




Exact and metaheuristic methods for a real-world examination timetabling problem

Mats Carlsson¹ · Sara Ceschia² · Luca Di Gaspero² · Rasmus Ørnstrup Mikkelsen³  · Andrea Schaerf² · Thomas Jacob Riis Stidsen³

Accepted: 20 February 2023 / Published online: 4 May 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract

We propose a portfolio of exact and metaheuristic methods for the rich examination timetabling problem introduced by Battistutta et al. (in: Hebrard, Musliu (eds) 17th International conference on the integration of constraint programming, artificial intelligence, and operations research (CPAIOR-2020), LNCS, vol 12296. Springer, Berlin, pp 69–81, 2020). The problem includes several real-world features that arise in Italian universities, such as examinations split into two parts, possible requirements of multiple rooms for a single examination, and unavailabilities and preferences for periods and rooms. We developed a CP model encoded in the MiniZinc modeling language and solved it with Gecode, as well as two MIP models solved with Gurobi. The first MIP model is encoded natively and the second one again in MiniZinc. Finally, we extended the metaheuristic method based on simulated annealing of Battistutta et al. by introducing a new neighborhood relation. We compare the different techniques on the real-world instances provided by Battistutta et al., which have been slightly refined by correcting some semantic issues. Finally, we developed a solution checker that is publicly available, together with all instances and solutions, for inspection and future comparisons.

Keywords Examination timetabling · Constraint programming · Integer programming · Simulated annealing

1 Introduction

The timetabling of the final examinations is a difficult and crucial task that every university department has to solve regularly. Many versions of the examination timetabling problem (ETTP) have been proposed in the literature, as any

educational institution has its own rules and practices. They range from very simple ones, in which only examination conflicts are taken into consideration, so that they turn out to be simple extensions of the *graph coloring* problem (see, e.g., Carter et al., 1996), to extremely complex ones that include room constraints, preferences, heterogeneous timeslots, and many other practical features (see, e.g., McCollum et al., 2007).

In this paper, we consider the real-world version introduced by Battistutta et al. (2020) that applies to Italian universities, which includes composite examinations, preferences and unavailabilities for periods and rooms, examinations in multiple rooms, and many other features. In addition, conflicts are based on predefined curricula, rather than on student enrollments as more commonly done. The original formulation of Battistutta et al. (2020) has been refined to better capture a few real-world practices and to correct some semantic issues.

For this problem, we propose a portfolio of solution techniques comprising both exact and metaheuristic methods. Regarding the exact techniques, we developed two mixed integer programming (MIP) models and a constraint pro-

✉ Rasmus Ørnstrup Mikkelsen
rasmusomikkelsen@gmail.com

Mats Carlsson
mats.carlsson@ri.se

Sara Ceschia
sara.ceschia@uniud.it

Luca Di Gaspero
luca.digaspero@uniud.it

Andrea Schaerf
andrea.schaerf@uniud.it

Thomas Jacob Riis Stidsen
thst@dtu.dk

¹ RISE Research Institutes of Sweden, Uppsala, Sweden

² University of Udine, Udine, Italy

³ Technical University of Denmark, Lyngby, Denmark

gramming (CP) model, where the latter model is an enhanced version of the one originally proposed by Battistutta et al. (2020). On the metaheuristic side, we improved the simulated annealing (SA) proposed by Battistutta et al. (2020), by developing a new neighborhood relation.

Our search methods have been tested and compared among themselves and with the method by Battistutta et al. (2020), providing insights into the performances of the different methods.

All instances and best solutions are available at <https://bitbucket.org/satt/examtimetablinguniuddata> for inspection and future comparison, in both JSON and `dzn` (MiniZinc data files) formats. Furthermore, to encourage future contributions from other researchers, we have also developed and made publicly available a solution checker to protect against possible misinterpretations of the data and the constraints. Thanks to this tool, we have also detected and corrected some discrepancies between the JSON and `dzn` formats in the original files of Battistutta et al. (2020).

The paper is organized as follows. The problem formulations are provided in Sect. 2. Section 3 is dedicated to the discussion on the related work. Our search methods are introduced in Sect. 4. The experimental analysis is illustrated in Sect. 5. Finally, conclusions and future work are discussed in Sect. 6.

2 Problem formulation

Our problem consists of scheduling the final examination of a set of courses in a given time horizon. Within the horizon, we have to schedule one or more examination of the same course. Each examination represents a fully independent round (or call), so that students have multiple chances to pass a course. In turn, each examination can be composed of one or two parts (written and oral), called events.

Events must be assigned to a period, which represents a timeslot and a day of the time horizon. Events take place in rooms, which are classified according to their size. A single event might require more than one room. Events might also require no room at all so that they are conventionally assigned to the *dummy* room. In particular, in the oral part, students are questioned sequentially one at the time, so that it can be done also at the teacher's office.

Courses are grouped into curricula that determine conflicts and separations between their examinations. Each curriculum has a set of primary and secondary courses, which imply different levels of conflict and different distances among examinations.

In this formulation, we do not consider the enrollments of students to the specific examinations, as this information is not known at the time of the creation of the timetable.

Therefore, the only sources of conflicts are the curricula and the presence of the same teacher for two courses.

In detail, the main entities of the problem are the following:

Courses, examinations, & Events For each course, there are one or more examinations. Each examination might be a single event (either written or oral) or composed by two events: the written part (first) and the oral part (second), to be scheduled in this strict order.

Rooms Events require zero, one, or more rooms. Rooms are classified as small, medium, or large, and for each written event, we set the number and the size of rooms requested. (Mixed requests are not allowed.) The sets of rooms that can be assigned to a single event are called *composite rooms*, and they are listed in the input data. Composite rooms are homogeneous, such that all rooms of the set have the same size. An event may also use a room larger than it requested, but this does not apply to composite rooms. Oral events might require at most one single room (of any size).

Days, Timeslots, & Periods The time horizon is divided into days, and each day is split into timeslots. (The same number of timeslots are given in each day.) A period is a pair day/timeslot.

Curricula Curricula are sets of courses that have students in common, which therefore undergo the corresponding examinations. The set of courses of a curriculum is classified in *primary* courses and *secondary* ones, according to the expected number of students of the curriculum that has to take the corresponding examination. As explained below, the level of conflict between primary and secondary examinations of a curriculum is different.

The problem consists in assigning a period and a room (single, composite, or dummy) to each event, satisfying the hard constraints and minimizing the violations of the soft constraints.

The hard constraints are the following:

- H1. **RoomRequest:** Rooms assigned to an event must be of the correct size and quantity.
- H2. **RoomOccupation:** There must be at most one event per room per period.
- H3. **HardConflicts:** Two events must have different periods if they are in *hard conflict*. Two events have a hard conflict in the following cases: (i) they are part of courses that are both primary courses of one curriculum; and (ii) they have the same teacher.
- H4. **Precedences:** If one event *precedes* another, the first one must be scheduled before the second one. Two events have a precedence constraint in the following cases: (i) they are written and oral parts of the same

examination; and (ii) they are part of two consecutive examinations of the same course.

H5. Unavailabilities: Events cannot be assigned to an unavailable period or an unavailable room. Unavailabilities are explicitly stated for each specific event in the input data.

The objectives (soft constraints) are the following:

S1. *SoftConflicts*: Two event should have different periods if they are in *soft conflict*, i.e., if they belong to courses that are in the same curriculum, either as primary and secondary or both as secondary.

S2. *Preferences*: Events should not be assigned to an undesired period or an undesired room. A period can also be specified as preferred (a positive preference), so that in case of preferred periods for an event, all indifferent ones are assumed undesired (and explicitly undesired one are given a larger penalty).

S3. *Distances*: Requested period separations between events should be observed. We have two types of separations: *directed*, the first event must precede the second one, and *undirected*, such that they can be in any order. We have a directed distance constraint in the following cases:

- Same examination: written and oral events of the same examination have a minimum and a maximum distance.
- Same course: events belonging to consecutive examinations of the same course must be separated by a minimum distance. The separation constraint is applied between the first (or single) part of each of the two examinations.

We have a undirected distance constraint in the following case:

- Same curriculum: if two courses belong to the same curriculum, there should be a minimum separation between the examinations (as above, for two-part examinations, we consider the first one). The amount of separation and the weight for its violation depend on the type of the two (primary or secondary) memberships.

The weight of the violation of the various types of soft conflicts is normally set by the end-user. Similarly, all distance limits and the corresponding weights are configurable. For the sake of reproducibility, in this work we fix the weights to the values summarized in Table 1, whereas the distances are listed in the input data.

