University course timetabling and International Timetabling Competition 2019

Tomáš Müller · Hana Rudová · Zuzana Müllerová

Abstract University course timetabling belongs to classical problems which have been studied for many years by many researchers. This paper will outline existing research and emphasize new research directions and challenges in this area. It is clear that the organization of international competitions has a high impact on the timetabling research. We intend to discuss the organization of the new International Timetabling Competition (ITC 2019) with the aim to motivate further research on complex university course timetabling problems coming from practice. Our goal is the creation of rich real-world data sets. Thanks to the UniTime timetabling system, we can collect a strong set of data with diverse characteristics which we will discuss in the paper. The key novelty lies in the combination of student sectioning together with standard time and room assignment of events in courses. To make the problems more attractive, we remove some of the less important aspects of the real-life data while retaining the computational complexity of the problems.

Keywords University course timetabling · Competition · Real-world problems

T. Müller
Purdue University
West Lafayette, Indiana, USA
E-mail: muller@unitime.org

H. Rudová
Faculty of Informatics, Masaryk University
Brno, Czech Republic
E-mail: hanka@fi.muni.cz

Z. Müllerová
UniTime, s.r.o.
Zlín, Czech Republic
E-mail: mullerova@unitime.cz

1 Introduction

Educational timetabling problems have been widely studied for many years [28, 30]. As the research evolved, a conference series on the Practice and Theory of Automated Timetabling (PATAT) started in 1995 for the international community working on all aspects of computer-aided timetable generation [2]. As years went by, various surveys were published concentrating on different aspects of timetabling. Schaerf [27] presented problems of school timetabling, university course timetabling and examination timetabling. In [6], Burke and Petrovic discussed many approaches to timetabling problems developed in the Automated scheduling, optimization, and planning research group at the University of Nottingham. Further surveys often concentrated on specific areas such as metaheuristic-based techniques [12], school timetabling [20] or curriculum-based timetabling [3], a more recent general study on educational timetabling was given by Kingston in [11].

Starting in 2002, the research in timetabling has been highly influenced by the organization of international competitions. The European Metaheuristics Network prepared the first International Timetabling Competition (ITC 2002) [19] which focused on a simplified version of the university course timetabling problem. The next ITC 2007 competition [15] was able to introduce three tracks on curriculum-based timetabling [9, 4], post-enrollment timetabling [13], and examination timetabling [14]. It is interesting to see that results for curriculum-based timetabling are still maintained at the website [5] where various data sets were added until 2014, and new results are still updated. Thanks to this strong support of curriculum-based timetabling, many studies focused on this problem as demonstrated in [3]. Research in another area of educational timetabling was encouraged by the ITC 2011 competition [23] where high-school timetabling problems were solved. Again the website is still maintained [21], and a rich body of research has been initiated. An XML format for benchmarks has been proposed [22], an archive is maintained [24] and various groups have started to work on this problem [20].

Three other competitions are listed on the PATAT web pages [2]. The competition organized in 2010 [10] was oriented on nurse rostering problems with the similar goals as the previous educational timetabling competitions. The competition wanted to generate new approaches by attracting researchers from different areas of research, reduce the gap between research and practice and stimulate debate within the research community. This competition introduced an interesting proposal on how to handle different solving times for instances of various size and complexity. It considered three tracks: the sprint track required interactive use, the middle distance track allowed a few minutes, and the long distance track simulated overnight solving. The second International Nurse Rostering Competition (INRC-II) [8] emphasized another important feature of nurse rostering problems: the multi-stage problem formulation referring to consecutive weeks of a longer planning horizon taking 4 or 8 weeks. The last competition currently published at the PATAT web pages [2] is the first Cross-domain Heuristic Search Challenge (CHeSC 2011) [7] which aimed to measure performance over several problem domains rather than just one. HyFlex [18] software framework was proposed to deal with different combinatorial optimization problems. The challenge was to design a search algorithm that works well across different problem domains.

We will organize a new competition focusing on the university course timetabling. Following the tradition of previous timetabling competitions, it will be called the International Timetabling Competition 2019 (ITC 2019). Information about it will be available through the website `http://www.itc2019.org`. Our goal is the creation of rich real-world data sets which would encourage new research directions in the theory and practice of automated timetabling. The competition is supported by the PATAT conference and by the EURO working group on Automated Timetabling (EWG PATAT) [1]. The PATAT will provide one free registration to the PATAT 2020 conference for the winner and 50 % and 25 % discount of the conference fee for the 2nd and 3rd place, respectively. The EWG PATAT will provide 500 EUR for the winner. We will also organize a track allocated for the competition at the PATAT 2020 conference.

The next section provides a general specification of the timetabling problems that we intend to work with in the competition. Section 3 states the problem precisely and proposes an XML format of the data sets. Further, we will specify distinct features of our problems in Section 4.1 and necessary transformations of real-life data in Section 4.2. Organization of the competition is described in Section 5 and final comments are given at the end.

2 Our timetabling problems

Our educational timetabling problems are closely related to the post-enrollment and curriculum-based timetabling problems introduced in the ITC 2007. We focus on university course timetabling in which the aim is to find an optimal assignment of times, rooms, and students to events related to the set of courses. As it is common, various hard (or required) constraints ensure the feasibility of the timetable and soft constraints define optimization criteria and the quality of the timetable. Roots of our new competition are related to the ITC solver [16] which was the winner of the two tracks in ITC 2007 and a finalist in the last track. The solver was initially developed to solve course timetabling problems at Purdue University [26]. It has been applied there to handle timetabling as a single university-wide timetabling problem [25]. Based on that, the comprehensive timetabling system UniTime [29] has been developed as an open source project and implemented at many universities worldwide. Thanks to UniTime, we have access to many diverse timetabling problems from various countries. We already have an agreement with ten institutions including Purdue University in the USA, Masaryk University in the Czech Republic, AGH University of Science and Technology in Poland and Istanbul Kültür University in Turkey that we can use their data. We intend to have problems with different characteristics, some of which were already described in our earlier works published about the timetabling at Purdue University [25] and Masaryk University [17]. Further distinct features are described in Section 4.1.

All data will be anonymized. All data sets will be available in the XML format which is described along with the considered timetabling problem in Section 3. A sample XML file is also provided in the Appendix.

Important parts of the preparation work are the selection and transformation of the problem characteristics which will be included in the competition. The real-life problem formulations are too complicated to be of interest for a wider timetabling community. To make the problems more attractive, we plan to remove some of the less important aspects of the real-life data while retaining the computational complexity of the problems. The key transformations and changes are given in Section 4.2.

3 Problem description

In this section, we describe particular components of the problem as they are defined in the XML format. A sample XML is available in Appendix. The 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 request, respecting various constraints and preferences.

Optimization needs a specification of costs, weights or preferences. In the presented problem, all costs are specified as non-negative integer penalties. A smaller penalty (meaning a smaller violation) is considered better. The zero penalty represents complete satisfaction. Penalties of particular components are summarized. The best possible solution could have zero penalty, but such a solution does not typically exist. Later on, we will discuss that summarized penalties for different criteria (time, room, ...) are weighted to introduce their proper normalization.

For example, each class has a list of available time and room assignments given in the specified problem, each with a penalty. This means, for example, that only rooms that are big enough for the class and that meet all the other requirements (room type, equipment, building, etc.) are listed. Similarly, all the preferences have been already combined into a single penalty that is incurred when the time or room is assigned (see also details about problem transformations in Section 4.2).

