A variant of the high-school timetabling problem and a software solution for it based on integer linear programming

Iulian Ober

▶ **To cite this version:**

Iulian Ober. A variant of the high-school timetabling problem and a software solution for it based on integer linear programming. 11th International Conference on Practice and Theory of Auto-mated Timetabling (PATAT-2016), Aug 2016, Udine, Italy. pp.283-294. hal-02943035

HAL Id: hal-02943035
https://hal.science/hal-02943035

Submitted on 18 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A variant of the high-school timetabling problem and a software solution for it based on integer linear programming

Iulian Ober

Abstract The paper presents a practical method for handling a particular class of timetabling problem through integer linear programming and a software tool which implements this approach. The class of problems tackled by the tool is a relatively classical high-school timetabling problem, which however presents distinctions that set it apart, such as the existence of overlapping time slots or the extensive use of hierarchically organized student groups. The results obtained on real-life examples of workloads from our institution are very encouraging, both in terms of quality of the generated timetable and of tool performance and ease of use.

1 Introduction

This paper presents the timetabling problem faced at the University of Toulouse in the Computer Science Department of the Blagnac IUT (University Institute of Technology) and reports on a way to approach this problem through integer linear programming (ILP) and on a software tool which resulted from this approach.

The problem is a particular case of the general High-School Timetabling (HSTT) problem, as characterized for example in [9], but has two features that set it apart. The first feature is that the time slots may be overlapping and are not totally ordered. While it would be possible in principle to re-formulate the problem by sub-dividing the time slots into unique, totally ordered and equal-length sub-slots and assigning several consecutive sub-slots to each lecture, this solution would increase the complexity of the problem specification by the user, as well as the complexity of the ILP model, hindering performance. The second feature is the extensive use of hierarchically organized student groups: as we will show in section 2, the student set is divided into semesters, which are divided into classes that are further divided into smaller groups. Each of these three levels of granularity may constitute the audience of several courses. With the exception of these distinctive features, the problem is a classical one of assigning time-slots and rooms with regard to a curriculum and to a predefined course-group-teacher assignment.

Université de Toulouse, 118 Route de Narbonne
F -31062 Toulouse, France
E-mail: Iulian.Ober@irit.fr

Proceedings of the 11th International Confenference on Practice and Theory of Automated Timetabling (PATAT-2016) – Udine, Italy, August 23–26, 2016

The initial motivation for this work was practical and the author approached it not as a research problem but as a problem of convenient tooling. The main reason for looking for a custom-made solution was the perception that in order to obtain good quality solutions automatically, one fares better if one masters the internals of the timetable generator tool and is able to add specific constraints that fit the needs of the institution. In the process of designing the solution, we nevertheless tried to separate what is relatively generic and applicable to the timetabling problem of many institutions similar to ours from what is specific to the institution. The result is a generic-enough method and an easily extensible tool, described in this paper.

1.1 Related work

There are many research papers describing various techniques for solving variants of the HSTT problem; for a broad literature survey one may consult [8], which focusses on HSTT, and [6,4], which also cover other types of timetabling problems. Many approaches listed there concentrate on heuristic methods, however there are several previous proposals that rely upon integer linear programming [5,2,11,1,13].

A notable recent evolution in the field is the proposal of XHSTT [9], and XML-based format for specifying HSTT problems that has become a de facto standard. A repository of benchmark examples is available in this format, the examples coming from different institutions in various countries [10]. This has stimulated research in the field and opened up the possibility of benchmarking different approaches. Concerning ILP-based approaches, a general resolution method based on this format has been recently proposed in [5].

While the class of problems formulated in this paper is in many ways less general than the class of problems that may be captured in XHSTT format [10], this also opens up the possibility to define a simpler and more efficient ILP model for it, as it can be seen in the following. There is also one respect in which the problem described here is not a sub-class of that of [9], namely the possibility for overlap between time slots, and therefore the existing solutions such as [5] cannot be readily applied, although it is in principle possible to reformulate the problem in terms of totally ordered, non-overlapping, equal-length sub-slots, as mentioned above – at the cost of increasing its complexity.

