*(2), 187–207.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review, 13*(2), 87–127.

Schmidt, G., & Ströhlein, T. (1980). Timetable construction-an annotated bibliography. *The Computer Journal, 23*(4), 307–316.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.