3.1 Times

Each problem has a certain number of weeks `nrWeeks`, a number of days `nrDays` in each week, and a number of time slots per day `slotsPerDay`. These parameters, together with the instance name are defined on the root element of the XML file as shown in Figure 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem name="unique-instance-name" nrDays="7" nrWeeks="13" slotsPerDay="288">
    <optimization ... />
    <rooms> ... </rooms>
    <courses> ... </courses>
    <distributions> ... </distributions>
    <students> ... </students>
</problem>
```

Fig. 1 XML specification of the problem

The example in Figure 1 specifies that the semester has `13` weeks and `7` days a week, e.g., from Monday to Sunday. While the data format allows for variations, in all the competition instances one time slot takes 5 minutes, which allows us to model travel times as well as classes that meet at irregular times. There are `288` time slots covering the whole day, from midnight to midnight.

A class can meet several times a week during certain weeks of the semester. In such a case, all meetings of a class start at the same time, run for the same number of slots and are placed in the same room. Examples of possible time assignments of a class are listed in Figure 2. Each time is specified by its starting time slot `start` within a day and the duration `length` in the number of time slots. Days and weeks with a meeting are specified using the `days` and `weeks` binary strings. For example, days `1010100` means that the class meets three times a week (on Monday, Wednesday, and Friday) each week when it is meeting. Similarly, weeks `0101010101010` specifies that the class would only meet during even weeks of the semester (during the 2nd, the 4th, . . . , and the 12th week of the semester).

```
<class id="1" limit="20">
  <!-- Monday-Wednesday-Friday 7:30 - 8:20 All Weeks -->
  <time days="1010100" start="90" length="10" weeks="1111111111111" penalty="20"/>
  <!-- Tuesday-Thursday 9:00 - 10:15 All Weeks -->
  <time days="0101000" start="108" length="15" weeks="1111111111111" penalty="0"/>
  <!--Thursday 8:00 - 9:50 Even Weeks-->
  <time days="0001000" start="96" length="22" weeks="0101010101010" penalty="2"/>
</class>
```

Fig. 2 XML specification of possible times when a class can meet

Having this representation of time, we define what it means when two classes overlap. Two classes overlap when they share at least one week, at least one day of the week and they overlap within this day in their time slots using the start and length of the classes. More information can be found in the description of distribution preferences (see Section 3.5).

3.2 Rooms

Each `room` is specified by its `id` and `capacity`. A room may not be available at certain times, which is given by the `unavailable` elements using the `days` of the week, the `start` time slot and its `length`, during the `weeks` of the semester. Non-zero travel times from other rooms are specified by the `travel` elements from a `room` having a certain `value` which expresses the number of time slots that it takes to get from one room to the other. Travel times are expected to be symmetric and are listed only on one of the rooms. See Figure 3 for an example.

```
<rooms>
  <room id="1" capacity="50"/>
  <room id="2" capacity="100">
    <travel room="1" value="2"/> <!-- travel time is in the number of slots -->
  </room>
  <room id="3" capacity="80">
    <travel room="2" value="3"/> <!-- only non-zero travel times are present -->
    <!-- not available on Mondays and Tuesdays betwen 8:30 - 10:30, all weeks -->
    <unavailable days="110000" start="102" length="24" weeks="1111111111111"/>
    <!-- not available on Fridays betwen 12:00 - 24:00, odd weeks only -->
    <unavailable days="000100" start="144" length="144" weeks="1010101010101"/>
  </room>
</rooms>
```

Fig. 3 XML specification of the rooms available, including their availability and travel times

A class cannot be placed in a room when its assigned time overlaps with an unavailability of the room or when there is some other class placed in the room at an overlapping time. Besides available times, each class also has a list of rooms where it can be placed as shown in Figure 4. Each class needs one time and one room assignment from its list of possible assignments. It is possible for a class to only need a time assignment and no room. In this case, there are no rooms listed and the `room` attribute of the class is set to `false`.

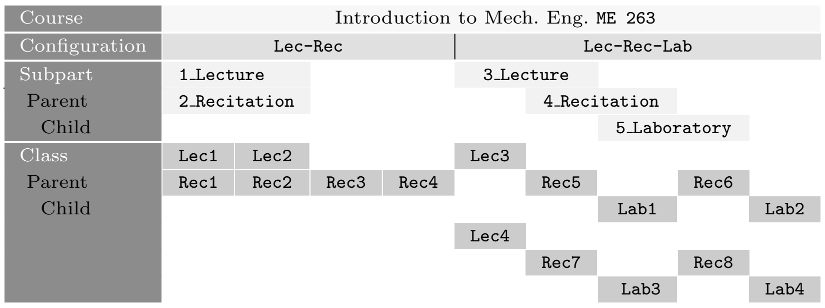
```
<class id="1" limit="20">
  ...
  <room id="1" penalty="20"/>
  <room id="3" penalty="0"/>
</class>
<class id="2" limit="10" rooms="false"/>
```

Fig. 4 XML specification of possible rooms where a class can be placed

3.3 Courses

Courses may have a very complex hierarchical structure of classes, i.e., events to be scheduled. Example of one course `ME 263` with the corresponding XML specification is shown in Figure 5.

Each `course` (`ME 263` in Figure 5) has a unique `id` and consists of one or more configurations named `config` in the XML (`Lec-Rec` and `Lec-Rec-Lab`) and identified by their unique `id` such that each student attends (some)



```
<course id="ME 263">
  <config id="1"> <!-- Lec-Rec configuration, not linked (any Lec with any Lab) -->
    <subpart id="1_Lecture"> <!-- Lecture subpart -->
      <class id="Lec1" limit="100"/>
      <class id="Lec2" limit="100"/>
    </subpart>
    <subpart id="2_Recitation"> <!-- Recitation subpart -->
      <class id="Rec1" limit="50"/>
      <class id="Rec2" limit="50"/>
      <class id="Rec3" limit="50"/>
      <class id="Rec4" limit="50"/>
    </subpart>
  </config>
  <config id="2"> <!-- Lec-Rec-Lab configuration, linked -->
    <subpart id="3_Lecture"> <!-- Lecture subpart -->
      <class id="Lec3" limit="100"/>
      <class id="Lec4" limit="100"/>
    </subpart>
    <subpart id="4_Recitation"> <!-- Recitation subpart -->
      <class id="Rec5" parent="Lec3" limit="50"/>
      <class id="Rec6" parent="Lec3" limit="50"/>
      <class id="Rec7" parent="Lec4" limit="50"/>
      <class id="Rec8" parent="Lec4" limit="50"/>
    </subpart>
    <subpart id="5_Laboratory"> <!-- Laboratory subpart -->
      <class id="Lab1" parent="Rec5" limit="50"/>
      <class id="Lab2" parent="Rec6" limit="50"/>
      <class id="Lab3" parent="Rec7" limit="50"/>
      <class id="Lab4" parent="Rec8" limit="50"/>
    </subpart>
  </config>
</course>
```

Fig. 5 Example of hierarchical course structure with its XML specification

classes in one configuration only. Each configuration consists of one or more `subparts` with their unique `id` (`Lec-Rec-Lab` configuration has three subparts `3_Lecture`, `4_Recitation` and `5_Laboratory`). Each student must attend one class from each subpart of a single configuration. All students of the course configuration must be sectioned into classes of each subpart such that their `limit` is not exceeded (one student attending configuration `Lec-Rec` must take one class from each of its subparts `1_Lecture` and `2_Recitation`, e.g., `Lec1` and `Rec3`). Each `class` has a unique `id` and belongs to one subpart (classes `Rec5`, `Rec6`, `Rec7`, and `Rec8` belong to subpart `4_Recitation`).

A class may have a parent class defined which means that a student which attends the class must also attend its parent class. For example, `Lab3` has the parent `Rec7` which has the parent `Lec4`. This means that a student attending `Lab3` must also attend `Rec7` and `Lec4` and no other combination including `Lab3` is allowed. On the other hand there is no parent-child relationship between classes of subparts `1_Lecture` and `2_Recitation`, so a student may take any lecture `Lec1` or `Lec2` and any recitation `Rec1`, `Rec2`, `Rec3` or `Rec1`.

In the described problem, the imposed course structure is needed only for student sectioning, to be able to evaluate the possible combinations of classes that a student needs to take. All the other constraints that could be derived from the structure are already included in the distribution constraints of the problem (see their description in Section 3.5) as each institution may decide to use a different set of constraints. These typically include:

– classes that are in a parent-child relation cannot overlap in time (`SameAttendees` constraint is required, e.g., between any valid Lecture–Laboratory–Seminar combination that a student can take),
– classes of a subpart need to be spread in time as much as possible (`NotOverlap` constraint is placed between these classes imposing a penalty for each pair of classes that overlap in time)
– the lecture may (or must) be placed before all the seminars (`Precedence` constraint is placed between any pair of a lecture and a seminar that a student can take).

Each class has defined a set of possible times when it can meet. Each eligible time has specified its `penalty` which must be included in the overall time penalization when the time is selected for the class (see Section 3.6). Valid time specifications were described in Section 3.1.

Each class has also defined a set of possible rooms where it can meet (room may not be possibly needed). Each eligible room has given its `penalty` to be included in the overall time penalization when selected (see Section 3.6). Valid room specifications were given in Section 3.2.

3.4 Students

Each `student` has a unique `id` and a list of courses that he or she needs to attend. Each course is specified by its `course id`. See Figure 6 for an example.