2 The timetabling problem

The problem tackled in this work is a particular case of the general High-School Timetabling (HSTT) problem, aiming to assign time-slots and rooms with regard to a curriculum and to a predefined course-group-teacher assignment. Being based on a common national curriculum, other IUTs in France face a very similar problem, although sometimes there may be additional constraints due to local specificities. A particular feature of the IUT curriculum is that it is composed of modules that do not usually last for a whole semester and therefore the set of courses changes usually from one week to another, requiring the timetable to be redefined each week. Adding that to a curriculum that is particularly dense (33h of class per student per week on average) and to a high utilization of specialized rooms (amphitheatres, computer rooms, multimedia rooms) which are in limited number, make this problem very

challenging to tackle manually. On the positive side, the problem is made easier by the fact that the course-group-teacher assignment is usually predefined and there are few courses with high durations requiring spreading throughout the week.

The elements to be taken into account in the timetable are as follows:

- *Time slots:* Contrary to the usual HSTT problem, there is not necessarily a list of disjoint time-slots. The reason for this is that the duration may vary from one lecture to another. For example, at the Blagnac IUT, most lectures have a duration of 1:30 hours (in which case they may start at fixed times: 8AM, 9:30AM, 11AM, 14:15PM, 15h45PM, 17h15PM) but there may be lectures with different durations and starting at different times (for example, a 2-hours lecture starting at 9AM). For this reason we consider, as the basis for the timetable, a list of possible time-slots, each having a specified duration, and a binary relation defining which pairs of time-slots overlap one another (for example, the 2-hours 9AM-11AM time-slot overlaps two 1:30-hours time-slots, namely 8AM-9:30AM and 9:30AM-11AM).
- *Teachers:* As usual, teachers may have availability constraints and preferences, which may be defined on a per-time-slot basis.
- *Student groups:* In the IUT curriculum, students at a certain level (*semester group*, usually somewhere around one hundred students) are divided in classes of around 26 students (called *TD groups*) which are further divided in two sub-groups (called *TP groups*). Teachings take the form of *lectures* (delivered to a whole semester group), "TDs" (delivered separately to each TD group) and "TPs" (delivered separately to each TP group). Obviously, the IUT timetable has to take into account this division and not plan an event for a sub-group in parallel to another event which concerns any one of its super-groups.
- *Rooms:* There are different categories of rooms: amphitheatres (fitted for a whole semester group), basic classrooms (fitted for a TD group), computer and multi-media rooms (fitted for a TP group or in some cases for a TD group). In each category there may be several rooms, which are often interchangeable. Quantity is however important and is a limiting factor for the timetable (for example, there are 6 computer rooms for a total of 14 TP groups in Blagnac, so there may not be more than 6 computer TPs in parallel). Moreover, due to sharing with other departments, the quantity for a room category may vary from one slot to another (e.g., 2 amphitheatres are available on Monday morning, but not the other days).
- *Courses:* In the IUT curriculum, a course is composed of a certain number of lectures, TDs and TPs. However, to ease the definition, we will consider each of these components as a separate course, for each of the instructors involved. Therefore, in our problem definition, a course is characterised by:
  - a course name
  - an instructor
  - a time-slot duration (1:30, 2 hours...)
  - a room category as well as a quantity of rooms that are used for one session (usually one, but sometimes certain courses need two computer rooms at once)
  - the number of sessions for each group concerned

  In addition, there may be several other types of constraints on courses:
  - *consecutiveness*: two or more sessions must be scheduled consecutively in the timetable,

- *precedence*: the sessions of one course must precede the sessions of another course, for example when the amphitheatres lecture must precede the corresponding TDs or TPs,
- *time-slot constraints*: certain courses may prefer or forbid the use of certain time-slots, for example one would forbid amphitheatre lectures late in the evening and prefer morning to after-noon slots. Also, in Blagnac, most courses are forbidden to take place on Thursday afternoon which is reserved for sports and other activities.

With the exception of the overlapping time-slots of different duration, all the elements and constraints expressed above can in principle be captured in the XHSTT format [9]. However, there is not a 1-to-1 correspondence between the elements above and those of an XHSTT model: for example, XHSTT resources such as rooms cannot specify a multiplicity and therefore one would have to model the individual rooms and use resource groups in order to define room categories. For this reason, the ILP formulation presented in the next section is simpler and more direct than one based on the general XHSTT format, such as [5]. Also, a timetable constructed based on an XHSTT formulation of our problem would necessarily include more detailed information (e.g., individual rooms as opposed to just the room type), the computation of which would incur an overhead.

3 The ILP formulation

3.1 Model constants

This section lists, by category, the various data that define an instance of the problem. They correspond to the elements listed informally in section 2.