Table 1 Weights of soft constraints

ID	Name	Type	Weight
S1	<i>SoftConflicts</i>	Primary/secondary	5
		Secondary/secondary	1
S2	<i>Preferences</i>	Undesired period	10
		Not preferred period	2
		Undesired room	5
S3	<i>DirectedDistances</i>	Same examination	15
		Same course	12
S3	<i>UndirectedDistances</i>	Primary/primary	2
		Primary/secondary	2

3 Related work

Examination timetabling is a classical optimization problem that has been extensively studied in the scientific literature (see, e.g., Qu et al., 2009; Schaerf, 1999). Here, we discuss the formulations proposed, with particular attention to those equipped with some public datasets, which can be used to compare search methods.

The two formulations that have received most attention in the literature are the one by Carter et al. (1996) and the one proposed by McCollum et al. (2007) as part of the 2nd International Timetabling competition (ITC2007). These two formulations are rather different from each other, as the first is an extremely simple formulation (basically an extension of graph coloring), whereas the second one is more complex, including several peculiar rules of the British universities. Both formulations are complemented by very challenging datasets composed of 13 and 12 instances, respectively, none of which has been solved to proven optimality so far.

Another dataset comes from the Yeditepe University (Turkey) introduced by Özcan and Ersoy (2005) and subsequently modified by Bilgin et al. (2006), who also extended the Carter dataset by including room capacity (see Parkes and Özcan, 2010 for a discussion on Yeditepe and modified Carter datasets).

A study about real-world ETP in the Asian context was first introduced and subsequently extended by Kahar and Kendall (2010, 2015), who presented the case of Universiti Malaysia Pahang (Malaysia). Demeester et al. (2012) addressed the ETP at KAHO Sint-Lieven (Belgium), whereas Müller (2016) showed the application of examination timetabling to a large American university (Purdue University).

Woumans et al. (2016) proposed another interesting formulation, which addressed the problem from a student-centric perspective, considering the possibility of scheduling the same examination more than once if it improves the fairness among different student groups. The authors developed a Column Generation approach applied to a test case at

the KU Leuven campus (Belgium). Muklason et al. (2017) carried out an in-depth analysis of fairness in examination timetabling.

More recent activity includes Keskin et al. (2018), Abou Kasm et al. (2019), and Güler et al. (2021) that have presented different specific formulations for real-world ETPs.

Our formulation (Battistutta et al., 2020) consists of many peculiar constraints and objectives that have not previously been addressed together in the scientific literature. These include one or more examination of the same course, which, in turn, can combine a written and an oral part; examinations split across multiple rooms; and primary and secondary courses for each curriculum, determining different levels of conflicts between the respective examinations.

After discussing the variants of ETPs proposed in the literature, we move on to discussing the techniques employed for their solution. Many different search methods have been applied, ranging from exact methods, heuristics, metaheuristics, and hybrid techniques.

The literature on the application of integer programming (IP) for ETPs is relatively sparse. Some works focus on improved preprocessing and mixed integer programming (MIP) formulations of the examination timetabling problem posed in the ITC2007 (Arbaoui et al., 2015, 2019). Others investigate university-specific problems. Al-Yakoub et al. (2010) consider both examination timetabling and proctor assignment at Kuwait University, defining and solving a MIP for each problem. Cataldo et al. (2017) solve the examination timetabling problem at Diego Portales University (Chile) using IP in four stages. The interesting feature of the latter problem is that, similar to ours, it is *curriculum-based*, which means that conflicts are defined based on the predefined set of curricula. This is not the case for all other formulations, for which conflicts are expressed on the basis of the actual enrollment of the students to the specific examinations (called *post-enrollment* timetabling). Recently, Al-Hawari et al. (2020) have implemented a multistage MIP approach to solve the ETP at German Jordanian University (Jordan), which is validated both on real-world instances and on Carter's benchmarks.

June et al. (2019) studied the ETP at the Universiti Malaysia Sabah Labuan International Campus (UMSLIC) and developed a hybrid solution method that integrates CP and SA in two phases: An initial feasible schedule is obtained through constraint programming; then, the quality of the solution is improved by SA. The method was tested on datasets collecting the data of two semesters at UMSLIC (Malaysia). Genc and O'Sullivan (2020) provided a CP model, mainly based on bin-packing constraints, to solve the examination timetabling problem at University College Cork (Ireland).

Metaheuristics have proved to be really effective in solving ETPs. Current best-known results on Carter's instances

have been obtained by the SA developed by Bellio et al. (2021), the genetic algorithm of Leite et al. (2018), and the Great Deluge approach by Burke and Bykov (2016). Similarly, state-of-the-art solvers for the ITC2007 formulation have implemented variants of SA (Burke and Bykov, 2016; Battistutta et al., 2017; Leite et al., 2019).

4 Search methods

In this section, we describe the search methods developed in our study, namely mixed integer programming (Sects. 4.1 and 4.3), constraint programming (Sect. 4.2), and simulated annealing (Sect. 4.4).

Table 2 shows some common notation used in our MIP and CP models of the problem. In particular, we define the set \mathcal{R} of all rooms, which includes single, composite, and dummy rooms. On members of \mathcal{R} , we define the notion of *equivalence*: Two single rooms are equivalent if they have the same size (small, medium, or large); two composite rooms are equivalent if their members have the same size (composite rooms are homogeneous) and they have the same cardinality. Finally, for composite rooms, we define the notion of overlapping rooms, which are their (single) members and other composite rooms with members in common.

4.1 Mixed integer programming

In this section, we define an integer programming (IP) model for the problem. However, we can relax several of the integer variables, resulting in a mixed integer programming (MIP) model. We generally refer to the model as a MIP for simplicity, although it may be an IP. We use both model variants for computational testing and explicitly state when the model is an IP model. When comparing solution approaches, we designate this method **MIP**.

We introduce the **MIP** model in Sect. 4.1.1, and in Sect. 4.1.2, we describe a two-stage decomposition approach using the model which we call **MIP_{2S}**.

4.1.1 The MIP model

Table 3 shows the core decision variables of the model. We have $x_{e,p,r}$ as the main decision variable, determining each event's period and room assignment. We include auxiliary variable $y_{e,p}$ to simplify writing some constraints and reduce the number of nonzeros of the model. Additionally, we include h_e to attain the ordinal value of the period assigned to each event. We use "lazy" notation and do not include the most obvious conditions in sums. For example, since we only define $x_{e,p,r}$ for $p \in \mathcal{P}_e \wedge r \in \mathcal{R}_e$, a sum over events for

Table 2 Notation for modeling the problem

Sets	Description
\mathcal{E}	The set of events
\mathcal{P}	The set of periods
\mathcal{R}	The set of rooms, including dummy room \tilde{r}
\mathcal{R}^c	The set of composite rooms ($\mathcal{R}^c \subset \mathcal{R}$)
\mathcal{K}	The set of room equivalence classes
$\mathcal{R}_{r^c}^o$	The set of overlapping rooms of composite room $r^c \in \mathcal{R}^c$
\mathcal{P}_e	The set of available periods for event $e \in \mathcal{E}$
\mathcal{R}_e	The set of available rooms for event $e \in \mathcal{E}$
\mathcal{K}_e	The set of available room equivalence classes for event $e \in \mathcal{E}$
F	The set of examination pairs with precedence constraints (H4)
HC_e	The set of events that is in hard conflict with $e \in \mathcal{E}$ (H3)
DP^{\leftarrow}	The set of event pairs with a directed soft distance constraint
DP^{\leftrightarrow}	The set of event pairs with an undirected soft distance constraint
Helper	Description
$O(\cdot)$	Ordinal operator

Table 3 Decision variables of the MIP model

Variables	Description
$x_{e,p,r} \in \mathbb{B}$	1 if event $e \in \mathcal{E}$ is assigned to period $p \in \mathcal{P}_e$ and room $r \in \mathcal{R}_e$
$y_{e,p} \in \mathbb{B}$	1 if event $e \in \mathcal{E}$ is assigned to period $p \in \mathcal{P}_e$
$h_e \in \mathbb{Z}_0$	The ordinal value of the period assigned to event $e \in \mathcal{E}$