```
<student id="1">
  <course id="1"/>
  <course id="5"/>
</student>
<student id="2">
  <course id="1"/>
  <course id="3"/>
  <course id="4"/>
</student>
```

Fig. 6 XML specification of students and their courses

A student needs to be sectioned in one class of every subpart of a single configuration for each course from his or her list of courses. If a parent-child relation between classes of a course is specified, this relation must be respected as well (see also Section 3.3). Also, the number of students that can attend each class is constrained by the `limit` that is part of the `class` definition.

A student conflict occurs when a student is enrolled in two classes that overlap in time (they share at least one week and one day of the week and they overlap in time of a day) or they are one after the other in rooms that are too far apart. This means that the number of time slots between the two classes is smaller than the travel time value between the two rooms. Student conflicts are allowed and penalized. The same penalty of one student conflict occurs for any pair of classes that a student cannot attend, regardless of the number of actual meetings that are in conflict or the length of the overlapping time.

3.5 Distribution constraints

Besides the already described time and room constraints and student course demands, there are the following `distribution` constraints that can be placed between any two or more classes. Any of these constraints can be either hard or soft. Hard constraints cannot be violated and are marked as `required`. Soft constraints may not be satisfied and there is a `penalty` for each violation. See Figure 7 for example.

```xml
<distributions>
  <!-- classes 1 and 2 cannot overlap in time -->
  <distribution type="NotOverlap" required="true">
    <class id="1"/>
    <class id="2"/>
  </distribution>
  <!-- class 1 should be before class 3, class 3 before class 5 -->
  <distribution type="Precedence" penalty="2">
    <class id="1"/>
    <class id="3"/>
    <class id="5"/>
  </distribution>
</distributions>
```

Fig. 7 XML specification of the distribution constraints

The various distribution constraints are listed in the Table 1. Each constraint may affect the time of the day, the days of the week, the weeks of the semester, or the room assigned. Constraints from the upper two sections of the table are evaluated between individual pairs of classes. For example, if three classes need to be placed at the same starting time, such a constraint is violated if any two of the three classes start at different times. Distribution constraints from the last section of the table need to consider all classes for evaluation between which the constraint is created.

Table 1 List of distribution constraint types with their parameters, their potential affect on times, days, weeks, and rooms, as well as type of evaluation

Constraint	Opposite	Time	Days	Weeks	Room	Pairs
`SameStart`		$\surd$	–	–	–	$\surd$
`SameTime`	`DifferentTime`	$\surd$	–	–	–	$\surd$
`SameDays`	`DifferentDays`	–	$\surd$	–	–	$\surd$
`SameWeeks`	`DifferentWeeks`	–	–	$\surd$	–	$\surd$
`SameRoom`	`DifferentRoom`	–	–	–	$\surd$	$\surd$
`Overlap`	`NotOverlap`	$\surd$	$\surd$	$\surd$	–	$\surd$
`SameAttendees`		$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
`Precedence`		$\surd$	$\surd$	$\surd$	–	$\surd$
`WorkDay(S)`		$\surd$	$\surd$	$\surd$	–	$\surd$
`MinGap(G)`		$\surd$	$\surd$	$\surd$	–	$\surd$
`MaxDays(D)`		–	$\surd$	–	–	days over `D`
`MaxDayLoad(S)`		$\surd$	$\surd$	$\surd$	–	slots over `S`
`MaxBreaks(R,S)`		$\surd$	$\surd$	$\surd$	–	breaks over `R`
`MaxBlock(M,S)`		$\surd$	$\surd$	$\surd$	–	blocks over `M`

When any of the constraints that can be validated on pairs of classes is soft, the provided penalty is incurred for every pair of classes of the constraint that are in a violation. In other words, if M pairs of classes do not satisfy the distribution constraint, the total penalty for violation of this constraint is $M \times$ `penalty`. It means that the maximal penalty for violation of one distribution constraint is `penalty` $\times N \times (N-1)/2$, where N is the number of classes in the constraint.

`SameStart` Given classes must start at the same time slot, regardless of their days of week or weeks. This means that C_i`.start` $=$ C_j`.start` for any two classes C_i and C_j from the constraint; C_i`.start` is the assigned start time slot of a class C_i.

`SameTime` Given classes must be taught at the same time of day, regardless of their days of week or weeks. For the classes of the same length, this is the same constraint as `SameStart` (classes must start at the same time slot). For the classes of different lengths, the shorter class can start after the longer class but must end before or at the same time as the longer class. This means that

$$(C_i.\texttt{start} \leq C_j.\texttt{start} \ \wedge \ C_j.\texttt{end} \leq C_i.\texttt{end}) \vee$$
$$(C_j.\texttt{start} \leq C_i.\texttt{start} \ \wedge \ C_i.\texttt{end} \leq C_j.\texttt{end})$$

for any two classes C_i and C_j from the constraint; $C_i.\texttt{end} = C_i.\texttt{start} + C_i.\texttt{length}$ is the assigned end time slot of a class C_i.

`DifferentTime` Given classes must be taught during different times of day, regardless of their days of week or weeks. This means that no two classes of this constraint can overlap at a time of the day. This means that

$$(C_i.\texttt{end} \leq C_j.\texttt{start}) \ \vee \ (C_j.\texttt{end} \leq C_i.\texttt{start})$$

for any two classes C_i and C_j from the constraint.

`SameDays` Given classes must be taught on the same days, regardless of their start time slots and weeks. In case of classes of different days of the week, a class with fewer meetings must meet on a subset of the days used by the class with more meetings. For example, if the class with the most meetings meets on Monday–Tuesday–Wednesday, all others classes in the constraint can only be taught on Monday, Wednesday, and/or Friday. This means that

$$((C_i.\texttt{days} \text{ or } C_j.\texttt{days}) = C_i.\texttt{days}) \vee ((C_i.\texttt{days} \text{ or } C_j.\texttt{days}) = C_j.\texttt{days})$$

for any two classes C_i and C_j from the constraint; C_i.`days` are the assigned days of the week of a class C_i, doing binary "or" between the bit strings.

`DifferentDays` Given classes must be taught on different days of the week, regardless of their start time slots and weeks. This means that

$$(C_i.\texttt{days} \text{ and } C_j.\texttt{days}) = 0$$

for any two classes C_i and C_j from the constraint; doing binary ,,and" between the bit strings representing the assigned days.

`SameWeeks` Given classes must be taught in the same weeks, regardless of their time slots or days of the week. In case of classes of different weeks, a class with fewer weeks must meet on a subset of the weeks used by the class with more weeks. This means that

$$(C_i.\texttt{weeks} \text{ or } C_j.\texttt{weeks}) = C_i.\texttt{weeks}) \vee (C_i.\texttt{weeks} \text{ or } C_j.\texttt{weeks}) = C_j.\texttt{weeks})$$

for any two classes C_i and C_j from the constraint; doing binary "or" between the bit strings representing the assigned weeks.

`DifferentWeeks` Given classes must be taught on different weeks, regardless of their time slots or days of the week. This means that

$$(C_i.\texttt{weeks} \text{ and } C_j.\texttt{weeks}) = 0$$

for any two classes C_i and C_j from the constraint; doing binary ,,and" between the bit strings representing the assigned weeks.

`Overlap` Given classes overlap in time. Two classes overlap in time when they overlap in time of day, days of the week, as well as weeks. This means that

$$\begin{array}{ll}(C_j.\texttt{start} < C_i.\texttt{end}) & \wedge \\ (C_i.\texttt{start} < C_j.\texttt{end}) & \wedge \\ ((C_i.\texttt{days} \text{ and } C_j.\texttt{days}) \neq 0) & \wedge \\ ((C_i.\texttt{weeks} \text{ and } C_j.\texttt{weeks}) \neq 0) & \end{array}$$

for any two classes C_i and C_j from the constraint, doing binary ,,and" between days and weeks of C_i and C_j.

`NotOverlap` Given classes do not overlap in time. Two classes do not overlap in time when they do not overlap in time of day, or in days of the week, or in weeks. This means that

$$\begin{array}{ll}(C_i.\texttt{end} \leq C_j.\texttt{start}) & \vee \\ (C_j.\texttt{end} \leq C_i.\texttt{start}) & \vee \\ ((C_i.\texttt{days} \text{ and } C_j.\texttt{days}) = 0) & \vee \\ ((C_i.\texttt{weeks} \text{ and } C_j.\texttt{weeks}) = 0) & \end{array}$$

for any two classes C_i and C_j from the constraint, doing binary ,,and" between days and weeks of C_i and C_j.

`SameRoom` Given classes should be placed in the same room. This means that $(C_i.\texttt{room} = C_j.\texttt{room})$ for any two classes C_i and C_j from the constraint; $C_i.\texttt{room}$ is the assigned room of C_i.

`DifferentRoom` Given classes should be placed in different rooms. This means that $(C_i.\texttt{room} \neq C_j.\texttt{room})$ for any two classes C_i and C_j from the constraint.

`SameAttendees` Given classes cannot overlap in time, and if they are placed on overlapping days of week and weeks, they must be placed far enough so that the attendees can travel between the two classes. This means that

$$\begin{array}{ll}(C_i.\texttt{end} + C_i.\texttt{room.travel}[C_j.\texttt{room}] \leq C_j.\texttt{start}) & \vee \\ (C_j.\texttt{end} + C_j.\texttt{room.travel}[C_i.\texttt{room}] \leq C_i.\texttt{start}) & \vee \\ ((C_i.\texttt{days} \text{ and } C_j.\texttt{days}) = 0) & \vee \\ ((C_i.\texttt{weeks} \text{ and } C_j.\texttt{weeks}) = 0) & \end{array}$$

for any two classes C_i and C_j from the constraint; $C_i.\texttt{room.travel}[C_j.\texttt{room}]$ is the travel time between the assigned rooms of C_i and C_j.

`Precedence` Given classes must be one after the other in the order provided in the constraint definition. For classes that have multiple meetings in a week or that are on different weeks, the constraint only cares about the first meeting of the class. That is,

- the first class starts on an earlier week or
- they start on the same week and the first class starts on an earlier day of the week or
- they start on the same week and day of the week and the first class is earlier in the day.

This means that

