Data Format

In this document, we describe each component of the problem as it is defined in the XML format. A sample XML is included at the bottom of this page. 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 requires 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. A zero penalty represents complete satisfaction. Penalties associated with each component of the problem are combined in a weighted sum reflecting the prioritization of each component in the overall solution. The best overall solution would again have zero penalty, but such a solution does not typically exist.

As an example, each class has a list of available time and room assignments given in the specific 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 the paper (/papers/itc2019-patat2018.pdf)).

Contents

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 in the root element of the XML file as shown in Figure 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE solution PUBLIC
        "-//ITC 2019//DTD Problem Format/EN"
        "http://www.itc2019.org/competition-format.dtd (http://www.itc2019.org/competition-
format.dtd)">
<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 some or all weeks of the semester. 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 have a duration **length** given as a 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, 4th, . . . , and 12th weeks 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 Distribution Constraints (format#distributions)).

Rooms

Each **room** is specified by its **id** and **capacity**. A room may not be available at certain times, which are defined by **unavailable** elements using the **days** of the week, the **start** time slot, and a **length** in slots for a set of **weeks** during the semester. Non-zero travel times from other rooms are specified by the **travel** elements, which express the number of time slots it takes to get from one room to another. 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" room="false"/>
```

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

Courses

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

Course					Introduction to Mech. Eng. ME 263				
Configuration		Lec-Rec					Lec-Rec-Lab		
Subpart	1_Lecture				3_Lecture				
Parent	2_Recitation					4_Recitation			
Child							5_Laboratory		
Class	Lec1	Lec2			Lec3				
Parent	Rec1	Rec2	Rec3	Rec4		Rec5		Rec6	
Child							Lab1		Lab2
					Lec4				
						Rec7		Rec8	
							Lab3		Lab4

```
<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

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**s such that each student attends (some) classes in one configuration only. Each configuration consists of one or more subparts with their unique **id**s (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 in 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 who 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 Rec4.

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 given in the problem (see their description in Section Distribution Constraints (format#distributions)) since each institution may use a different set of constraints. These typically include:

- classes that are in a parent-child relation cannot overlap in time (`SameAttendees` (`format#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` (`format#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 (format#Precedence)` constraint is placed between any pair of a lecture and a seminar that a student can take).

Each class has a defined set of possible times when it can meet. Each eligible time has a specified **penalty** which must be included in the overall time penalization when the time is selected for the class (see Section Solution (format#solution)). Valid time specifications were described in Section Times (format#times).

Each class also has a defined a set of possible rooms where it can meet (if a room is needed). Each eligible room has a specified penalty to be included in the overall time penalization when selected (see Section Solution (format#solution)). Valid room specifications were given in Section Rooms (format#times).

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.

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

Fig. 6 XML specification of students and their courses

A student needs to be sectioned into 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 Courses (format#courses)). 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.