$x_{e,p,r}$ should be written as

$$\sum_{\substack{e \in \mathcal{E} \\ r \in \mathcal{R}_e \wedge p \in \mathcal{P}_e}} x_{e,p,r}$$

but we simply write

$$\sum_{e \in \mathcal{E}} x_{e,p,r}$$

Constraints (1)–(7) ensure that we satisfy all hard constraints. Constraint (1) assigns each event to an available period and room, satisfying H1 and H5. Constraints (2) and (3) enforce H2 by ensuring that at most one event can use a single room in any period and that composite rooms can only be used if none of its overlapping rooms are simultaneously used. In constraint (3), M equals the number of elements in the overlapping rooms sum. We enforce precedence relationships (H4) using (4) and avoid hard conflicts (H3) using (5), where M equals the number of elements in the sum. Constraints (6) and (7) set the values of $y_{e,p}$ and h_e , respectively.

$$\sum_{p \in \mathcal{P}_e} \sum_{r \in \mathcal{R}_e} x_{e,p,r} = 1 \quad \forall e \in \mathcal{E} \tag{1}$$

$$\sum_{e \in \mathcal{E}} x_{e,p,r} \leq 1 \quad \forall r \in \mathcal{R}, p \in \mathcal{P} \tag{2}$$

$$M \sum_{e \in \mathcal{E}} x_{e,p,r^c} + \sum_{r^o \in \mathcal{R}_{r^c}^o} \sum_{e \in \mathcal{E}} x_{e,p,r^o} \leq M \quad \forall r^c \in \mathcal{R}^c, p \in \mathcal{P} \tag{3}$$

$$h_{e_1} - h_{e_2} \leq -1 \quad \forall (e_1, e_2) \in F \tag{4}$$

$$M \cdot y_{e,p} + \sum_{e_2 \in HC_e} y_{e_2,p} \leq M \quad \forall e \in \mathcal{E}, p \in \mathcal{P}_e \tag{5}$$

$$y_{e,p} - \sum_{r \in \mathcal{R}_e} x_{e,p,r} = 0 \quad \forall e \in \mathcal{E}, p \in \mathcal{P}_e \tag{6}$$

$$\sum_{p \in \mathcal{P}_e} O(p) \cdot y_{e,p} = h_e \quad \forall e \in \mathcal{E} \tag{7}$$

We now move to the soft constraints, starting with soft conflicts (S1), which involve primary/secondary (PS) and secondary/secondary (SS) curriculum connections. For each event $e \in \mathcal{E}$, we define a set of PS and SS conflicting events as SC_e^{PS} and SC_e^{SS} , respectively. We also introduce integer variables $s_{e,p}^{PS}$ and $s_{e,p}^{SS}$ to, respectively, “count” the number of PS and SS soft conflicts for event e in period p . We set the variable values using constraints (8) and (9). For these constraints, we get the values of big- M as the number of elements in the sum, e.g., $M_{e,p}^{PS} = |\{e_2 \in SC_e^{PS} : O(e) < O(e_2) \wedge p \in \mathcal{P}_{e_2}\}|$. The $O(e) < O(e_2)$ condition ensures that a conflict is only counted once. Variables $s_{e,p}^{PS}$ and $s_{e,p}^{SS}$ are bounded by 0 and $M_{e,p}^{PS}$ and $M_{e,p}^{SS}$,

respectively.

$$M_{e,p}^{PS} \cdot y_{e,p} + \sum_{\substack{e_2 \in SC_e^{PS} \\ :O(e) < O(e_2)}} y_{e_2,p} \leq s_{e,p}^{PS} + M_{e,p}^{PS} \quad \forall e \in \mathcal{E}, p \in \mathcal{P}_e \tag{8}$$

$$M_{e,p}^{SS} \cdot y_{e,p} + \sum_{\substack{e_2 \in SC_e^{SS} \\ :O(e) < O(e_2)}} y_{e_2,p} \leq s_{e,p}^{SS} + M_{e,p}^{SS} \quad \forall e \in \mathcal{E}, p \in \mathcal{P}_e \tag{9}$$

To penalize violations of soft conflicts, we add the following terms to the objective function, where β^{PS} and β^{SS} denote the soft conflict costs for PS and SS event pairs, respectively.

$$Cost^{S1} = \beta^{PS} \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{P}_e} s_{e,p}^{PS} + \beta^{SS} \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{P}_e} s_{e,p}^{SS}$$

The second soft constraint (S2) is event period and event room preferences. We define nonnegative costs α_{ep} and α_{er} and add the following terms to the objective function.

$$Cost^{S2} = \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{P}_e} \alpha_{ep} y_{e,p} + \sum_{e \in \mathcal{E}} \sum_{p \in \mathcal{P}_e} \sum_{r \in \mathcal{R}_e} \alpha_{er} x_{e,p,r}$$

As the third soft constraint (S3), the problem includes preferred distances between event pairs. For these event pairs, we are given a minimum/maximum distance parameter P_{e_1,e_2}^{min} and P_{e_1,e_2}^{max} , respectively. First, we show how we “measure” the distance between event pairs before showing how we penalize violations. Let DP^{\leftarrow} and DP^{\leftrightarrow} be the sets of event pairs with a soft distance (min. or max.) constraint with directed and undirected distances, respectively. An event pair (e_1, e_2) has a directed distance requirement when we have a hard precedence constraint (H4) that guarantees that e_1 precedes e_2 .

In order to include these soft constraints, we introduce the variables shown in Table 4. First, we define the integer variable d_{e_1,e_2} to take the absolute value of the distance between e_1 and e_2 . Notice that the maximum distance possible between any two events is always bounded by $|\mathcal{P}|$. When event pairs have a directed distance requirement, we set d_{e_1,e_2} correctly using constraint (10).

$$d_{e_1,e_2} = h_{e_2} - h_{e_1} \quad \forall (e_1, e_2) \in DP^{\leftarrow} \tag{10}$$

When event pairs have an undirected distance requirement, we need to use some additional auxiliary variables and constraints to get the absolute distance correctly. We introduce integer variable $d_{e_1,e_2}^{\leftrightarrow}$ to measure the distance between the two events. We set this variable using constraint (11), and the variable may then take a negative value. We need $d_{e_1,e_2} = |d_{e_1,e_2}^{\leftrightarrow}|$, which we linearize in the following.

We introduce binary variable $g_{e_1,e_2}^{\leftrightarrow}$ to be 1 if $d_{e_1,e_2}^{\leftrightarrow}$ is positive and 0 otherwise. We set $g_{e_1,e_2}^{\leftrightarrow}$ using constraints (12) and (13). Then we introduce two variables d_{e_1,e_2}^{abs1} and d_{e_1,e_2}^{abs2} where exactly one takes the absolute value of $d_{e_1,e_2}^{\leftrightarrow}$ and the other takes the value of zero. We set the values of these two variables using (14)–(18). For simplicity in writing the model, we set the value of d_{e_1,e_2} using constraint (19). In practice, we (of course) simply use $d_{e_1,e_2}^{abs1} + d_{e_1,e_2}^{abs2}$ in place of d_{e_1,e_2} .

$$d_{e_1,e_2}^{\leftrightarrow} = h_{e_2} - h_{e_1} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{11}$$

$$d_{e_1,e_2}^{\leftrightarrow} \leq |\mathcal{P}| \cdot g_{e_1,e_2}^{\leftrightarrow} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{12}$$

$$d_{e_1,e_2}^{\leftrightarrow} \geq -|\mathcal{P}| (1 - g_{e_1,e_2}^{\leftrightarrow}) \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{13}$$

$$d_{e_1,e_2}^{abs1} \leq |\mathcal{P}| g_{e_1,e_2}^{\leftrightarrow} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{14}$$

$$d_{e_1,e_2}^{abs1} \geq -|\mathcal{P}| g_{e_1,e_2}^{\leftrightarrow} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{15}$$

$$d_{e_1,e_2}^{abs1} \leq d_{e_1,e_2}^{\leftrightarrow} + |\mathcal{P}| (1 - g_{e_1,e_2}^{\leftrightarrow}) \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{16}$$

$$d_{e_1,e_2}^{abs1} \geq d_{e_1,e_2}^{\leftrightarrow} - |\mathcal{P}| (1 - g_{e_1,e_2}^{\leftrightarrow}) \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{17}$$

$$d_{e_1,e_2}^{abs2} = d_{e_1,e_2}^{abs1} - d_{e_1,e_2}^{\leftrightarrow} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{18}$$

$$d_{e_1,e_2} = d_{e_1,e_2}^{abs1} + d_{e_1,e_2}^{abs2} \quad \forall (e_1, e_2) \in DP^{\leftrightarrow} \tag{19}$$