```
$(\texttt{first}(C_i.\texttt{weeks}) < \texttt{first}(C_j.\texttt{weeks})) \vee$
  $[ (\texttt{first}(C_i.\texttt{weeks}) = \texttt{first}(C_j.\texttt{weeks})) \wedge$
    $[ (\texttt{first}(C_i.\texttt{days}) < \texttt{first}(C_j.\texttt{days})) \vee$
      $((\texttt{first}(C_i.\texttt{days}) = \texttt{first}(C_j.\texttt{days})) \wedge (C_i.\texttt{end} \leq C_j.\texttt{start}))$
    $]$
  $]$
```

for any two classes C_i and C_j from the constraint where $i < j$ and $\texttt{first}(x)$ is the index of the first non-zero bit in the binary string x.

`WorkDay(S)` There should not be more than `S` time slots between the start of the first class and the end of the last class on any given day. This means that classes that are placed on the overlapping days and weeks that have more than `S` time slots between the start of the earlier class and the end of the later class are violating the constraint. That is

$$\begin{array}{ll} ((C_i.\texttt{days} \text{ and } C_j.\texttt{days}) = 0) & \vee \\ ((C_i.\texttt{weeks} \text{ and } C_j.\texttt{weeks}) = 0) & \vee \\ (\max(C_i.\texttt{end}, C_j.\texttt{end}) - \min(C_i.\texttt{start}, C_j.\texttt{start}) \ \leq \ \texttt{S}) & \end{array}$$

for any two classes C_i and C_j from the constraint.

`MinGap(G)` Any two classes that are taught on the same day (they are placed on overlapping days and weeks) must be at least `G` slots apart. This means that there must be at least `G` slots between the end of the earlier class and the start of the later class. That is

$$\begin{array}{ll} ((C_i.\texttt{days} \text{ and } C_j.\texttt{days}) = 0) & \vee \\ ((C_i.\texttt{weeks} \text{ and } C_j.\texttt{weeks}) = 0) & \vee \\ (C_i.\texttt{end} + \texttt{G} \leq C_j.\texttt{start}) & \vee \\ (C_j.\texttt{end} + \texttt{G} \leq C_i.\texttt{start}) & \end{array}$$

for any two classes C_i and C_j from the constraint.

`MaxDays(D)` Given classes cannot spread over more than `D` days of the week, regardless whether they are in the same week of semester or not. This means that the total number of days of the week that have at least one class of this distribution constraint $C_1, \ldots, C_n$ is not greater than `D`,

$$\texttt{countNonzeroBits}(C_1.\texttt{days} \text{ or } C_2.\texttt{days} \text{ or } \cdots \ C_n.\texttt{days}) \leq \texttt{D}$$

where $\texttt{countNonzeroBits}(x)$ returns the number of non-zero bits in the bit string x. When the constraint is soft, the penalty is multiplied by the number of days that exceed the given constant `D`.

`MaxDayLoad(S)` Given classes must be spread over the days of the week (and weeks) in a way that there is no more than a given number of `S` time slots on every day. This means that for each week $w \in \{0, 1, \ldots, \texttt{nrWeeks} - 1\}$ of the semester and each day of the week $d \in \{0, 1, \ldots, \texttt{nrDays} - 1\}$, the total number of slots assigned to classes C that overlap with the selected day d and week w is not more than `S`,

$$\texttt{DayLoad}(d, w) \ \leq \ \texttt{S}$$

$$\texttt{DayLoad}(d, w) = \sum_i \{C_i.\texttt{length} \mid (C_i.\texttt{days} \,\text{and}\, 2^d) \neq 0 \ \wedge \ (C_i.\texttt{weeks} \,\text{and}\, 2^w) \neq 0)\}$$

where 2^d is a bit string with the only non-zero bit on position d and 2^w is a bit string with the only non-zero bit on position w. When the constraint is soft (it is not required and there is a penalty), its penalty is multiplied by the number of slots that exceed the given constant S over all days of the semester and divided by the number of weeks of the semester (using integer division). Importantly the integer division is computed at the very end. That is

$$\left(\texttt{penalty} \times \sum_{w,d} \max(\texttt{DayLoad}(d,w) - \texttt{S}, 0)\right) / \, \texttt{nrWeeks} \ .$$

MaxBreaks(R,S) This constraint limits the number of breaks during a day between a given set of classes (not more than R breaks during a day). For each day of week and week, there is a break between classes if there is more than S empty time slots in between.

Two consecutive classes are considered to be in the same block if the gap between them is not more than S time slots. This means that for each week $w \in \{0, 1, \ldots, \texttt{nrWeeks} - 1\}$ of the semester and each day of the week $d \in \{0, 1, \ldots, \texttt{nrDays} - 1\}$, the number of blocks is not greater than $\texttt{R} + 1$,

$$\begin{aligned} | \, \texttt{MergeBlocks}(\ \{ \ & (C.\texttt{start}, C.\texttt{end}) | \\ & (C.\texttt{days}\,\text{and}\,2^d) \neq 0 \ \wedge \ (C.\texttt{weeks}\,\text{and}\,2^w) \neq 0 \\ \} \ ) \, | \leq \texttt{R} + 1 & \end{aligned}$$

where 2^d is a bit string with the only non-zero bit on position d and 2^w is a bit string with the only non-zero bit on position w.

The MergeBlocks function recursively merges to the block B any two blocks B_a and B_b that are identified by their start and end slots that overlap or are not more than S slots apart, until there are no more blocks that could be merged.

$$\begin{aligned} & (B_a.\texttt{end} + \texttt{S} > B_b.\texttt{start}) \ \wedge \ (B_b.\texttt{end} + \texttt{S} > B_a.\texttt{start}) \ \Rightarrow \\ & (B.\texttt{start} = \min(B_a.\texttt{start}, B_b.\texttt{start})) \ \wedge \ (B.\texttt{end} = \max(B_a.\texttt{end}, B_b.\texttt{end})) \end{aligned}$$

When the constraint is soft, the penalty is multiplied by the total number of additional breaks computed over each day of the week and week of the semester and divided by the number of weeks of the semester at the end (using integer division, just like for the MaxDayLoad constraint).

MaxBlock(M,S) This constraint limits the length of a block of consecutive classes during a day (not more than M slots in a block). For each day of week and week, two consecutive classes are considered to be in the same block if the gap between them is not more than S time slots. For each block, the number of time slots from the start of the first class in a block till the end of the last class in a block must not be more than M time slots. This means that for each week $w \in \{0, 1, \ldots, \texttt{nrWeeks} - 1\}$ of the semester and each day of the week $d \in \{0, 1, \ldots, \texttt{nrDays} - 1\}$, the maximal length of a block does not exceed M slots

$$\begin{aligned} \max(\ \{ \ & B.\texttt{end} - B.\texttt{start} \, | \, B \in \texttt{MergeBlocks}(\{(C.\texttt{start}, C.\texttt{end}) \\ & | \, (C.\texttt{days}\,\text{and}\,2^d) \neq 0 \ \wedge \ (C.\texttt{weeks}\,\text{and}\,2^w) \neq 0\}) \\ \} \ ) \leq \texttt{M} & \end{aligned}$$

When the constraint is soft, the penalty is multiplied by the total number of blocks that are over the `M` time slots, computed over each day of the week and week of the semester and divided by the number of weeks of the semester at the end (using integer division, just like for the `MaxDayLoad` constraint).

3.6 Feasible and optimal solution

Solution It is guaranteed that a feasible solution exists for each competition problem. This means that it is possible to assign every class with one of the available times as well as with one of the available rooms (unless the class requires no room assignment) without breaking any of the hard constraints. Moreover, all courses have enough space so that it is always possible to enroll all students that request the course such that all class limits are respected. There may, however, be student conflicts, violated soft distribution constraints as well as time and room penalizations.

Each solution is described using the XML data format as shown in Figure 8. Each `solution` must be identified by the `name` of the instance (matching the `problem name` from Figure 1). There are also a few solution attributes that can be used to analyze the solutions. These are the solver `runtime` in seconds, the number of CPU `cores` that the solver employs (optional, defaults to 1), the name of the solver `technique` or algorithm, the name of the competitor or his/her team (called `author`), the name and the `country` of the `institution` of the competitor. Note that neither the runtime nor the number of cores will play any role in the decision about the winner or the finalists.

Each solution consists of a set of `class`es, each with a given `id`, a `start` time slot, `days` of the week, `weeks` of the semester, a `room` and a list of `id`s for all `students` who are expected to attend the class. Each student must be enrolled in all the courses following the rules mentioned earlier (see Sections 3.3 and 3.4). All classes of the problem must be present and contain a time and a room assignment from their domain.

