

Uporabniška in tehnična dokumentacija
Projektni seminar 2019/2020
The automated timetable at UP FAMNIT

Nevena Pivač

July, 2020

Contents

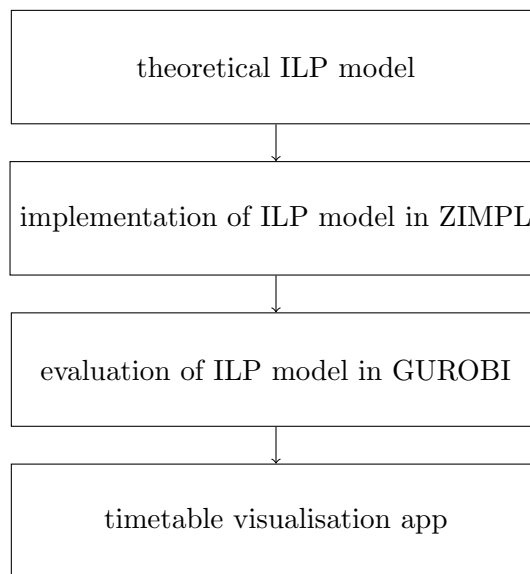
1	Introduction	3
2	About the project	3
3	Development Environment	3
4	Integer Linear Program	3
4.1	Parameters of the ILP	3
4.2	Variables of the ILP	5
4.3	Constraints of the ILP	5
4.4	Soft constraints	7
4.5	The objective function	9
5	Implementation of ILP model - ZIMPL	10
5.1	Objective function	11
5.2	Constraints	11
5.3	Input .txt files	12
5.4	Zimpl evaluation	14
6	Evaluation of ILP model - Gurobi Optimizer	15
7	Timetable app - visualisation	16
7.1	Features	16
7.1.1	Nothing was selected	16
7.1.2	Display relevant events	17
7.1.3	Day view	17
7.1.4	Week view	18
7.1.5	Month view	18
7.1.6	Tasks view	18
7.2	Full Calendar	19
7.3	Methods	19

1 Introduction

In this document one can find the documentation of the project. This project represents a smart solution, so a user that wants to use all parts of the project is supposed to understand the underlying things, and that is why he should go through the documentation. The project represents the implementation of an integer linear programming model for a timetable problem, so if a user wants to produce a new timetable solution, should understand the model, and how to change its single part.

2 About the project

In this project the smart solution of a timetable problem is presented. We developed the Integer Linear Programming model (ILP model) for a timetable problem at UP FAMNIT, and implemented it using ZIMPL. Then the model is evaluated using Gurobi software, and finally displayed using the timetable visualisation app. Every part of this smart solution is described in its own chapter, and mostly can be read independently. In case of any changes in the model (conditions, restrictions,etc.) the user should read the Section 4. The scheme of the project is presented in figure below.



3 Development Environment

For the implemetation of ILP model we use ZIMPL software, and for its evaluation we use Gurobi software. The visualisation part represents a single page web app, developer in HTML and Javascript, using Angular library. A calendar in the timetable app is based on an open-source Javascript calendar library, called FullCalendar. The app is styled using Bootstrap and is responsive. All the components of the app are described in the rest of this documentation. The whole project is developed using the FAMNIT server:

- CPU: Intel Xeon 80 cores @ 3Ghz
- RAM : 1,5TB ECC DDR4 RAM
- Disk : Server grade NVME 2TB.

4 Integer Linear Program

In this section we present an integer linear programming formulation for the university timetabling problem at UP FAMNIT.

4.1 Parameters of the ILP

Here we define the structural elements of FAMNIT TIMETABLE PROBLEM and describe them in detail.