Table 5 shows an overview of when min. and max. distance soft constraints exist between two events and if the distance is directed. Additionally, the table shows the variable used to capture each specific distance violation, the associated cost parameter, and the event pair sets for each category. WO is shorthand for written and oral. Additionally, PP and PS indicate primary–primary and primary–secondary relations, respectively. Thus, for example, the last line of the table relates to events that are associated two courses such that one is a primary and the other a secondary course of the same curriculum. To get the minimum and maximum distance violation, we, respectively, use constraints with the following structure: $p_{e_1,e_2} + d_{e_1,e_2} \geq P_{e_1,e_2}^{min}$, and $d_{e_1,e_2} - p_{e_1,e_2} \leq P_{e_1,e_2}^{max}$. Thus, constraints (20)–(24) correctly set the p_{e_1,e_2} variables in the order they are shown in Table 5.

$$p_{e_1,e_2}^{min E} + d_{e_1,e_2} \geq P_{e_1,e_2}^{min} \quad \forall (e_1, e_2) \in DP^E \tag{20}$$

$$p_{e_1,e_2}^{min WO} + d_{e_1,e_2} \geq P_{e_1,e_2}^{min} \quad \forall (e_1, e_2) \in DP^{WO} \tag{21}$$

$$d_{e_1,e_2} - p_{e_1,e_2}^{max WO} \leq P_{e_1,e_2}^{max} \quad \forall (e_1, e_2) \in DP^{WO} \tag{22}$$

$$p_{e_1,e_2}^{min PP} + d_{e_1,e_2} \geq P_{e_1,e_2}^{min} \quad \forall (e_1, e_2) \in DP^{PP} \tag{23}$$

$$p_{e_1,e_2}^{min PS} + d_{e_1,e_2} \geq P_{e_1,e_2}^{min} \quad \forall (e_1, e_2) \in DP^{PS} \tag{24}$$

We then extend the objective function with

$$Cost^{S3} = \gamma^E \sum_{(e_1,e_2) \in DP^E} P_{e_1,e_2}^{min E}$$

Table 4 Decision variables included for the S3 constraints

Variables	Description
$d_{e_1,e_2} \in \mathbb{Z}_+$	The absolute distance value between assignments of e_1 and e_2
$d_{e_1,e_2}^{\leftrightarrow} \in \mathbb{Z}$	The actual distance between assignments of e_1 and e_2
$g_{e_1,e_2}^{\leftrightarrow} \in \mathbb{B}$	1 if $d_{e_1,e_2}^{\leftrightarrow}$ is positive
$d_{e_1,e_2}^{abs1} \in \mathbb{Z}_+$	The absolute value of $d_{e_1,e_2}^{\leftrightarrow}$ or zero
$d_{e_1,e_2}^{abs2} \in \mathbb{Z}_+$	The absolute value of $d_{e_1,e_2}^{\leftrightarrow}$ or zero

Table 5 Overview of soft constraints related to minimum and maximum distance (S3)

Event pair connection	Type	Directed	Variable	Cost	Set
Examinations of same course	Min	Yes	P_{e_1,e_2}^{minE}	γ^E	DP^E
WO parts of same examination	Min	Yes	P_{e_1,e_2}^{minWO}	γ^{WO}	DP^{WO}
WO parts of same examination	Max	Yes	P_{e_1,e_2}^{maxWO}	γ^{WO}	DP^{WO}
Courses in PP curriculum	Min	No	P_{e_1,e_2}^{minPP}	γ^{PP}	DP^{PP}
Courses in PS curriculum	Min	No	P_{e_1,e_2}^{minPS}	γ^{PS}	DP^{PS}

$$\begin{aligned}
 & + \gamma^{WO} \sum_{(e_1,e_2) \in DP^{WO}} (P_{e_1,e_2}^{minWO} + P_{e_1,e_2}^{maxWO}) \\
 & + \gamma^{PP} \sum_{(e_1,e_2) \in DP^{PP}} P_{e_1,e_2}^{minPP} \\
 & + \gamma^{PS} \sum_{(e_1,e_2) \in DP^{PS}} P_{e_1,e_2}^{minPS}
 \end{aligned}$$

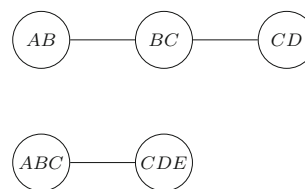


Fig. 1 Composite room conflict graph for the given example

Finally, we note that we can relax the soft constraint penalty counting variables ($s_{e,p}^{PS}$, $s_{e,p}^{SS}$, P_{e_1,e_2}^{minE} , P_{e_1,e_2}^{minWO} , P_{e_1,e_2}^{maxWO} , P_{e_1,e_2}^{minPP} , and P_{e_1,e_2}^{minPS}) to be continuous, as they naturally will attain integer values using the given constraints. Thus, we can express the model either as an IP or a MIP.

4.1.2 Two-stage decomposition

A typical approach for timetabling problems, especially for IP-based methods, is constructing the timetable in two stages (see, e.g., Daskalaki and Birbas, 2005; Kristiansen et al., 2015; Al-Yakoob and Sherali, 2015). One successful decomposition strategy involves first assigning events to periods and afterward to rooms (Lach & Lübbecke, 2008, 2012; Sørensen & Dahms, 2014). For this decomposition scheme to be effective, we must ensure that the solution found in the first stage is feasible or that we can easily fix any infeasibility in the second stage. In this problem, the only infeasibility that could arise between the two stages is not observing the room double-booking constraints (H2). However, rooms are generally in excess for the available data, especially since events can use rooms larger than they requested. Additionally, we have no event room forbidding (hard) constraints in the current data, and thus, with regard to feasibility, events are indifferent to specific rooms, and rooms can be treated as generic rooms of a specific size.

Here we discuss how we implement such a decomposition for this problem, which we designated MIP_{2S} . In the first stage, $y_{e,p}$ becomes the primary assignment variable, and we leave out all room-related aspects, e.g., the $x_{e,p,r}$ variables, room double-booking constraints, and room preference objectives. We add these parts to the model in the second stage. We use the first stage solution as a warm start and do not fix any assignments, allowing the MIP solver to change period assignments freely, and since the second stage model is the full model, we gain valid lower bounds. In the following, we define constraints to significantly reduce the risk of overbooking a period in the first stage regarding room capacity in the second stage.

Let \mathcal{T} be the set of single room sizes, i.e., $\mathcal{T} = \{S, M, L\}$ where S, M, L denote small, medium, and large, respectively. Composite rooms consist of multiple rooms of the same size, and we can therefore consider the combination of room size and number of rooms, which we call a “room-type.” Let \mathcal{N}_t be a set of “number of rooms” for a given room size t . For example, we have $\mathcal{N}_S = \{1, 2, 3\}$ if the instance data includes single small rooms and composite rooms consisting of two, and three small rooms. Then $(n, t) = (2, S)$ denotes the room-type consisting of two small rooms.

Let $\mathcal{E}_{n,t}^{RT}$ be the set of events that require a room of room-type (n, t) and $c_{n,t,p}$ the capacity of room-type (n, t) in period p . For single rooms ($n = 1$), we set the capacity

to the number of single rooms of size t , or larger, available in period p . For composite rooms, we need to handle composite room member conflicts. We consider an example with five single small rooms ($\{A, B, C, D, E\}$) and three (2, S) ($\{AB, BC, CD\}$) and two (3, S) ($\{ABC, CDE\}$) composite rooms. Figure 1 shows the composite room conflict graphs for the example. By inspection, we see that at most two (2, S) and one (3, S) can be used at a time. These limits correspond to each graph’s independence number. However, due to room unavailabilities, the graph’s nodes may depend on the period p . Thus, we get the capacity $(c_{n,t,p})$ for composite room-types as the independence number of the conflict graph in period p .

Examinations requesting a single room can use rooms of that size or larger, e.g., an examination requesting a small room can use small, medium, and large rooms. Examinations requesting a composite room can only use composite rooms that meet their exact specification. Constraint (25) ensures that we observe single room occupation and (26) specifically limits the number of events assigned in a single period that require composite rooms.

$$\sum_{i' \in \mathcal{T}} \sum_{n \in \mathcal{N}_i} n \sum_{e \in \mathcal{E}_{n,t}^{RT}} y_{e,p} \leq \sum_{i' \in \mathcal{T}} C_{i,1,p} \quad \forall t \in \mathcal{T}, p \in \mathcal{P} \quad (25)$$

$$\sum_{e \in \mathcal{E}_{n,t}^{RT}} y_{e,p} \leq C_{n,t,p} \quad \forall t \in \mathcal{T}, n \in \mathcal{N}_t : n > 1, p \in \mathcal{P} \quad (26)$$

These constraints do not consider composite room conflicts for composite rooms with a different number of member rooms. However, in most cases, they are sufficient as few instances have composite rooms with three and four member rooms, and fewer still with the possibility of conflicts between them. We have not experienced a solution from the first stage being infeasible in the second stage throughout our testing. By far, it is most important to forbid double-booking between single and composite rooms with two member rooms, as they are most common. Constraints (25) and (26) handle this fully.