```
<?xml version="1.0" encoding="UTF-8"?>
<solution name="unique-instance-name"
        runtime="12.3" cores="4" technique="Local Search"
        author="Pavel Novak" institution="Masaryk University" country="Czech Republic">
  <class id="1" days="1010100" start="90" weeks="1111111111111" room="1">
    <student id="1"/>
    <student id="3"/>
  </class>
  <class id="2" days="0100000" start="86" weeks="0101010101010" room="4">
    <student id="2"/>
    <student id="4"/>
  </class>
  <class id="3" days="0010000" start="108" weeks="0100000000000">
    <student id="1"/>
  </class>
</solution>
```

Fig. 8 XML specification of the solution

It is also possible to encapsulate the solution within the problem instance like it is shown in Figure 9. In this case, the solution validation can run against the described problem instead of the problem of the matching name from the competition instance database. This could be useful for debugging certain aspects of the problem.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem name="unique-instance-name" nrDays="7" nrWeeks="13" slotsPerDay="288">
  ...
  <solution>
    <class id="1" days="1010100" start="90" weeks="1111111111111" room="1">
      <student id="1"/>
      <student id="3"/>
    </class>
    ...
  </solution>
</problem>
```

Fig. 9 XML specification of the solution for a modified problem

Optimization The problem has several `optimization` criteria, see Figure 10. The selection of a time for each class is associated with a penalty. The sum of these penalties for all classes represents the `time` penalization criterion. The selection of a room for each class can also be related to some penalty, and their summarization represents the `room` penalization criterion. Penalties for all distribution constraints are as well summarized for each solution and represent the `distribution` constraint criterion. Last but not least, it is important to take into account students who cannot attend some of their enrolled courses. The `student` conflict minimization criterion counts for all students all pairs of classes which the student cannot attend because the two classes overlap in time or are close to each other in rooms that are too far apart (same condition as in the `SameAttendees` distribution constraint described in Section 3.5).

```
<optimization time="2" room="1" distribution="1" student="2"/>
```

Fig. 10 XML specification of weights for the optimization criteria

The importance of each criterion differs based on the institution. To handle that, each criterion is associated in the problem definition with its `weight`. The weighted sum of all criteria is to be minimized. While the possible minimum corresponds to zero, it is not typically achieved because it is not possible to handle all components perfectly without any penalization.

4 Characteristics of data sets

This section discusses the most typical differences among data sets which will be present in the competition as well as simplifications of the real-life data that have been introduced in the competition problem. As already mentioned, these changes were made to remove some of the difficulties of the problem formulation while retaining the computational complexity of the problem.

4.1 Differences among data sets

Size of the problem The size of the problem is the first significant characteristic of the data sets. Problem instances that consider only one school or faculty may involve about 500 classes, 2,000 students, and 50 rooms. Other problems that represent timetabling for a large part of a university may consists of up to 2,500 classes, 32,000 students, or 200 rooms.

Room utilization For some problem instances, it is not the size of the problem as such, but the high room utilization that makes the problem difficult to solve. In some of these instances, it is not the overall utilization, but there are clusters of rooms that are in high demand. Large classrooms are a typical example of rooms with high utilization. In other problems, utilization may not be the critical part of the problem, and the optimization component is more emphasized.

Student course demands The real-life data may have the student course demands collected from various sources. Some problems consider pre-enrollments of students to courses or last year's student course enrollments. These can be very diverse and may introduce a high violation of student requests as there can be a few students found for pretty much any conceivable combination of two courses.