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.

```
<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.

Constraint	Complementary	Time	Days	Weeks	Room	Pairs
`SameStart (format#SameStart)`		✔	✖	✖	✖	✔
`SameTime (format#SameTime)`	`DifferentTime (format#DifferentTime)`	✔	✖	✖	✖	✔
`SameDays (format#SameDays)`	`DifferentDays (format#DifferentDays)`	✖	✔	✖	✖	✔
`SameWeeks (format#SameWeeks)`	`DifferentWeeks (format#DifferentWeeks)`	✖	✖	✔	✖	✔
`SameRoom (format#SameRoom)`	`DifferentRoom (format#DifferentRoom)`	✖	✖	✖	✔	✔
`Overlap (format#Overlap)`	`NotOverlap (format#NotOverlap)`	✔	✔	✔	✖	✔
`SameAttendees (format#SameAttendees)`		✔	✔	✔	✔	✔
`Precedence (format#Precedence)`		✔	✔	✔	✖	✔
`WorkDay(S) (format#WorkDay(S))`		✔	✔	✔	✖	✔
`MinGap(G) (format#MinGap(G))`		✔	✔	✔	✖	✔
`MaxDays(D) (format#MaxDays(D))`		✖	✔	✖	✖	days over **D**
`MaxDayLoad(S) (format#MaxDayLoad(S))`		✔	✔	✔	✖	slots over **S**
`MaxBreaks(R,S) (format#MaxBreaks(R,S))`		✔	✔	✔	✖	breaks over **R**
`MaxBlock(M,S) (format#MaxBlock(M,S))`		✔	✔	✔	✖	blocks over **M**

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

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`

Short Description: All classes in this constraint must start at the same time of day.

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`

Short Description: All classes in this constraint must be offered within the same times of day as those required by the longest class in the set, i.e., the times of day they meet must be completely overlapped by the longest class in the constraint.

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` (`format#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

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

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

`DifferentTime`

Short Description: The times of day during which all classes in this constraint meet can not overlap.

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

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

for any two classes C_i and C_j from the constraint.

`SameDays`

Short Description: All classes in this constraint must be offered on the same days of the week or, if a class meets fewer days, it must meet on a subset of the days used by the class with the largest number of days.

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

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

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

`DifferentDays`

Short Description: All classes in this constraint can not meet on any of the same days of the week.

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

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

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

`SameWeeks`

Short Description: All classes in this constraint must be offered during the same weeks of the term or, if a class meets fewer weeks, it must meet during a subset of the weeks used by the class meeting for the largest number of weeks.

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

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

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

`DifferentWeeks`

Short Description: The weeks of the term during which any classes in this constraint meet can not overlap.

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

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

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

`Overlap`

Short Description: Any two classes in this constraint must have some overlap in time.

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

$(C_j.\textbf{start} < C_i.\textbf{end}) \wedge (C_i.\textbf{start} < C_j.\textbf{end}) \wedge ((C_i.\textbf{days}$ and $C_j.\textbf{days}) \neq 0) \wedge ((C_i.\textbf{weeks}$ and $C_j.\textbf{weeks}) \neq 0)$

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`

Short Description: No two classes in this constraint can overlap in time.

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

$(C_i.\textbf{end} \leq C_j.\textbf{start}) \vee (C_j.\textbf{end} \leq C_i.\textbf{start}) \vee ((C_i.\textbf{days} \text{ and } C_j.\textbf{days}) = 0) \vee ((C_i.\textbf{weeks} \text{ and } C_j.\textbf{weeks}) = 0)$

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`

Short Description: All classes in this constraint must be assigned to the same room.

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

`DifferentRoom`

Short Description: All classes in this constraint must be assigned to different rooms.

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

`SameAttendees`

Short Description: All classes in this constraint must meet at times and locations such that someone who must attend, e.g., an instructor, is reasonably able to attend all classes. This means that no two classes can overlap in time or follow one another without sufficient time to travel between the two class locations.

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

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

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

`Precedence`

Short Description: This constraint establishes an ordered listing which requires that the first meeting of each of the listed classes occurs in its entirety prior to the first meeting of all subsequently listed classes.

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

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

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)

Short Description: This constraint prevents or penalizes the occurrence of class pairs among the listed classes where the time between the start of one class and the end of another which occur on the same day is greater than S time slots.

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

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

for any two classes C_i and C_j from the constraint.

MinGap(G)

Short Description: The times assigned to any two classes in this constraint that are placed on the same day must allow for at least G slots between the end of the earlier class and the start of the later class.

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

$$((C_i.\textbf{days} \text{ and } C_j.\textbf{days}) = 0) \vee ((C_i.\textbf{weeks} \text{ and } C_j.\textbf{weeks}) = 0) \vee (C_i.\textbf{end} + G \leq C_j.\textbf{start}) \vee (C_j.\textbf{end} + G \leq C_i.\textbf{start})$$

for any two classes C_i and C_j from the constraint.

MaxDays(D)

Short Description: The classes in this constraint should not be assigned to more than D different weekdays, regardless of whether this occurs during the same week of a term.

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, ..., C_n is not greater than D,

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

where `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)

Short Description: This constraint limits the total amount of time assigned to the set of classes listed in the constraint to no more than S time slots per day over the entire term.

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 $w \in \{0, 1, ..., \textbf{nrWeeks} - 1\}$ of the semester and each day of the week $d \in \{0, 1, ..., \textbf{nrDays} - 1\}$, the total number of slots assigned to classes C that overlap with the selected day d and week w is not more than S,

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

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

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

$$(\textbf{penalty} \times \Sigma_{w,d} \texttt{max}(\texttt{DayLoad}(d,w) - S, 0)) \;/\; \textbf{nrWeeks}$$

MaxBreaks(R,S)

Short Description: This constraint limits the number of breaks between classes during a day which exceed S time slots to no more than R per day.

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 $w \in \{0, 1, ..., \textbf{nrWeeks} - 1\}$ of the semester and each day of the week $d \in \{0, 1, ..., \textbf{nrDays} - 1\}$, the number of blocks is not greater than $R + 1$,

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

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.

$$(B_a.\textbf{end} + S \geq B_b.\textbf{start}) \wedge (B_b.\textbf{end} + S \geq B_a.\textbf{start}) \Rightarrow (B.\textbf{start} = \texttt{min}(B_a.\textbf{start}, B_b.\textbf{start})) \wedge (B.\textbf{end} = \texttt{max}(B_a.\textbf{end}, B_b.\textbf{end}))$$

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 (format#MaxDayLoad) constraint).

MaxBlock(M,S)

Short Description: This constraint limits the amount of time (measured as *M* slots) that a set of classes may be consecutively scheduled to which are each separated by no more than *S* slots.

This constraint limits the length of a block of two or more 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 $w \in \{0, 1, ..., \textbf{nrWeeks} - 1\}$ of the semester and each day of the week $d \in \{0, 1, ..., \textbf{nrDays} - 1\}$, the maximal length of a block does not exceed *M* slots

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

Please note that only blocks of two or more classes are considered. In other words, if there is a class longer than *M* slots, it does not break the constraint by itself, but it cannot form a larger block with another class.

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 (format#MaxDayLoad) constraint).

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.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE solution PUBLIC
        "-//ITC 2019//DTD Problem Format/EN"
        "http://www.itc2019.org/competition-format.dtd (http://www.itc2019.org/competition-
format.dtd)">
<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. 9 XML specification of the solution for a modified problem

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 ids for all **student**s who are expected to attend the class. Each student must be enrolled in all the courses following the rules mentioned earlier (see Sections Courses (format#courses) and Students (format#students)). All classes of the problem must be present and contain a time and a room assignment from their domain.

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 summation represents the **room** penalization criterion. Penalties for all distribution constraints are also summed 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 the courses they are enrolled in. 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 there is insufficient time between classes assigned to rooms that are too far apart (same condition as in the `SameAttendees` (`format#SameAttendees`) distribution constraint described in Section Distribution Constraints (format#distributions)).