4.2 Constraint programming

In this section, we define CP, a constraint programming (CP) model for the problem, encoded in the MiniZinc modeling language (see Nethercote et al., 2007) and solved with the Gecode backend (see Gecode Team, 2019).

Although CP is an exact method, we actually use a heuristic search method, because exhaustive search does not work well for this problem. The heuristic search uses restarts and large neighborhood search (LNS, see Shaw, 1998, Dekker et al., 2018)

on top of a branch-and-bound scheme. This is expressed by search annotations, which are passed to Gecode for execution. It is worth noting that this is supported by MiniZinc straight out of the box.

As noted earlier, events are indifferent to specific rooms, and rooms can be treated as generic rooms of a specific size. We thus split \mathcal{R} into a set \mathcal{K} of equivalence classes. Then, we solve the problem in terms of equivalence classes instead of specific rooms, relaxing constraint H2 to allow multiple events per class simultaneously up to the capacity of the class. Finally, we trivially transform the obtained solution back into one expressed in terms of specific rooms. We now present the MiniZinc model step by step.

The parameters Events, Periods, Rooms, and CRooms correspond to $|\mathcal{E}|$, $|\mathcal{P}|$, $|\mathcal{R}|$, and $|\mathcal{K}|$, respectively. The decision variables are as follows, where CRoomPeriodIndex(k, p) is a bijective function of the room class k and period p . Also, CRoomPeriodIndex(k, p) is monotonically increasing in the absolute distance between p and the middle period. In other words, the farther away p is from the middle period, the larger the function value is. EventCRP[e] is an auxiliary variable for the (period, room class) combination of event e :

```
array [1..Events] of var 1..Periods: EventPeriod;
array [1..Events] of var 1..CRooms: EventCRoom;
array [1..Events] of var int: EventCRP =
  [CRoomPeriodIndex(EventCRoom[e], EventPeriod[e]) ::domain | e in 1..Events];
```

The first constraints encode constraints H1 and H5:

```
constraint
  forall(e in 1..Events, p in 1..Periods
    where EventPeriodConstraints[e, p] = -1) (EventPeriod[e] != p);

constraint
  forall(e in 1..Events, r in 1..CRooms
    where EventCRoomConstraints[e, r] = -1) (EventCRoom[e] != r);
```

By eliminating up front infeasible values from the domain of EventCRP, the next constraint contributes to encoding constraint H5:

```
constraint
  forall(e in 1..Events, c in 1..CRooms, p in 1..Periods
    where CRoomPeriodConstraints[c,p] = -1)
    (EventCRP[e] != CRoomPeriodIndex(c,p));
```

The next constraint encodes constraint H2:

- The quantity count[k, p] counts the number of events for given equivalence class k and period p by means of a global_cardinality_closed constraint (see Stuckey et al., 2020), and its value can be at most the cardinality of given equivalence class k .
- A composite room cannot be used simultaneously with an overlapping room.


```

constraint
let (array[1..CRooms,1..Periods] of var 0..max(CRoomCap): count) in
global_cardinality_closed({ EventCRP[e]
| e in 1..Events where RoomedEvent[e]=1
},
1..CRooms*Periods,
arrayId(count)) /\
forall(p in 1..Periods, r in 1..CRooms)(count[r,p] <= CRoomCap[r]) /\
forall(p in 1..Periods,
r1 in 1..CRooms,
r2 in 1..CRooms where r1<r2 /\ CRoomsetOverlap[r1,r2]=1){
min(count[r1,p], count[r2,p]) = 0
};
    
```

The next constraint encodes the hard conflicts rule (constraint H3) using an alldifferent constraint (see Stuckey et al., 2020), where each $c \in \text{AllCliques}$ is a maximal clique of events that are in hard conflict with each other. The cliques are computed by the Python function `networkx.find_cliques`, which is based on the algorithm published by Bron and Kerbosch (1973) in a pre-processing step, which runs in negligible time compared to the solving time.

```

constraint
forall(c in AllCliques)(alldifferent(e in c)(EventPeriod[e]));
    
```

The next constraint enforces precedences (constraint H4):

```

constraint
forall(e1 in 1..Events - 1, e2 in e1 + 1..Events
where Precedence[e1, e2] = 1)
(EventPeriod[e1] < EventPeriod[e2]);
    
```

The next constraint defines the `ConflictDistanceCost` term of the objective function (constraints S1 and S3). We have transformed the raw input data, which was given as several large arrays and several cost terms for many pairs of events, into two arrays `CostKeys` and `CostVectors` that aggregate these data into at most one cost term per event pair. This transformation was done in the same spirit as tabling (see Dekker et al., 2017). Let i be a row of `CostKeys` containing a pair (e_1, e_2) of events and let δ be their temporal distance. Then `CostVectors[i, δ]` is the incurred cost for that pair:

```

constraint
ConflictDistanceCost =
sum(i in index_set_1of2(CostKeys)) (
let {int: e1 = CostKeys[i,1],
int: e2 = CostKeys[i,2],
var int: delta = EventPeriod[e2]-EventPeriod[e1],
} in CostVectors[i, delta]
);
    
```

The last constraints define the remaining terms of the objective function (constraint S2):

```

constraint
CRoomPreferenceCost =
max(0, sum(e in 1..Events)(EventCRoomConstraints[e, EventCRoom[e]]));
constraint
PeriodPreferenceCost =
max(0, sum(e in 1..Events)(EventPeriodConstraints[e, EventPeriod[e]]));
constraint
CRoomPeriodCost =
max(0, sum(e in 1..Events)(CRoomPeriodConstraints[EventCRoom[e],
EventPeriod[e]]));
    
```

Finally, the search and minimization statement is as follows, where `SortedEvents` is the sequence of events in some heuristic order, and the four `*Cost` expressions are the terms of the objective function:

```

array[1..Events] of 1..Events: SortedEvents;
array[int] of var int: CRPE = [EventCRP[e] | e in SortedEvents];
solve
:: restart_luby(100)
:: relax_and_reconstruct(CRPE, 97)
:: int_search(CRPE, dom_w_deg, indomain_split)
minimize ConflictDistanceCost + CRoomPreferenceCost +
PeriodPreferenceCost + CRoomPeriodCost;
    
```

The search annotations are:

- `restart_luby(100)`—the k th restart is given a node limit of $100 \cdot L_k$, where L is the Luby restart sequence (see Luby et al., 1993).
- `relax_and_reconstruct(CRPE, 97)`—at every restart after finding the first solution, an LNS step is performed where 3% randomly selected decision variables are set free and the remaining 97% keep their current values.
- `int_search(CRPE, dom_w_deg, indomain_split)`—this annotation determines the order in which variables and their values are explored. The next variable to explore is the `EventCRP` variable with the smallest current domain size divided by weighted degree, breaking ties by the `SortedEvents` order. For the designated variable, explore the lower half of its domain first, splitting domains until one value has been singled out. By the monotonicity property of `CRoomPeriodIndex`, this has the effect of attempting to place events close to the middle period before attempting to place them farther away from the middle.

The choice of Luby as opposed to other restart schemes and of the parameter values was made after preliminary experimentation.

4.3 Mixed integer programming with MiniZinc

In this section, we define MIP^{MZ} , another MIP model for the problem. It is solved with the Gurobi backend (see Gurobi Optimization, LLC, 2021). MiniZinc has rich support for binarizing multiple-value domains and linearizing constraints to make models suitable for execution by MIP solvers. We have not relied much on that support; instead, our model uses 0–1 variables and linear constraints almost exclusively. In fact, MIP^{MZ} was derived from the `CP` model by manual binarization and linearization.