- *Time slots:*
  - *Slots* is an set of time-slots (identifiers)
  - *SlotTypes* is a set of time-slot types (identifiers; for example, the durations may be used as types)
  - $slType : Slots \to SlotTypes$ is a function assigning a type to each slot
  - $overlap \subseteq Slot \times Slot$ is a binary relation on slots. $(s_1, s_2) \in overlap$ (denoted $s_1 \parallel s_2$) iff slots s_1 and s_2 overlap chronologically
  - $rank : Slots \to \mathbb{N}$ is a numbering function which defines the partial order between slots. The function is used for imposing consecutiveness and precedence constraints between events, defined below. As such, the function has to observe two constraints:
    - two chronologically consecutive slots have to be assigned consecutive natural numbers as *rank*
    - a slot that chronologically precedes another must have a lower *rank* than the other

    Note that with these constraints, the *rank* function defines only a *partial* order, in particular in the case of overlapping slots (e.g., two overlapping slots may be assigned the same rank). Moreover, the *rank* function may be used to model gaps in the timetable, e.g., nights: it suffices to leave a gap bigger than 1 between the *rank* of the last slot of a day and the *rank* of the first slot of the next day.

- *Teachers:*
  - *Teachers* is a set of teachers (identifiers)
  - $tAvail : Teachers \times Slots \to \{0, 1\}$ defines teacher availability per time-slot. $tAvail(t, s) = 1$ iff teacher t is available for slot s
  - $tPref : Teachers \times Slots \to \{0, 1\}$ defines teacher preferences per time-slot. $tPref(t, s) = 1$ iff s is a preferred slot for teacher t. Note that $tPref(t, s) = 1$ implies $tAvail(t, s) = 1$.
- *Student groups:*
  - *Groups* is a set of student groups (identifiers)
  - $subgroup \subseteq Groups \times Groups$ is a binary relation, $(g_1, g_2) \in subgroup$ (denoted $\prec$) iff g_1 is a (strict) sub-group of g_2. Note that the relation must be acyclic.
- *Rooms:*
  - *RoomCategories* is a set of room categories (identifiers)
  - $rAvail : RoomCategories \times Slots \to \mathbb{N}$ defines how many rooms of a particular category are available at each time-slot
- *Courses:*
  - *Courses* is a set of courses (identifiers)
  - $courseTeacher : Courses \to Teachers$ defines who teaches the course. If different parts of a curriculum course are taught by more than one teacher (for the same or for different groups), they will appear as separate elements in $Courses$.
  - $courseSlotType : Courses \to slType$ defines what kind of slot type is used by the course
  - $courseRoomCat : Courses \to RoomCategories$ defines what kind of room is used by the course
  - $courseRoomNb : Courses \to \mathbb{N}$ defines how many rooms are used by the course (usually 1)
  - $courseGroupNb : Courses \times Group \to \mathbb{N}$. $courseGroupNb(c, g)$ defines how many sessions there are in the course c for the group g.
  - $consecCourses \subseteq Courses$. If $c \in consecCourses$ then the sessions of this course for each concerned group have to be consecutive (applies to groups g such that $courseGroupNb(c, g) > 1$).
  - $courseAvail : Courses \times Slots \to \{0, 1\}$ defines availability per time-slot. $courseAvail(c, s) = 1$ iff course c may be scheduled at slot s. Note that this function may also be used to model a partial solution for the timetable, if one is available: the events that are already scheduled are assigned a $courseAvail$ of 0 for all the other slots except those that are pre-assigned.
  - $coursePref : Courses \times Slots \to \{0, 1\}$ defines preference per time-slot. $coursePref(c, s) = 1$ iff s is a preferred slot for course c.
    Note that $coursePref(c, s) = 1$ implies $courseAvail(c, s) = 1$.
  - $precedes \subseteq Course \times Course$ defines the precedence between courses. $(c_1, c_2) \in precedes$ (denoted $c_1 \ll c_2$) means all time-slots allocated to c_1 must chronologically precede the slots of c_2. Note that the relation must be acyclic.

3.2 Model variables and objective function

The model uses the following set of variables $\mathbf{x}_{s,c,g} \in \{0,1\}$ where $s \in Slots$, $c \in Courses$ and $g \in Groups$, taking the value 1 when group g has a course c in time-slot s and 0 otherwise. Note that since rooms are not explicitly modelled as resources to be assigned, we need not index x on rooms: the model only needs to satisfy the constraint that enough rooms of each type are available at every slot. This reduces the size of the set of decision variables making resolution more efficient.