- M is defined as the set of meetings to be assigned: every element $m \in M$ is an ordered pair containing a course as the first coordinate, and a set of student subgroups as the second one. All meetings have a type determined: lectures or tutorials. The set of meetings in M labelled as “lectures” or “tutorials” will be denoted by M^{lec} and M^{tut} , respectively. Obviously, M is the disjoint union of these two sets. An example of element $m \in M$ is an ordered pair (Course 1, $\{s_1, s_2\}$). For any meeting, lecturer and number of hours of the meeting to be assigned are known, so for meeting m , number of hours per week is a_m . Any meeting m is divided in parts, with respect to lecturers’ desire. These parts will be called blocks. So for any meeting m there is a vector p_m , with element $p_m(i) = k$, if a block of duration i of meeting m has to be repeated k times per week. For any meeting m it holds that $\sum_i p_m(i) \cdot i = a_m$. For any meeting m we define a set H_m to be the set of all block lengths appearing in the division of meeting m , i.e., $H_m = \{i \mid p_m(i) \neq 0\}$. Let us introduce a short example: let meeting m with $a_m = 5$ be separated in two blocks, with durations of 2 and 3 hours, respectively. Such a division is denoted as $(2 + 3)$. Then the corresponding vector is $p_m = (0, 1, 1)$ and so $H_m = \{2, 3\}$. Another way of division of meeting m is in two blocks of length 1 and one block of length 3, denoted by $1 + 1 + 3$. In that case we have $p_m = (2, 0, 1)$ and $H_m = \{1, 3\}$.
- L : the set of lecturers.
- D : the set of days in a week, $D = \{1, 2, 3, 4, 5\}$.
- R : the set of classrooms.
- T : the set of all timeslots in the week; each element $t \in T$ is an ordered pair, $t = (d, h)$, where d and h represent day and timeslot within the day, respectively; in this work timeslots are supposed to have length 60 minutes. T is linearly ordered set, with $t < t'$, where $t = (d, h)$, $t' = (d', h')$, if $d < d'$ or if $d = d'$ and $h < h'$. The set of timeslots belonging to day d is denoted by the T_d . The number of timeslots in one day is denoted by a constant τ (hence $h \in \{1, \dots, \tau\}$, so the number of all timeslots is 5τ ; in this implementation we have $\tau = 13$, each timeslot is 60 minutes long, starting at 8AM until 9PM). Given a timeslot $t = (d, h)$ and a number i , such that $i \leq \tau - h$, we have $t + i := (d, h + i)$.
- S : the set of student groups.
- M_ℓ : the set of class meetings given by lecturer $\ell \in L$.
- M_s : the set of class meetings of a group of students $s \in S$.
- M_k : the set of class meetings that have to take place at location $k \in K$, in particular: M_{k_1}, M_{k_2} ;
- R_m : the set of rooms that are acceptable for meeting $m \in M$. A classroom is acceptable for a particular meeting m if it satisfies requirements connected with equipment of classroom and if it has sufficient capacity. From this set we can construct a set M^R of ordered pairs (m, r) containing all acceptable combinations of meetings m and rooms r : $M^R := \{(m, r) \mid m \in M, r \in R_m\}$.
- T_ℓ : the set of timeslots $t \in T$ in which lecturer $\ell \in L$ can have lectures. Here few conditions have to be included. First, individual requirements of lecturers are reflected in this set. The first and the last timeslots in a day are not acceptable for lecturers not situated in same municipality as the institution. If a lecturer has another job, is a member of some committee, or has some other regular obligations, these constraints are also assumed to included in the set T_ℓ . For instance, every Monday at 10AM teaching staff of the Mathematics department should have the possibility to attend the Mathematical research seminar. In order to make this possible, they should not have any teaching obligations at that time. Similarly, every Monday at 4PM teaching staff of Department of Information Sciences and Technologies (abbreviated as DIST) should have the possibility to attend the DIST research seminar, so there are no teaching obligations for them at that time.
- T_r : the set of timeslots $t \in T$ in which a given classroom $r \in R$ can be used.
- M_{DIST} : set of the meetings given by lecturers working in Department of Information Sciences and Technologies – in order to make fewer overlapping with DIST research seminar;
- M_{PM} : the set of meetings that are supposed to be assigned at afternoon timeslots;

- T_{AM} : the set of morning timeslots; denote the number of morning timeslots in a day by τ_{AM} ;
- T_{PM} : the set of afternoon timeslots; denote the number of afternoon timeslots in a day by τ_{PM} ;
- T_{MAS} : the set of timeslots for the Mathematical research seminar;
- T_{DIST} : the set of timeslots for the DIST research seminar;

4.2 Variables of the ILP

In the integer linear program all variables are binary. There are three different sets of variables and we list them in the following.