The key difference between `CP` and MIP^{MZ} are in the decision variables, which are all 0–1, as follows. `EventPeriodCRoom` is the main decision variable, which relates events, periods, and room classes. The `CP` counterparts are `EventPeriod` and `EventCRoom`.

```

array [1..Events,1..Periods,1..CRooms] of var 0..1: EventPeriodCRoom;
    
```

Another decision variable `CostDistance` facilitates the part of the objective function that depends on the temporal distance between events. Let i be a row of `CostKeys` containing a pair (e_1, e_2) of events and let δ be their temporal distance. Then `CostDistance`[i, d] = 1 if $d = \delta$ and 0 otherwise:

```
set of int: CostIndex = index_set_1of2(CostKeys);
set of int: Distance = index_set_2of2(CostVectors);
array [CostIndex, Distance] of var 0..1: CostDistance;
```

The MIP^{MZ} constraints consist of a straightforward linearization and adaptation of the CP constraints to the MIP^{MZ} variables.

4.4 Simulated annealing

The simulated annealing (SA) approach is an extension of the one proposed by Battistutta et al. (2020). In detail, to represent a state in the search space we use two vectors that store the period and the room of each event, respectively. Only periods in \mathcal{P}_e and rooms in \mathcal{R}_e can be assigned to event e , thus explicitly enforcing constraints H1 and H5. The other constraints, namely H2 (RoomOccupation), H3 (HardConflicts), and H4 (Precedences), can be violated and are included in the cost function with a high weight. The cost function is thus a linear combination of the soft constraints S1–S3 and a measure of the *distance to feasibility*, corresponding to the degree of violation of constraints H2, H3, and H4.

The initial solution is generated totally at random, except that it always satisfies constraints H1 and H5. This is obtained simply by drawing randomly period and room assignments for event e from \mathcal{P}_e and \mathcal{R}_e , respectively.

Regarding the neighborhood relation, Battistutta et al. (2020) employed the one, called MEE (MoveEventOrExam), that moves either a single event (with probability $1 - p_b$) or the two events associated with a composite examination jointly (with probability p_b).

We also use MEE, but in combination with a new neighborhood, called SE (SwapEvents), which swaps period and room of two single events. Only events that do not belong to a composite examination are included in the SE neighborhood.

At each iteration a random move is drawn from the neighborhood $\text{MEE} \cup \text{SE}$. The move selection is biased on the basis of a parameter p_s (called *swap rate*), so that a SE move is selected with probability p_s , and a MEE move with probability $1 - p_s$. The parameters p_b and p_s are fixed experimentally, according to the tuning procedure. The drawing of the specific move inside the selected neighborhood is made according to a uniform distribution.

As customary for SA, we use the Metropolis acceptance criterion: A move is always accepted if it is improving or sideways (i.e., same cost), whereas it is accepted based on a time-decreasing exponential distribution $e^{-\Delta/T}$ in case it is

worsening, where Δ is the difference of total cost induced by the move, and T is the *temperature*.

The temperature starts at the initial value T_0 and evolves according to the standard geometric cooling scheme of SA, with the cutoff mechanism. That is, it is decreased during the search by multiplying it by a value α (with $0 < \alpha < 1$) after a fixed number of samples N_s have been drawn, or a fixed number of moves N_a have been accepted.

For the tuning procedure, we decided to use as stop criterion the total number of iterations \mathcal{I} , in order to keep the running time approximately equal for all configurations of the parameters. In our experiments, we fixed $\mathcal{I} = 10^8$, corresponding to an average time of approximately 660 s per run.

The tuning procedure has been performed using the tool JSON2RUN (Urli, 2013), which samples the configurations using the *Hammersley point set* (Hammersley & Handscomb, 1964) and implements the F-Race procedure (Birattari et al., 2010) for comparing them.

The winning configuration turned out to be: $T_0 = 188.89$, $\alpha = 0.875$, $N_s = 2092772$, $N_a = 292988$, $p_b = 0.967$, and $p_s = 0.288$.

For the experiments in the comparison with the other techniques, in order to have the same fixed running time on all instances, we use an additional stop criterion based on total running time.

5 Experimental analysis

We first introduce the dataset employed in the analysis (Sect. 5.1), then we show the settings used for the comparison (Sect. 5.2), and finally, we report and discuss the experimental results.

5.1 Problem instances

The dataset is composed of real-world instances extracted from various Italian universities and collected by Battistutta et al. (2020). They are written in JSON file format but are also made available in `dzn` format, thanks to a preprocessing procedure that splits courses into single events and distributes constraints accordingly.

In reference to the extraction and preprocessing procedures, we have corrected a few minor issues in the code of Battistutta et al. (2020) so that the instances that we use here are actually slightly revised with respect to the ones used by Battistutta et al. (2020).

The repository https://bitbucket.org/satt/examtimetablin_gnuiddata contains both the original instances and the revised ones. In addition, it also contains a *software toolbox*, written in Python, that allows us to check the validity of instances and solutions. The software has been written

Table 6 Instance features

Instance	Courses	Events	Periods	Timeslots	Single rooms	Composite rooms
D1-1-16	261	261	40	2	64	0
D1-1-17	247	247	46	2	65	0
D1-2-16	254	254	36	2	64	0
D1-2-17	281	281	38	2	65	0
D1-3-16	239	239	26	2	65	0
D1-3-17	254	254	34	2	65	0
D1-3-18	258	258	52	2	64	0
D2-1-18	57	62	156	6	0	0
D2-2-18	58	61	162	6	0	0
D2-3-18	58	61	204	6	0	0
D3-1-16	81	164	188	4	14	3
D3-1-17	89	177	188	4	15	3
D3-1-18	87	174	188	4	15	3
D3-2-16	76	78	48	4	14	3
D3-2-17	87	88	48	4	15	3
D3-2-18	82	84	48	4	15	3
D3-3-16	78	80	48	4	14	3
D3-3-17	84	85	48	4	15	3
D3-3-18	81	83	48	4	15	3
D4-1-17	234	361	80	2	34	0
D4-1-18	223	476	80	2	34	0
D4-2-17	226	482	88	2	34	0
D4-2-18	238	514	86	2	34	0
D4-3-17	223	235	38	2	34	0
D4-3-18	240	260	38	2	34	0
D5-1-17	134	277	122	2	17	4
D5-1-18	148	311	136	2	20	4
D5-2-17	125	344	136	2	17	4
D5-2-18	156	426	122	2	20	4
D5-3-18	129	132	24	2	17	4
D6-1-16	189	487	66	2	29	41
D6-1-17	194	494	80	2	29	41
D6-1-18	198	511	80	2	29	41
D6-2-16	193	511	78	2	29	41
D6-2-17	195	501	88	2	29	41
D6-2-18	207	539	90	2	29	41
D6-3-16	192	346	58	2	29	41
D6-3-17	192	350	52	2	29	41
D7-1-17	63	150	155	5	22	0
D7-2-17	60	136	330	10	22	0

independently from the solvers so to be used as a sort of *third-party* solution checker in the spirit of the methodology outlined in (Bonutti et al., 2012). Besides allowing the debugging of the different solution approaches presented in this paper, this toolbox will provide against misinterpretations of the different constraints in case of future works on the same problem by different researchers, thus enabling the

comparability of results. In addition, the software provides a format translator between the original (and richer) JSON format to the event-based *dzn* format and a tool for computing a set of features from both the instances and the solutions.

Related to this last functionality, Table 6 shows the main features of the revised instances in terms of the total number of courses, events, periods, single rooms and composite

rooms, and the number of timeslots (i.e., periods in a day). The name of each instance follows this pattern: $Dx-y-z$, where x is the department identifier, y is the examination season, and z is the academic year. It can be noticed that instances of the same department are quite homogeneous, except for D3, D4, and D5, where the number of events and periods changes during the year depending on the season.

5.2 Setting and tuning

We run all experimental tests in a high-performance computing cluster on 64bit computers running Scientific Linux 7.7. We use computers equipped with 756GB RAM and two Intel Xeon Gold 6226R CPUs clocked at 2.90GHz. We run the MiniZinc model using MiniZinc version 2.5.5 (see Nethercote et al., 2007) and Gecode version 6.3.0 (see Gecode Team, 2019). To solve MIP models, we use Gurobi 9.1.0 (see Gurobi Optimization, LLC, 2021). The simulated annealing procedure has been implemented in C++ and compiled using GNU g++ (v. 9.2.0).

For all tests, except for lower bounds (LB), we have ten runs using a single thread for an hour. To get lower bounds, we have five runs using four threads and a runtime of 24h. Table 7 shows the parameter settings used for each test. The Gurobi Presolve parameter controls the level of presolve, and a value of 2 indicates “aggressive” and 0 turns it off. The MIPFocus modifies Gurobi’s high-level search strategy, such that a value of 1 instructs Gurobi to focus on finding solutions and 3 to focus on improving the bound.