```
<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 this, each criterion is associated with a **weight** in the problem definition. 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.

Sample XML Problem

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE problem PUBLIC
        "-//ITC 2019//DTD Problem Format/EN"
        "http://www.itc2019.org/competition-format.dtd (http://www.itc2019.org/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="1111111111111"
                                                        penalty="0"/>
                                                <!-- TTh 8:00 - 9:15 all weeks -->
                                                <time days="0101000" start="96" length="15"
weeks="1111111111111"
                                                        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="0101010101010"
                                                        penalty="0"/>
                                                <!-- Tue 7:10 - 8:40 even weeks -->
                                                <time days="0100000" start="86" length="18"
weeks="0101010101010"
                                                        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="1000000000000"
                                                        penalty="2"/>
                                                <!-- Wed 9:00 - 10:50 second week -->
                                                <time days="0010000" start="108" length="22"
weeks="0100000000000"
                                                        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>
</problem>
```

Sample XML Solution

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE solution PUBLIC
	"-//ITC 2019//DTD Problem Format/EN"
	"http://www.itc2019.org/competition-format.dtd (http://www.itc2019.org/competition-
format.dtd)">
<!--
	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>
```

ITC 2019 Discussion Forum (/forum)

For any questions or comments, please contact us at organizers@itc2019.org (mailto:organizers@itc2019.org).