- **x -variables:** For every triple of a meeting $m \in M$, a timeslot $t \in T$, and a room $r \in R_m$ that is acceptable for that meeting, there is one corresponding variable $x_{m,t,r}$. This variable takes value 1 if meeting m is scheduled at timeslot t in classroom r , and 0 otherwise:

$$x_{m,t,r} = \begin{cases} 1, & \text{if meeting } m \text{ is scheduled at timeslot } t \text{ in classroom } r, \\ 0, & \text{otherwise.} \end{cases}$$

- **y -variables:** For every triple of a meeting $m \in M$, a timeslot $t \in T$ and a predefined length $i \in H_m$ of individual blocks of meeting m we define a variable $y_{m,t,i}$. The variable takes value 1 if timeslot t is the first appearance of i consecutive hours of m , and 0 otherwise:

$$y_{m,t,i} = \begin{cases} 1, & \text{if timeslot } t \text{ is first appearance of } i \text{ consecutive hours of meeting } m, \\ 0, & \text{otherwise.} \end{cases}$$

- **z -variables:** In the last set of variables are so called z -variables, auxiliary variables for modelling some soft constraints. Given a constraint of type p and the corresponding index set I_p , we define a variable $z_{p,i}$ for every $i \in I_p$. Values of such variables will be determined by description of corresponding constraint. These variables will appear in the modelling of soft constraints of types S_2 and S_3 (Section 4.4).

4.3 Constraints of the ILP

A) Every meeting has to be assigned to available resources:

A₁) Lecturers cannot have lectures at unacceptable timeslots:

$$\sum_{m \in M_\ell} \sum_{t \in T \setminus T_\ell} \sum_{r \in R_m} x_{m,t,r} = 0 \quad \forall \ell \in L.$$

A₂) Classrooms can only be used at specified timeslots:

$$\sum_{(m,r) \in M^R} \sum_{t \in T \setminus T_r} x_{m,t,r} = 0, \quad \forall r \in R.$$

A₃) Every meeting has to take place in an acceptable classroom: this constraint is already satisfied by definition of variables, since $x_{m,t,r}$ variables are defined just for classrooms $r \in R$ that satisfy classroom requirements for the meeting $m \in M$.

B) Overlapping is not permitted

B₁) For every student group at most one meeting and one classroom can be assigned to every teaching period:

$$\sum_{m \in M_s} \sum_{r \in R_m} x_{m,t,r} \leq 1 \quad \forall s \in S, \forall t \in T.$$

B₂) Every member of the teaching staff shall be assigned at most one meeting and one classroom at a time:

$$\sum_{m \in M_\ell} \sum_{r \in R_m} x_{m,t,r} \leq 1 \quad \forall \ell \in L, \forall t \in T.$$

B₃) Every classroom can be assigned to at most one meeting at a time:

$$\sum_{(m,r) \in M^R} x_{m,t,r} \leq 1 \quad \forall r \in R, \forall t \in T.$$

C) Timetable has to be complete:

C₁) All meetings in the curriculum of each student subgroup should be in the timetable and in the right amount of teaching periods, with respect to weekly duration:

$$\sum_{t \in T} \sum_{r \in R_m} x_{m,t,r} = a_m \quad \forall m \in M.$$

C₂) A meeting m of duration $i \in H_m$ has to start and finish at the same day, so some variables $y_{m,t,i}$ are defined to have value 0:

$$y_{m,t,i} = 0 \quad \forall m \in M, \forall i \in H_m, \forall t = (d, h) \in T \text{ s.t. } h > \tau - i + 1.$$

Recall that the parameter τ represents the number of timeslots in a day.

C₃) Given a meeting m , at most one timeslot can be the first appearance of m in a single day:

$$\sum_{i \in H_m} \sum_{t \in T_d} y_{m,t,i} \leq 1, \quad \forall m \in M, \forall d \in D.$$

C₄) A given meeting m of duration i (where i is the index of a nonzero element of vector p_m) has to appear exactly $p_m(i)$ times per week (i.e. in $p_m(i)$ days). All such indices i are contained in set H_m , so we have:

$$\sum_{t \in T} y_{m,t,i} = p_m(i), \quad \forall m \in M, \forall i \in H_m.$$

C₅) If a course m of duration i is assigned at day d , it has to be assigned to exactly i hours:

$$i \cdot \sum_{t \in T_d} y_{m,t,i} \leq \sum_{r \in R_m} \sum_{t \in T_d} x_{m,t,r}, \quad \forall m \in M, \forall d \in D, \forall i \in H_m.$$

C₆) Appearances of meeting m of duration i in a single day should be consecutive:

$$y_{m,t,i} \leq \sum_{r \in R_m} x_{m,t+j,r}, \quad \forall t = (d, h) \in T, \forall m \in M, \forall i \in H_m, \forall j = 0, \dots, i - 1,$$

where for $t = (d, h)$, such that $h + j \leq \tau$ we have $t + j := (d, h + j)$.