On the other hand, student course requests can be based on curricula requirements which are typically easier to satisfy. There are large groups of students taking the same or a very similar set of courses as they are following the same program of study. Curricula at such schools may be rather standard with some compulsory and optional courses. However, other schools may construct combinations of curricula for students which may be harder to satisfy for a few atypical combinations.

As an example, we consider a school of education where students that are training to be high school teachers always have two majors representing two different subjects such as Mathematics and Chemistry. Typical combinations, such as Mathematics–Physics or English–History may involve many students. Some other, less popular combinations like Music–Chemistry or Art–Mathematics, have only a few students. This results in a very diverse set of student course demands.

Lastly, there may be problem instances that have no student course demands at all. In these cases, the required distribution of classes is usually achieved with the distribution constraints, typically using the `SameAttendees` or the `NotOverlap` constraints.

Course structure Many universities have a very simple structure of courses. A course typically consists of one lecture. Or it is a set of seminars where each student must be sectioned to a particular seminar. The most complex cases introduce a course with one lecture to be attended by all students and several seminars to which the students are to be sectioned.

Some other universities have a complex course structure for some courses as introduced in Section 3.3. For example, there can be an introductory Biology course that is offered to most freshmen students at the university. Such a course may have a couple of large lectures that are needed to cover the student demand. Besides a lecture, each student may need to take a laboratory and a seminar which are typically taught for much smaller groups of students. The parent-child relations may be used to link students of particular pairs of a laboratory and a seminar together so that the same instructor can teach them. The course can also be offered in multiple configurations.

Times Each institution may use time in different patterns. While some problems have the time of the day split into a nice set of non-overlapping teaching hours, other institutions may allow a class to start at various times (e.g., at any quarter of an hour). Some classes can meet only once a week; others can have multiple meetings, following the same start time, same length and same room schema which allows us to model such meetings as a single class. Similarly, different institutions may make different use of the weeks of the semester. While most classes are typically offered during all the weeks of the semester, there can be classes offered only during the first or the last half of the semester, during even or odd weeks, or even just once during a particular week.

For example, European institutions tend to have their classes once a week for a longer time period such as 2 hours. On the other hand, institutions in the United States typically have classes several times a week such as Monday–Wednesday–Friday, Tuesday–Thursday or Monday–Friday following the above described pattern.

Distance learning may offer an example of a rather specific use of the time that may be seen in the problem instances. Here, the students only come to school once a week or once every two weeks (e.g., every Friday or Saturday), and have all their classes during that day. Each distance learning course may have only two or three meetings during the semester which occur on different weeks. Each of these meetings is usually modeled as a single class and they are linked with additional distribution constraints (e.g., required `DifferentWeeks`).

Travel times A classical problem of one school involves all the rooms situated in one building. More complex problems may consider several buildings or a campus where classes need to consider non-negligible travel times. Some exceptional schools such as a school of sports studies may involve many sports facilities spread over the city and consideration of travel times introduces a crucial part of the problem solution.

Distribution constraints Problem instances may differ by the importance and the amount of distribution constraints. Some schools may use many distribution constraints, while others do not rely on them so heavily. For example, some problems rely on the `WorkDay`, the `MaxBlock` and the `MaxDayLoad` constraints to provide good schedules for instructors, while other problems may not contain these three particular constraints at all.

4.2 Transformations of real-life data into the competition problems

In order to make the problem instances easier to model and to work with, a number of less important features have been removed or simplified. The aim was to simplify the problem formulation while maintaining the computational complexity of the problems. The most important changes are summarized below.

Rooms In reality, the computation of what rooms are available for a class is quite complex. For instance, only rooms of a particular room type can be considered; they must have the necessary equipment, be of a particular building or location at the campus, etc. Also, only certain departments may be allowed to use a particular room, and the room must be big enough for the class to fit in. Some classes may have a room ratio defined that further refines the relation between the minimal size of the room and the class limit. For example, it is possible for the room capacity to be smaller than the class limit when it is expected that only half of the students would actually show up for the class. Similarly, there can be preferences set on the type of the room, its location, equipment, etc. All these characteristics have been combined together to create a list of rooms that are acceptable for the class and to compute their individual penalties.

The penalization also includes additional optimization criteria which may be present in the original problems. For instance, there is a penalization for rooms with bigger size than what the class needs, this penalization is proportional to the size of excessive space. Also, some rooms may generally be discouraged which is also implemented by their high penalization, while the original problem tries to minimize the overall use of such rooms. An additional optimization criterion, though with a very small weight, may be applied to optimize the space left in the rooms. For instance, gaps that are shorter than one teaching hour are penalized.

In some of the original problems, the sharing of a room between two or more departments can be more refined, allowing a department to have exclusive use of the room during certain times of the week. In the competition problem, all classes of the departments that share a room can use the room during the time when the room is available, given that the other class requirements are met (class limit, required room type, equipment, etc.).

Travel times Many problem formulations involve distances, which could be defined on both buildings and rooms as GPS coordinates, or between pairs of these locations as travel times that are typically expressed in minutes. In order to anonymize the data sets, all distances have been converted into travel times, which also makes it easier for the conflict checking (for both the student conflicts and the `SameAttendees` constraint) as the travel time can be easily compared with the gap between two consecutive classes. In the original problem, there is an additional optimization criterion checking for instructor distances between two rooms for classes that are being taught immediately one after the other, considering the distance in meters. This has been replaced with the `SameAttendees` constraint in the competition problems.

Times Time penalization is also adjusted. For example, considered penalties may combine the individual time preferences set on the class with the default time penalties specified in each problem. For instance, early morning and late evening times are usually discouraged with a small penalty.

Distribution constraints A large part of the transformations relates to the distribution constraints. A few rarely used constraints have been removed from the formulation completely, others have been simplified or transformed into other existing constraints. For instance, we have the `Back-to-Back` constraint in the original problems making sure that all given classes are placed on the same day, one after the other. This constraint has been translated using the `SameDays` and `WorkDay` constraints, modeling the same thing (when fully satisfied) with more straightforward penalization. Similarly, the original problem allows for two classes to meet together in a room when the room is big enough. In this case, both the room and the assigned instructor or instructors are fine with the overlap. In the competition problem this has been translated as follows:

– When two (or more) classes are required to meet together, the first class can only be put in a room that fits all classes of the constraint, and the remaining classes have no room and no instructor (if the same instructor is assigned). All these classes are tight together with required `SameStart`, `SameDays`, and `SameWeeks` constraints.
– When two (or more) classes can meet together, but the meeting is not required, the room sharing component gets either ignored (if it is possible to assign the classes in different rooms) or the same transformation is done as when the constraint is required.

The `NextDay` constraint offers an example of a constraint that got simplified. In the original problem, it means that the classes of the constraint must be placed on consecutive days. However, based on the configuration, the constraint may or may not ignore weeks and weekends (Friday can be followed by Monday on the next week). Consideration of classes with multiple days of the week is also complicated. In the problem instances, this constraint has been replaced by the `Precedence` and the `DifferentDays` constraint.

Students In the original problem, reservations can be used to restrict certain students to some of the classes of the course. These can be based on the various student properties such as a program of study, an academic year, a major or a group affiliation. While these reservations create a considerable complication for student scheduling, the reservations do not get used often enough to justify the complication of the problem formulation.

In some problems, certain student conflicts may be prohibited. This means that two classes are prohibited to conflict when the number of students attending both classes is too great, typically expressed as a percentage of the size of the smaller class.

In some problems, there is an additional criterion that tries to keep students of the same curriculum or student group together. This is a very recent addition to the problem formulation, so most of the original problems do not use it yet.

Instructors Instructors are modeled using the `SameAttendees` distribution constraints, i.e. classes of one instructor cannot overlap in time or be one after the other in rooms that are too far away (there are fewer time slots in between than the travel time).

Other aspects A few additional constraints and criteria may be present in the original problems. These are usually very specific to a particular institution and would require additional information about the problem to be included in the data. Such aspects have also been removed from the competition instances, or where possible, replaced by the existing constructs.