Through testing we found that when running **MIP** (the full model discussed in Sect. 4.1), we get better results when using the MIP variant of the model as opposed to the IP variant. The **MIP_{2S}** gets better results when using the IP. Additionally, testing revealed that the best time distribution for **MIP_{2S}** was to run the first stage for 99% of the available time, leaving just 36 s for the second stage. It is generally easy for the solver to add rooms in the second stage warm start without incurring in any room preference penalties. Thus time is better spent in the first stage. We skip presolving as there is no benefit given such a short time limit, and it is better to try and eliminate any introduced room preference penalties. Finally, we use the IP model variant for getting lower bounds, as this model generally finds better bounds.

When running **MIP^{MZ}**, we leave the presolve parameter to the default value, as an aggressive presolve strategy on this model leads to a very long presolving phase on most instances, resulting in worse performance.

For **SA**, we used the winning configuration of parameters shown in Sect. 4.4, except for N_s and N_a , which have been increased in order to have one hour running time.

Table 7 Parameters used for testing

Method	Parameters
SA	$T_0 = 188.89$, $\alpha = 0.875$, $N_s = 6278316$ $N_a = 878964$, $p_b = 0.967$, and $p_s = 0.288$
CP	Default
MIP^{MZ}	MIPFocus = 1
MIP	Presolve = 2, MIPFocus = 1
MIP_{2S}	Stage 1: Presolve = 2, MIPFocus = 1 Stage 2: Presolve = 0, MIPFocus = 1
LB	Presolve = 2, MIPFocus = 3

5.3 Comparative results

Table 8 reports the average results of ten runs of each technique with a timeout of one hour. The second column represents the results of the code by Battistutta et al. (2020) (with some minor corrections), which we rerun on this test computer with the same timeout. The column Best reports the best-known results, obtained using all performed experiments.

First, we notice that **SA** improves upon the previous version on all instances but two, although the gap is relatively small. The results of **CP** are somewhat inferior, with some cases in which they are particularly poor. The situation is more extreme for the other MiniZinc-based model **MIP^{MZ}**, which has some good results, but in other cases, they are extremely bad or even with no solution returned within the given timeout. A more stable behavior could perhaps be obtained by using a lazy clause generation solver, which Gecode is not (see Ohrimenko et al., 2009), but unfortunately, at this time, none is available that also has support for LNS.

Regarding the **MIP** and **MIP_{2S}** methods, unsurprisingly, the two-stage method **MIP_{2S}** has (generally) better performance than the single-stage one **MIP**, as, like **CP**, it makes use of a preprocessing step that removes one dimension of the problem (i.e., the rooms), which has little effect on solution value. The fact that **MIP_{2S}** works better than **MIP** is because, in our instances, rooms are not a critical resource. Indeed, in most current cases, room endowment is sized for the course lectures, which normally require more space than the examinations, so it turned out oversized for the examinations. Nonetheless we might expect future cases in which rooms are more critical.

Finally, we notice that **MIP^{MZ}** and **MIP_{2S}** provide more robust results than **SA** on two instances, namely D4-1-17 and D6-3-17, consistently obtaining the best values (276 and 30, respectively).

Regarding the lower bounds, we see that, besides the cases in which there is a perfect solution (cost 0), they are quite

Table 8 Comparative results and lower bounds

Instance	SA	Batt. et al.	CP	MIP ^{MZ}	MIP	MIP _{2S}	Best	LB
D1-1-16	385.0*	393.9	414.2	388.0	399.0	394.0	381	192
D1-1-17	320.3*	328.2	370.9	323.0	329.0	324.0	318	181
D1-2-16	524.6*	535.0	561.8	543.0	554.4	536.0	521	216
D1-2-17	613.0*	625.7	661.9	690.0	654.0	631.0	609	233
D1-3-16	724.9*	733.5	757.9	767.4	741.0	729.0	720	218
D1-3-17	614.2*	622.3	648.3	634.0	639.0	621.0	612	221
D1-3-18	265.0*	266.8*	279.6	270.0	272.0	266.0	264	190
D2-1-18	427.6*	426.8*	440.8	464.0	448.0	N/A	426	4
D2-2-18	22.0*	22.0*	27.4*	N/A	22.0*	N/A	22	20
D2-3-18	22.0*	22.0*	22.0*	N/A	22.0*	N/A	22	10
D3-1-16	0.0*	0.0*	1.2*	0.0*	0.0*	0.0*	0	0
D3-1-17	0.0*	0.0*	0.6*	0.0*	0.0*	0.0*	0	0
D3-1-18	0.0*	0.0*	0.2*	0.0*	0.0*	0.0*	0	0
D3-2-16	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0	0
D3-2-17	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0	0
D3-2-18	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0	0
D3-3-16	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0	0
D3-3-17	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0	0
D3-3-18	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0	0
D4-1-17	276.6*	278.8*	296.3	276.0*	278.0	276.0*	276	256
D4-1-18	1037.3*	1060.1	1174.2	7984.1	1322.5	1156.1	1028	409
D4-2-17	1126.0*	1150.8	1320.7	–	1533.9	1490.6	1105	459
D4-2-18	1594.5*	1616.8	2360.6	–	–	2099.6	1579	530
D4-3-17	373.6*	381.1	403.3	381.0	383.6	377.0	372	224
D4-3-18	677.4*	691.9	757.1	826.0	802.0	752.0	670	265
D5-1-17	157.0*	158.1*	198.1	1261.0	218.0	160.2	156	0
D5-1-18	36.6*	38.2*	99.2	327.4	80.2	38.0	36	0
D5-2-17	60.0*	60.0*	81.6	–	296.0	64.0	60	0
D5-2-18	271.6	272.2*	339.3	–	629.2	356.0	264	0
D5-3-18	0.0*	0.0*	0.0*	0.0*	0.0*	0.0*	0	0
D6-1-16	443.3	442.0*	891.2	23194.0	1266.2	519.8	432	61
D6-1-17	367.1*	377.0	621.8	–	577.0	637.0	360	83
D6-1-18	400.2*	405.7*	1019.2	–	565.6	493.0	392	75
D6-2-16	516.1*	527.2	1792.9	–	741.9	573.4	506	76
D6-2-17	564.8	569.8*	1667.9	–	1095.3	715.1	558	80
D6-2-18	202.9*	214.6	590.9	–	–	280.8	199	56
D6-3-16	27.0*	27.0*	34.1	27.0*	27.0*	27.0*	27	27
D6-3-17	30.2*	30.3*	43.5	30.0*	30.0*	30.0*	30	26
D7-1-17	395.7*	397.5*	479.3	484.2	524.2	435.0	386	40
D7-2-17	764.2	764.4*	847.8	6161.2	949.4	938.0	758	10

Results in bold indicate best average solution values. Values marked with (*) show that the given method found the best solution for that instance

tight (below 20%) only in four cases. In particular, in one case the lower bound is equal to the best solution, thus proving its optimality.

6 Conclusions and future work

We have investigated three independent optimization paradigms, namely CP, MIP, and SA, for the real-world examination timetabling problem proposed by Battistutta et al. (2020).

For MIP, we have developed three different versions (MIP^{MZ}, MIP, and MIP_{2S}), thus resulting in five total alternative search methods that we compared among themselves and with the original SA of Battistutta et al. (2020). Even though the techniques have been developed independently (and by different authors), they have been run on the same machine with the same timeout, to have a fair competition ground. In addition, we also computed some valid lower bounds.

Unsurprisingly, the metaheuristic approach, properly tuned, turned out to work better on the majority of instances, but the exact techniques are close and actually better on a few instances.

All instances and solutions are available for inspection and future comparison, along with the solution checker that reports the value of the objective function.

In the future, we will investigate the correlation of the results of the different search methods with the features of the specific instances. To this aim, we plan to collect new real-world instances and possibly develop a principled instance generator to study such correlation on a potentially unlimited number of instances.

In addition, we plan to develop hybrid techniques that could benefit from the respective advantages of the different search methods. These hybrid techniques would range from simple ones using SA for warm-starting MIP and CP, to more complex interaction mechanisms such as Benders (1962) decomposition or the *matheuristic* paradigm (see Maniezzo et al., 2021).