On a first approach, the cost function has two components, which may be summed and assigned different weights according to the needs:

- a cost incurred by the use of unpreferred slots of teachers

$$UT = \sum_{\substack{s \in Slots \\ c \in Courses \\ g \in Groups}} (tAvail(courseTeacher(c), s) - tPref(courseTeacher(c), s)) \cdot \mathbf{x}_{s,c,g} \tag{1}$$

- a cost incurred by the use of unpreferred slots for courses

$$UC = \sum_{\substack{s \in Slots \\ c \in Courses \\ g \in Groups}} (courseAvail(c, s) - coursePref(c, s)) \cdot \mathbf{x}_{s,c,g} \tag{2}$$

In Section 3.4 we discuss how the model may be augmented to optimize other criteria.

3.3 Hard constraints

All the constraints listed in the following are hard, i.e. have to hold for the timetable to be considered valid.

- All the courses are allocated a slot:

$$\forall c \in Courses, \forall g \in Groups : \sum_{s \in Slots} \mathbf{x}_{s,c,g} = courseGroupNb(c, g) \tag{3}$$

- No group has two courses in parallel (or a course in parallel with another course of one of its super-groups):

$$\forall s \in Slots, \forall g \in Groups :$$

$$\sum_{c \in Courses} \left(\mathbf{x}_{s,c,g} + \sum_{\substack{g' \in Groups \\ g \prec g'}} \mathbf{x}_{s,c,g'} \right) \leq 1 \tag{4}$$

Since this constraint has also to take into account slot overlapping, an additional constraint is necessary for every pair of overlapping slots:

$$\forall s, s' \in Slots \text{ such that } s \parallel s', \forall g \in Groups :$$

$$\sum_{c \in Courses} \left(\mathbf{x}_{s,c,g} + \mathbf{x}_{s',c,g} + \sum_{\substack{g' \in Groups \\ g \prec g'}} (\mathbf{x}_{s,c,g'} + \mathbf{x}_{s',c,g'}) \right) \leq 1 \tag{5}$$

– The slots allocated to a *Course* have the right *SlotType*:

$$\forall s \in Slots, c \in Courses \text{ such that } courseSlotType(c) \neq slType(s),$$

$$\forall g \in Groups : \mathbf{x}_{s,c,g} = 0 \tag{6}$$

– The teachers are available at the allocated slots and a teacher does not have several courses in parallel:

$$\forall s \in Slots, \forall t \in Teachers :$$

$$\sum_{\substack{c \in Courses \\ courseTeacher(c)=t}} \sum_{g \in Groups} \mathbf{x}_{s,c,g} \leq tAvail(t,s) \tag{7}$$

– There are enough rooms of each category for each slot:

$$\forall s \in Slots, \forall t \in RoomCategories :$$

$$\sum_{\substack{c \in Courses \\ courseRoomCat(c)=t}} \sum_{g \in Groups} courseRoomNb(c) \cdot \mathbf{x}_{s,c,g} \leq rAvail(t,s) \tag{8}$$

Since this constraint has also to take into account slot overlapping, an additional constraint is necessary for every pair of overlapping slots:

$$\forall s, s' \in Slots \text{ such that } s \parallel s', \forall t \in RoomCategories :$$

$$\sum_{\substack{c \in Courses \\ courseRoomCat(c)=t}} \sum_{g \in Groups} courseRoomNb(c) \cdot (\mathbf{x}_{s,c,g} + \mathbf{x}_{s',c,g})$$

$$\leq min(rAvail(t,s), rAvail(t,s')) \tag{9}$$

where *min* denotes the minimum of two numbers.
Note that this may overconstrain the model in the case where $rAvail(t,s)$ and $rAvail(t,s')$ are different. However, in most cases *rAvail* should be equal for overlapping slots s, s'.