For example, various student conflicts may have different weights in some problems. These could be conflicts between courses that offer no or only a small number of options. Or, there could be a higher weight for students of a curriculum for which at least one of the two courses is mandatory, and the other course is either mandatory or an elective. Similarly, conflicts between two electives or conflicts with an optional course can be considered less important. For the competition instances, we have decided it is sufficient that the core curriculum courses typically have a high number of students in common, resulting in higher student conflict penalties when there is not enough space offered in non-conflicting times.

5 Organization of the competition

The competition will have its website maintained at Masaryk University and it will be available from `http://www.itc2019.org`. The website will contain the description of the competition, the rules, and the data instances that have been published so far. It will also include a web service that will validate the solutions for the competition problems and will allow for the valid solutions to be uploaded to the website. The best uploaded solution for each competitor and instance will be used for the ranking of the competitors, and it will be published on the competition website once the competition is concluded.

Three groups of data sets will be published during the competition. While the order of the competitors will be based on all the published data instances, the instances released later in the competition will have a much higher weight in the final ranking. Some anonymized information, like the values of the best-known solutions that have been gathered by the validator service so far, may be published at certain points of the competition. The solution validator is based on the UniTime solver [29]. Additional information with some important characteristics of the data will be published together with the data sets. Our goal is to maintain the website with results after the competition similarly to earlier competitions on curriculum-based [5] or high-school [21] timetabling.

We expect the following timeline.

1. The competition is announced at PATAT 2018 in *August 2018.*
2. The first group of the data sets is published on *November 15, 2018.*
3. The second group of the data sets is published on *September 18, 2019.*
4. The third group of the data sets is published on *November 8, 2019.*
5. Competition teams submit their final results for all the competition instances by *November 18, 2019.* All teams also submit a short report summarizing their implementation (3 – 5 pages in the PATAT typesetting style).
6. Finalists are published and winners are informed by *January 15, 2020.*
7. Competitors, as well as other people, are invited to submit their paper to a track at PATAT 2020 related to the competition.
8. Winners are announced at PATAT 2020.
9. The finalists are encouraged to submit their PATAT 2020 paper to the journal post-conference publication.

As we have mentioned, the website will also contain rules of the competition. The finalists and the winner will be determined according to the competition rules.

6 Conclusion

The International Timetabling Competition 2019 is aimed at solving common university course timetabling problems from practice. A wide set of features is considered. The key novelty lies in the combination of student sectioning together with standard time and room assignment of events in courses. As a part of the competition, we would like to collect an interesting set of data which will enrich further research.

While the new competition is based on ideas of the previous competitions, some important aspects are different. In particular, we do not constrain the time needed to compute solutions, the number of CPU cores or machines that the solver can use. At the same time, we do not expect to run solvers of the competitors on our hardware. We may ask to see the source code of solvers for the finalists.

Even though it may be tricky to allow arbitrary computational time, we believe that this format has important advantages for advancements in the future research and practice of timetabling. We will summarize them in the final paragraphs of the paper.

When different solvers are compared, the solution quality is always instrumental. Even the current websites are comparing the best solutions by merely comparing their quality [21, 5]. While the computational time may also be important, its comparison is very tricky, especially after the competition. We will ask the competitors to include the computational time in the solution, but it will not play any role in the decision about the winner or the finalists. The biggest downside of the time limits (together with the maximal number of executions) is undoubtedly their incorrect interpretation or reproducibility of the results, especially by other researchers.

An important aspect of modern solvers is parallelism. While the previous competitions were not able to handle this feature, it is possible to consider it now. This can support the development of new advanced solvers with parallel features.

Last but not least, commercial solvers that are commonly used in practice were not eligible in the earlier competitions. By allowing commercial solvers, we would like to encourage a larger community to participate in the competition. It will be certainly interesting to compare the results of the commercial solvers with the other ones.

Acknowledgements We would like to thank for the sponsorship of the PATAT conference and the EURO working group on Automated Timetabling (EWG PATAT). We also acknowledge Masaryk University which provided resources for maintenance of the competition website. Our great thanks go to all institutions which provided the data; this is essential for organization of the competition. The institutions will be published with the publication of the data.

References

1. EWG PATAT, EURO working group on automated timetabling. `https://www.euro-online.org/web/ewg/14/ewg-patat-euro-working-group-on-automated-timetabling`
2. PATAT conferences. `http://patatconference.org/`
3. Bettinelli, A., Cacchiani, V., Roberti, R., Toth, P.: An overview of curriculum-based course timetabling. TOP **23**(2), 313–349 (2015)
4. Bonutti, A., De Cesco, F., Di Gaspero, L., Schaerf, A.: Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Annals of Operations Research **194**(1), 59–70 (2012)
5. Bonutti, A., Gaspero, L.D., Schaerf, A.: Curriculum-based course timetabling. `http://tabu.diegm.uniud.it/ctt/`
6. Burke, E., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research **140**(2), 266–280 (2002)
7. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., McCollum, B., Ochoa, G., Parkes, A.J., Petrovic, S.: The cross-domain heuristic search challenge – an international research competition. In: C.A.C. Coello (ed.) Learning and Intelligent Optimization, pp. 631–634. Springer (2011)
8. Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: The second international nurse rostering competition. Annals of Operations Research (2018). First online

9. Gaspero, L.D., McCollum, B., Schaerf, A.: The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Tech. Rep. QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, Queen's University, Belfast (2007)
10. Haspeslagh, S., De Causmaecker, P., Schaerf, A., Stølevik, M.: The first international nurse rostering competition 2010. Annals of Operations Research **218**(1), 221–236 (2014)
11. Kingston, J.H.: Educational timetabling. In: A.S. Uyar, E. Ozcan, N. Urquhart (eds.) Automated Scheduling and Planning: From Theory to Practice, pp. 91–108. Springer Berlin Heidelberg (2013)
12. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum **30**(1), 167–190 (2008)
13. Lewis, R., Paechter, B., McCollum, B.: Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance A2007-3, Cardiff Business School, Cardiff University (2007)
14. McCollum, B., McMullan, P., Burke, E.K., Parkes, A.J., Qu, R.: The second International Timetabling Competition: Examination timetabling track. Tech. Rep. QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, Queen's University, Belfast (2007)
15. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Gaspero, L.D., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing **22**(1), 120–130 (2010)
16. Müller, T.: ITC2007 solver description: a hybrid approach. Annals of Operations Research **172**(1), 429 (2009)
17. Müller, T., Rudová, H.: Real-life curriculum-based timetabling with elective courses and course sections. Annals of Operations Research **239**(1), 153–170 (2016)
18. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A., Petrovic, S., Burke, E.: HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. In: J.K. Hao, M. Middendorf (eds.) European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2012), *Lecture Notes in Computer Science*, vol. 7245, pp. 136–147. Springer (2012)
19. Paechter, B., Gambardella, L.M., Rossi-Doria, O.: International timetabling competition 2002 (2002). `http://sferics.idsia.ch/Files/ttcomp2002/`
20. Pillay, N.: A survey of school timetabling research. Annals of Operations Research **218**(1), 261–293 (2014)
21. Post, G.: Benchmarking project for (high) school timetabling. `https://www.utwente.nl/ctit/hstt/`
22. Post, G., Ahmadi, S., Daskalaki, S., Kingston, J.H., Kyngas, J., Nurmi, C., Ranson, D.: An XML format for benchmarks in high school timetabling. Annals of Operations Research **194**(1), 385–397 (2012)
23. Post, G., Di Gaspero, L., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. In: Practice and Theory of Automated Timetabling 2012 Proceedings, pp. 479–484 (2012)
24. Post, G., Kingston, J.H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., Schaerf, A.: XHSTT: an XML archive for high school timetabling problems in different countries. Annals of Operations Research **218**(1), 295–301 (2014)
25. Rudová, H., Müller, T., Murray, K.: Complex university course timetabling. Journal of Scheduling **14**(2), 187–207 (2011)
26. Rudová, H., Murray, K.: University course timetabling with soft constraints. In: E. Burke, P. De Causmaecker (eds.) Practice and Theory of Automated Timetabling IV, pp. 310–328. Springer Berlin Heidelberg (2003)
27. Schaerf, A.: A survey of automated timetabling. Artificial Intelligence Review **13**(2), 87–127 (1999)
28. Schmidt, G., Ströhlein, T.: Timetable construction – an annotated bibliography. Computer Journal **23**(4), 307–316 (1980)
29. UniTime: University timetabling – Comprehensive academic scheduling solutions. `http://unitime.org`
30. de Werra, D.: An introduction to timetabling. European Journal of Operational Research **19**(2), 151–162 (1985)