References

- Abou Kasm, O., Mohandes, B., Diabat, A., & El Khatib, S. (2019). Exam timetabling with allowable conflicts within a time window. *Computers & Industrial Engineering*, *127*, 263–273.
- Al-Hawari, F., Al-Ashi, M., Abawi, F., & Alouneh, S. (2020). A practical three-phase ILP approach for solving the examination timetabling problem. *International Transactions in Operational Research*, *27*(2), 924–944.
- Al-Yakoob, S. M., Sherali, H. D. (2015). Mathematical models and algorithms for a high school timetabling problem. *Computers & Operations Research*, *61*, 56–68.
- Al-Yakoob, S. M., Sherali, H. D., & Al-Jazzaf, M. (2010). A mixed-integer mathematical modeling approach to exam timetabling. *Computational Management Science*, *7*(1), 19.
- Arbaoui, T., Boufflet, J. P., & Moukrim, A. (2015). Preprocessing and an improved MIP model for examination timetabling. *Annals of Operations Research*, *229*(1), 19–40.
- Arbaoui, T., Boufflet, J. P., & Moukrim, A. (2019). Lower bounds and compact mathematical formulations for spacing soft constraints for university examination timetabling problems. *Computers & Operations Research*, *106*, 133–142.
- Battistutta, M., Ceschia, S., De Cesco, F., Di Gaspero, L., Schaerf, A., & Topan, E. (2020). Local search and constraint programming for a real-world examination timetabling problem. In E. Hebrard, N. Musliu (Eds.). *17th International conference on the integration of constraint programming, artificial intelligence, and operations research (CPAIOR-2020)*, LNCS, Vol. 12296 (pp. 69–81). Springer.
- Battistutta, M., Schaerf, A., & Urli, T. (2017). Feature-based tuning of single-stage simulated annealing for examination timetabling. *Annals of Operations Research*, *252*(2), 239–254.
- Bellio, R., Ceschia, S., Di Gaspero, L., & Schaerf, A. (2021). Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. *Computers and Operations Research*, *132*, 1–9.
- Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*(1), 238–252.
- Bilgin, B., Özcan, E., & Korkmaz, E. E. (2006). An experimental study on hyper-heuristics and exam timetabling. In *International conference on the practice and theory of automated timetabling* (pp. 394–412). Springer.
- Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-race and iterated F-race: An overview. In *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Springer.
- 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.
- Bron, C., & Kerbosch, J. (1973). Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, *16*(9), 575–577.
- Burke, E. K., & Bykov, Y. (2016). An adaptive flex-deluge approach to university exam timetabling. *INFORMS Journal on Computing*, *28*(4), 781–794.
- Carter, M. W., Laporte, G., & Lee, S. Y. (1996). Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, *74*, 373–383.
- Cataldo, A., Ferrer, J. C., Miranda, J., Rey, P. A., & Sauré, A. (2017). An integer programming approach to curriculum-based examination timetabling. *Annals of Operations Research*, *258*(2), 369–393.
- Daskalaki, S., & Birbas, T. (2005). Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research*, *160*(1), 106–120.
- Dekker, J. J., Björdal, G., Carlsson, M., Flener, P., & Monette, J. (2017). Auto-tabling for subproblem presolving in MiniZinc. *Constraints*, *22*(4), 512–529.
- Dekker, J.J., de la Banda, M.G., Schutt, A., Stuckey, P.J., & Tack, G. (2018). Solver-independent large neighbourhood search. In J. N. Hooker (Ed.) *CP 2018*, LNCS, Vol. 11008 (pp. 81–98). Springer.
- Demeester, P., Bilgin, B., De Causmaecker, P., & Vanden Berghe, G. (2012). A hyperheuristic approach to examination timetabling problems: Benchmarks and a new problem from practice. *Journal of Scheduling*, *15*(1), 83–103.
- Gecode Team. (2019). Gecode: A generic constraint development environment. The Gecode solver and its MiniZinc backend are available at <https://www.gecode.org>

- Genc, B., & O'Sullivan, B. (2020). A two-phase constraint programming model for examination timetabling at University College Cork. In *International conference on principles and practice of constraint programming* (pp. 724–742). Springer.
- Güler, M. G., Geçici, E., Koroğlu, T., & Becit, E. (2021). A web-based decision support system for examination timetabling. *Expert Systems with Applications*, 183, 1–11.
- Gurobi Optimization, LLC. (2021). *Gurobi Optimizer Reference Manual*. <https://www.gurobi.com>
- Hammersley, J. M., & Handscomb, D. C. (1964). *Monte Carlo methods*. Chapman and Hall.
- June, T. L., Obit, J. H., Leau, Y. B., & Bolongkikit, J. (2019). Implementation of constraint programming and simulated annealing for examination timetabling problem. In R. Alfred, Y. Lim, A. Ibrahim, & P. Anthony (Eds.), *Computational Science and Technology. Lecture Notes in Electrical Engineering* (pp. 175–184). Springer.
- Kahar, M. M., & Kendall, G. (2010). The examination timetabling problem at Universiti Malaysia Pahang: Comparison of a constructive heuristic with an existing software solution. *European Journal of Operational Research*, 207(2), 557–565.
- Kahar, M. M., & Kendall, G. (2015). A great deluge algorithm for a real-world examination timetabling problem. *Journal of the Operational Research Society*, 66(1), 116–133.
- Keskin, M. E., Döyen, A., Akyer, H., & Güler, M. G. (2018). Examination timetabling problem with scarce resources: A case study. *European Journal of Industrial Engineering*, 12(6), 855–874.
- Kristiansen, S., Sørensen, M., & Stidsen, T. R. (2015). Integer programming for the generalized high school timetabling problem. *Journal of Scheduling*, 18(4), 377–392.
- Lach, G., & Lübbecke, M. E. (2008). Optimal university course timetables and the partial transversal polytope. In *International workshop on experimental and efficient algorithms* (pp. 235–248). Springer.
- Lach, G., & Lübbecke, M. E. (2012). Curriculum based course timetabling: New solutions to Udine benchmark instances. *Annals of Operations Research*, 194(1), 255–272.
- Leite, N., Fernandes, C., Melício, F., & Rosa, A. (2018). A cellular memetic algorithm for the examination timetabling problem. *Computers and Operations Research*, 94, 118–138.
- Leite, N., Melício, F., & Rosa, A. (2019). A fast simulated annealing algorithm for the examination timetabling problem. *Expert Systems with Applications*, 122, 137–151.
- Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4), 173–180.
- Maniezzo, V., Boschetti, M. A., & Stützle, T. (2021). *Matheuristics: Algorithms and implementations*. Springer.
- McCullum, B., McMullan, P., Burke, E. K., Parkes, A. J., & Qu, R. (2007). *The second international timetabling competition: Examination timetabling track*. Technical Report. QUB/IEEE/Tech/ITC2007/Exam/v4.0/17. Queen's University.
- Muklason, A., Parkes, A. J., Özcan, E., McCullum, B., & McMullan, P. (2017). Fairness in examination timetabling: Student preferences and extended formulations. *Applied Soft Computing*, 55, 302–318.
- Müller, T. (2016). Real-life examination timetabling. *Journal of Scheduling*, 19(3), 257–270.
- Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). MiniZinc: Towards a standard CP modelling language. In: C. Bessière (Ed). *CP 2007*, LNCS, Vol. 4741 (pp 529–543). Springer, the MiniZinc toolchain is available at <https://www.minizinc.org>
- Ohrimenko, O., Stuckey, P. J., & Codish, M. (2009). Propagation via lazy clause generation. *Constraints*, 14(3), 357–391.
- Özcan, E., & Ersoy, E. (2005). Final exam scheduler-fes. In *2005 IEEE congress on evolutionary computation*, Vol. 2 (pp. 1356–1363). IEEE.
- Parkes, A. J., & Özcan, E. (2010). Properties of Yeditepe examination timetabling benchmark instances. In *Proceedings of the 8th international conference on the practice and theory of automated timetabling* (pp 531–534).
- Qu, R., Burke, E. K., McCollum, B., Merlot, L., & Lee, S. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1), 55–89.
- Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review*, 13(2), 87–127.
- Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher, J. F. Puget (Eds.) *CP 1998*, LNCS, Vol. 1520 (pp. 417–431). Springer.
- Sørensen, M., & Dahms, F. H. (2014). A two-stage decomposition of high school timetabling applied to cases in Denmark. *Computers & Operations Research*, 43, 36–49.
- Stuckey, P. J., et al. (2020). *The MiniZinc Handbook*, 2nd edn. Monash University. <https://www.minizinc.org/>
- Urli, T. (2013). json2run: A tool for experiment design & analysis. [arXiv:1305.1112](https://arxiv.org/abs/1305.1112)
- Woumans, G., De Boeck, L., Beliën, J., & Creemers, S. (2016). A column generation approach for solving the examination-timetabling problem. *European Journal of Operational Research*, 253(1), 178–194.

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

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.