– *courseAvail* constraint is observed:

$$\forall c \in Courses, \forall s \in Slots : \sum_{g \in Groups} \mathbf{x}_{s,c,g} \leq courseAvail(c,s) \tag{10}$$

– Consecutiveness constraints are observed:

$$\forall c \in consecCourses, \forall g \in Groups,$$

$$\forall s, s' \in Slots \text{ such that } rank(s') - rank(s) \geq courseGroupNb(c,g) :$$

$$\mathbf{x}_{s,c,g} + \mathbf{x}_{s',c,g} \leq 1 \tag{11}$$

– Precedence constraints are observed:

$$\forall c, c' \in Courses \text{ such that } c \ll c',$$

$$\forall g, g' \in Groups \text{ such that } courseGroupNb(c,g) \cdot courseGroupNb(c',g') > 0,$$

$$\forall s, s' \in Slots \text{ such that } rank(s) < rank(s') :$$

$$\mathbf{x}_{s,c',g'} + \mathbf{x}_{s',c,g} \leq 1 \tag{12}$$

3.4 Integrating soft constraints

As mentioned in Section 3.2, the model aims at minimizing the use of unpreferred slots (with preferences given per-teacher or per-course). However, it is possible to augment the model to take into account other types of soft constraints. We discuss here two examples of soft constraints.

Minimizing "long days". When the difference of time between the first and the last slot of a day is big, such as in our running example where days run from 8AM to 6:45PM, it is important from a human point of view to avoid as much as possible days that start very early in the morning and end very late in the evening, both for teachers and for groups of students. In order to minimize such long days, say for teachers, the model may be augmented with the following:

- $Days$ is a set of the days of week
- $first : Days \to Slots$ maps each day to its first slot
- $last : Days \to Slots$ maps each day to its last slot
- $\mathbf{L}_{d,t} \in \{0, 1\}$, where $d \in Days$ and $t \in Teachers$, is an auxiliary variable which will be assigned the value 1 when d is a "long day" for teacher t (i.e., t teaches during both the first and the last slot of the day)
- the constraint linking $\mathbf{L}$ to $\mathbf{x}$:

$$\forall d \in Days, t \in Teachers :$$
$$\sum_{\substack{c \in Courses \\ courseTeacher(c)=t}} \sum_{g \in Groups} (\mathbf{x}_{first(d),c,g} + \mathbf{x}_{last(d),c,g}) - 2 \cdot \mathbf{L}_{d,t} \leq 1 \qquad (13)$$

- an additional component, which models the cost incurred by the "bad days", may then be weighted into the cost function:

$$LT = \sum_{\substack{t \in Teachers \\ d \in Days}} \mathbf{L}_{d,t} \qquad (14)$$

Minimizing long days for student groups works much in the same way as for teachers.

Clustering busy times. The idea here is to group the time slots in which a teacher or student group has courses around certain days (or half-days) while freeing up others. It is useful for freeing-up blocks of time either for students to allow for individual work, or for adjunct teachers who have another main job. For example, the model may be augmented with the following in order to minimize the number of presence days for teachers:

- $day : Slots \to Days$ maps each slot to a day of week
- $\mathbf{D}_{d,t} \in \{0, 1\}$, where $d \in Days$ and $t \in Teachers$, is an auxiliary variable which will be assigned the value 1 when t has at least a busy slot on day d ("used day")
- the constraint linking $\mathbf{D}$ to $\mathbf{x}$:

$$\forall d \in Days, t \in Teachers :$$
$$card(Slots) \cdot \mathbf{D}_{d,t} - \sum_{\substack{c \in Courses \\ courseTeacher(c)=t}} \sum_{g \in Groups} \sum_{\substack{s \in Slots \\ day(s)=d}} \mathbf{x}_{s,c,g} \geq 0 \qquad (15)$$

where $card(Slots)$ denotes the cardinal of the set $Slots$.

Proceedings of the 11th International Confenference on Practice and Theory of Automated Timetabling (PATAT-2016) – Udine, Italy, August 23–26, 2016

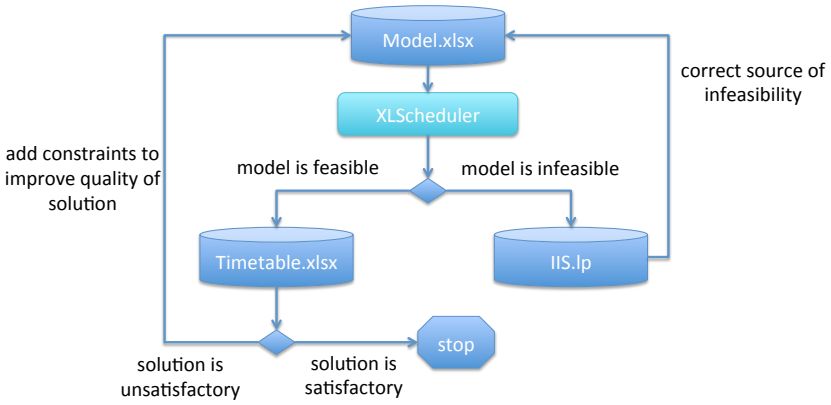

Fig. 1 Workflow of the XLScheduler tool.