Appendix: Sample XML format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE problem PUBLIC
  "-//Course Timetabling Competition//DTD Problem Format/EN"
  "http://www.unitime.org/interface/competition-format.dtd">
<!--
  Competition Problem Format
  It includes a unique name of the instance, number of days of the week, number
  of weeks of the semester, and the number of time slots during a days.
  In this example, each time slot takes 5 minutes and they go from
  midnight to midnight. This is typical for all the competition instances,
  however, the problem format allows for variation.
-->
<problem name="unique-instance-name" nrDays="7" nrWeeks="13" slotsPerDay="288">
  <!--
    Optimizatio Weights: These are the weights on the total penalty of assigned
    times, assigned rooms, violated soft distribution constraints,
    and the number of student conflicts.
   -->
  <optimization time="2" room="1" distribution="1" student="2"/>

  <!--
    List of Rooms: Each room has a unique id, capacity, availability and
    travel times.
  -->
  <rooms>
    <room id="1" capacity="50"/>
    <room id="2" capacity="100">
      <!--
        Travel time to another room is in the number of time slots
        it takes to travel from this room to the other room. All distances are
        symmetrical, and only non-zero distances are present.
      -->
      <travel room="1" value="2"/>
    </room>
    <room id="3" capacity="80">
      <travel room="2" value="3"/>
      <!-- Availability: list of times when the room is not available -->
      <!-- Not available on Mondays and Tuesdays, 8:30 - 10:30, all weeks -->
      <unavailable days="1100000" start="102" length="24" weeks="1111111111111"/>
      <!-- Not available on Fridays, 12:00 - 24:00, odd weeks only -->
      <unavailable days="0001000" start="144" length="144" weeks="1010101010101"/>
    </room>
    <!-- ... -->
  </rooms>

  <!--
    List of Classes that are to be timetabled, including their course structure.
    Each course has one or more configurations, each configuration has one or
    more scheduling subparts, and each subpart has one or more classes.
    All ids are sequentially generated and unique (for each type) within the XML
    file. A class may have a parent id if there is a parent-child relation
    defined.
  -->
  <courses>
    <course id="1">
      <config id="1">
        <subpart id="1">
          <!--
            Each class has a limit and a list of availabile rooms and times,
            each with a penatly. Only rooms that are big enough and meet all
            the requirements (room type, required equipment, etc.) are listed.
            Each class needs to be assigned to one room and one time from these.
          -->
          <class id="1" limit="20">
            <room id="1" penalty="0"/>
            <room id="2" penalty="10"/>
            <!--
              Each time has days of the week (as bit string, starting on Monday),
              time of the day (start slot and length), and weeks of the semester
              (also a bit string: week 1, week 2, ... ).
            -->
            <!-- MWF 7:30 - 8:20 all weeks -->
            <time days="1010100" start="90" length="10" weeks="111111111111"
              penalty="0"/>
            <!-- TTh 8:00 - 9:15 all weeks -->
            <time days="0101000" start="96" length="15" weeks="111111111111"
              penalty="2"/>
          </class>
          <!--
            The second class of the same course, configuration, and subpart.
            Alternative to class 1.
          -->
          <class id="2" limit="20">
            <room id="4" penalty="0"/>
            <!-- Mon 7:10 - 8:40 even weeks -->
            <time days="1000000" start="86" length="18" weeks="010101010101"
              penalty="0"/>
            <!-- Tue 7:10 - 8:40 even weeks -->
            <time days="0100000" start="86" length="18" weeks="010101010101"
              penalty="0"/>
          </class>
        </subpart>
        <subpart id="2">
          <!--
            Child of class 1: a student taking class 3 must also take class 1.
            Classes may have no rooms, these are only to be assingned with a time.
          -->
          <class id="3" parent="1" room="false">
            <!-- Fri 8:00 - 9:50 first week -->
            <time days="0000100" start="96" length="22" weeks="100000000000"
              penalty="2"/>
            <!-- Wed 9:00 - 10:50 second week -->
            <time days="0010000" start="108" length="22" weeks="010000000000"
              penalty="0"/>
          </class>
          <!-- ... -->
        </subpart>
      </config>
    </course>
    <!-- ... -->
  </courses>

  <!--
    List of Distribution Constraints: a distribution constraint can be hard
    (required=true) or soft (has a penalty). For most soft constraints,
    a penalty is incurred for each pair of classes that violates the constraint.
  -->
  <distributions>
    <!-- Classes 1 and 2 cannot overlap in time -->
    <distribution type="NotOverlap" required="true">
      <class id="1"/>
      <class id="2"/>
    </distribution>
    <!-- Class 1 should be before class 3, class 3 before class 5 -->
    <distribution type="Precedence" penalty="2">
      <class id="1"/>
      <class id="3"/>
      <class id="5"/>
    </distribution>
    <!--
      Instructors are modeled using the SameAttendees constraint: Classes cannot
      overlap in time or be one after the other in rooms that are too far away
      (there are fewer time slots in between than the travel time).
    -->
```

```
    <distribution type="SameAttendees" required="true">
      <class id="1"/>
      <class id="12"/>
    </distribution>
    <!-- Classes cannot span more than two days of the week -->
    <distribution type="MaxDays(2)" required="true">
      <class id="5"/>
      <class id="8"/>
      <class id="15"/>
    </distribution>
    <!-- ... -->
  </distributions>

  <!--
    Student Course Demands: Each student needs a class of each subpart of one
    configuration of a course. Parent-child relation between classes must be
    used when defined.
  -->
  <students>
    <!-- Each student has a list of courses he/she needs. -->
    <student id="1">
      <course id="1"/>
      <course id="5"/>
    </student>
    <student id="2">
      <course id="1"/>
      <course id="3"/>
      <course id="4"/>
    </student>
    <!-- ... -->
  </students>

  <!--
    Solution: A solution contains a list of classes with their assignments.
    There are also a few solution attributes that can be used to identify the
    solution. These are:
      - problem name (only needed when the XML does not contain the problem,
          i.e., solution is the root element)
      - solver runtime in seconds,
      - number of CPU cores that the solver employs (optional, defaults to 1),
      - name of the solver technique/algorithm,
      - name of the competitor or his/her team,
      - and the name and the country of the institution of the competitor
  -->
  <solution
      name="unique-instance-name"
      runtime="12.3" cores="4" technique="Local Search"
      author="Pavel Novak" institution="Masaryk University" country="Czech Republic">
    <!--
      Each class has an assigned time and (when there are rooms) an assigned
      room. Both must be from the domain of the class. There is also a list of
      students enrolled in the class.
    -->
    <class id="1" days="1010100" start="90" weeks="1111111111111" room="1">
      <student id="1"/>
      <student id="3"/>
    </class>
    <class id="2" days="0100000" start="86" weeks="0101010101011" room="4">
      <student id="2"/>
      <student id="4"/>
    </class>
    <class id="3" days="0010000" start="108" weeks="0100000000000">
      <student id="1"/>
    </class>
    <!-- ... -->
  </solution>
</problem>
```