– an additional component, which models the cost incurred by "used days", may then be weighted into the cost function for some (or for all) teachers t:

$$BT_t = \sum_{d \in Days} \mathbf{D}_{d,t} \tag{16}$$

4 Tool and experimental results

The model described above was implemented in a software tool, XLScheduler (`https://www.irit.fr/~Iulian.Ober/XLScheduler`). The software works in batch mode, reading the model constants (see Section 3.1) from a spreadsheet that has to observe a certain tabular format, described in the following. From this data, the software creates the ILP model and solves it with an off-the-shelf solver, which can be either Gurobi [14] or the open-source solver Cbc [12]. Two results are possible: either a feasible timetable is found, in which case it is displayed in text form and written to an Excel file with a convenient format, or the problem is infeasible. In the latter case, if the used solver is Gurobi, it is possible to generate an Irreducible Inconsistent Subsystem (IIS) in textual LP format, from which the user may be able to find the cause of infeasibility, at least in simple cases such as when the availability of a teacher does not allow him to deliver all his courses. Such cases are generally caused by over-restrictive constraints added by the teachers and are resolved by relaxing the constraints. The workflow is summarized in Figure 1 and it may be iterated several times, while adding soft constraints to the model file to improve the quality of the solution.

A web-based trial version of the tool is available at `https://www.irit.fr/~Iulian.Ober/XLScheduler/submit.php`. However, due to licencing limitations, the web-based version can only use Cbc as back-end solver. This limitation severely impacts the performance of the tool, as discussed below (§4.2).

4.1 Input/Output formats

Ease of use being one of the primary concerns for the tool, the input format for the tool is a spreadsheet. At most institutions, the timetabling data (teachers list, availability, etc.) is already managed with a spreadsheet software and therefore data is readily convertible to the format used by the tool. The input spreadsheet must contain eight sheets:

- *TimeSlots*: a table of time slots (names, types, overlapping relations, etc.)
- *InstructorAvailability*: a table of teachers with availability/preference per-slot
- *Groups*: a list of groups and value table for the *subgroup* function
- *Rooms*: a table of room categories and quantity per-slot
- *Courses*: a table of courses with associated data (teacher, slot type, room category and quantity, group assignment and quantity)
- *CourseSlotPrefs*: a value table for $courseAvail$ and $coursePref$ functions
- *CoursePrecedence*: a table defining the $precedes$ relation
- *Objectives*: a table defining the relative weights for the components of the cost function

An example file is available online[1]. The format presented above takes into account the hard constraints detailed in §3.3 and the objectives from §3.2. Adding soft constraints such as the ones discussed in Section 3.4 is less obvious since such soft constraints are hard to generalize and to specify in a generic way in the input file. For now, the solution we use is to hard-code them in a custom version of the tool. Making them part of the standard version of the tool would require extending the input format; since defining a generic-enough and understandable format is not always easy, we plan to do this when enough user requests justify the need.

If a feasible timetable is found by the tool, it is also output into a convenient spreadsheet format. An example of automatically generated timetable is available online [2].

4.2 Back-end solvers and experimental results

Two solvers may be used as back-end for XLScheduler: Gurobi [14] or the open-source solver Cbc [12]. In both cases, the interface with the solver passes through a Python [3] API: the native Python API for Gurobi, the PuLP [7] API for Cbc. The two APIs are quite similar and allow to express the model elements (see §3) in a natural way.

The tool was used on a real-life model with the following characteristics:

- 39 teachers
- 2 semester groups divided into 7 TD groups and 14 TP groups
- up to 15 rooms of 4 different categories, depending on the time-slot. Although this may seem quite a lot, a large part of the curriculum is delivered in TPs, and there are at most 6 rooms available for the 14 groups at any time.
- 83 courses. A course is defined by a unique triple (subject,group type,teacher) but may concern several groups.

[1] `https://www.irit.fr/~Iulian.Ober/XLScheduler/modelExample.xlsx`
[2] `https://www.irit.fr/~Iulian.Ober/XLScheduler/resultExample.xlsx`

– 169 events (course instances) to schedule

The configurations of five different weeks at our institution, with varying courses and availability constraints, were solved with the tool. Both solvers were able to solve all models to optimality, albeit with different performance: usually under 1 second with Gurobi, in the order of 3 minutes for the PuLP/Cbc version.However, on a closer analysis, the performance difference is mainly caused not by the solver, as Cbc also solves the model in a few seconds, but by the PuLP API during model creation. An alternative method for creating the Cbc model is currently sought.

The quality of the generated timetable could be adjusted by tightening the availability and preferences over time-slots, and in some cases by adding custom soft constraints such as the ones discussed in Section 3.4. In the end, the quality of the timetable from a human point of view is comparable to that obtained manually. In one case, it was possible to find a satisfactory solution where, manually, one could not be found without adding an extra resource (another computer room).

5 Conclusion

Efficiently obtaining good quality timetables is a problem faced by many teaching institutions and the need for practical tools to support this problem is recognized. This is even more so in institutions where the timetable is highly dynamic from one week to another, as it is the case in our example. The paper presents a practical approach for handling a particular class of timetabling problem, which has been implemented in an easy to use and extensible tool that handles timetable data in a simple spreadsheet format, facilitating its adoption by the concerned personnel.

The class of problems tackled by the tool is a relatively classical HSTT problem. However, it presents a few distinctions that set it apart, such as the existence of overlapping time slots or the extensive use hierarchically organized student groups. In some ways, the problem is also simpler than the general HSTT problem, since teachers are pre-assigned to groups and rooms within the same category are considered interchangeable, which make it possible to define a simpler and more efficient ILP-based model for it. The results obtained on real-life examples of workloads from our institution are very good, both in terms of quality of the generated timetable and of tool performance and ease of use.

References

1. T. Birbas, S. Daskalaki, and E. Housos. School timetabling for quality student and teacher schedules. *Journal of Scheduling*, 12(2):177–197, 2008.
2. Natashia Boland, Barry D. Hughes, Liam T.G. Merlot, and Peter J. Stuckey. New integer linear programming approaches for course timetabling. *Computers & Operations Research*, 35(7):2209 – 2233, 2008. Part Special Issue: Includes selected papers presented at the ECCO'04 European Conference on combinatorial Optimization.
3. Python Language Documentation. `https://www.python.org/doc/`.
4. Jeffrey H. Kingston. *Automated Scheduling and Planning: From Theory to Practice*, chapter Educational Timetabling, pages 91–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
5. Simon Kristiansen, Matias Sørensen, and Thomas R. Stidsen. Integer programming for the generalized high school timetabling problem. *Journal of Scheduling*, 18(4):377–392, 2014.

6. Simon Kristiansen and Thomas Riis Stidsen. A comprehensive study of educational timetabling-a survey. Technical report, Department of Management Engineering, Technical University of Denmark, 2013.
7. Stuart Mitchell. An Introduction to pulp for Python Programmers. *The Python Papers Monograph*, 1, 2009. Available online at `http://ojs.pythonpapers.org/index.php/tppm`.
8. Nelishia Pillay. A survey of school timetabling research. *Annals OR*, 218(1):261–293, 2014.
9. Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, and David Ranson. An XML format for benchmarks in high school timetabling. *Annals OR*, 194(1):385–397, 2012.
10. Gerhard Post, Jeffrey H. Kingston, Samad Ahmadi, Sophia Daskalaki, Christos Gogos, Jari Kyngäs, Cimmo Nurmi, Nysret Musliu, Nelishia Pillay, Haroldo Santos, and Andrea Schaerf. XHSTT: an XML archive for high school timetabling problems in different countries. *Annals OR*, 218(1):295–301, 2014.
11. Haroldo G. Santos, Eduardo Uchoa, Luiz Satoru Ochi, and Nelson Maculan. Strong bounds with cut and column generation for class-teacher timetabling. *Annals OR*, 194(1):399–412, 2012.
12. Cbc solver website. `https://projects.coin-or.org/Cbc`.
13. Christos Valouxis, Christos Gogos, Panayiotis Alefragis, and Efthymios Housos. Decomposing the high school timetable problem. In *Practice and Theory of Automated Timetabling*, 2012.
14. Gurobi website. `http://www.gurobi.com`.