C₇) All consecutive hours of one meeting should take place in the same classroom:

$$x_{m,t,r} + x_{m,t+1,r'} \leq 1 \quad \forall m \in M, \forall t \in T, \forall r, r' \in R_m \text{ s.t. } r \neq r'.$$

D) Pre-scheduled meetings:

D₁) There are activities at the Faculty that do not belong to the course offer. Such activities represent pre-scheduled meetings, which should be scheduled to already determined pairs of timeslots and classrooms. If a meeting is related to some lecturers or students groups, their regular course meetings are undesirable at that time. In particular, Research seminar at Mathematics' department have to be assigned to Monday, 10AM, in one specified room, call it r_1 . Hence, that room is unavailable for other lectures at that time. Also, the DIST research seminar has to be assigned to Monday, 4PM, in an already defined room, r_2 .¹

$$\begin{aligned} \sum_{m \in M} \sum_{t \in T_{MAS}} x_{m,t,r_1} &= 0 \\ \sum_{m \in M} \sum_{t \in T_{CSS}} x_{m,t,r_2} &= 0 \end{aligned}$$

¹These activities are also related to a subset of teaching staff; however, these requirements are already included in the definition of time availability of lecturers, T_l .

D_2) Meetings appearing as first coordinates of elements in set G have predetermined timeslot and classrooms:

$$x_{m,t,r} = 1, \quad \forall (m, t, r) \in G.$$

E) Upper bounds on number of hours at day level

E_1) It is not desired for lecturer ℓ to have more than ρ_ℓ timeslots of teaching obligations per day:

$$\sum_{t \in T_d} \sum_{m \in M_\ell} \sum_{r \in R_m} x_{m,t,r} \leq \rho_\ell, \quad \forall \ell \in L, \quad \forall d \in D.$$

There are also upper bounds with respect to meeting type, here we define it parametrically as ρ_1 and ρ_2 for types “lectures” and “tutorials”, respectively.

$$\sum_{t \in T_d} \sum_{m \in M_\ell \cap M^{lec}} \sum_{r \in R_m} x_{m,t,r} \leq \rho_1, \quad \forall \ell \in L, \quad \forall d \in D,$$

$$\sum_{t \in T_d} \sum_{m \in M_\ell \cap M^{tut}} \sum_{r \in R_m} x_{m,t,r} \leq \rho_2, \quad \forall \ell \in L, \quad \forall d \in D.$$

Also, one more condition can be introduced, specifying that the lecturers shall not have more than δ timeslots of teaching in every block of Δ timeslots per day.

$$\sum_{j=0}^{\Delta-1} \sum_{m \in M_\ell} \sum_{r \in R_m} x_{c,t+j,r} \leq \delta \quad \forall \ell \in L, \quad \forall t = (d, h) \in T \text{ s.t. } h \leq \tau - \Delta$$

This type of constraints assures, for example, that for every lecturer, every period of Δ timeslots of teaching is preceded and followed by a break of length at least $\Delta - \delta$ timeslots.

4.4 Soft constraints

Among the implicitly generated feasible solutions, we would like to get the best one. In order to do that, we define an objective function, and an optimal solution is one that will give the minimal value to the objective function. If some soft constraint is violated, then the objective function value will grow. Soft constraints will be included in the objective function as the sum of relevant variables, multiplied by the corresponding weights. If a soft constraint of type p cannot be included in the objective function using x - and y - variables, it will be modelled using z -variables, namely $z_{p,i}$, for every $i \in I_p$, where I_p is the index set relevant for constraints of type p . In this section we introduce a representation of each soft constraint in order to construct the corresponding variables. Given a constraint of type p , variable $z_{p,i}$, if exists, has following definition:

$$z_{p,i} = \begin{cases} 1, & \text{if constraint of type } p \text{ is not satisfied for element } i \text{ of the index set } I_p, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

From equation (1) it follows that a violation of the constraint of type p , for some $i \in I_p$, can be represented by a penalty term in the objective function, defined as $w_i z_{p,i}$, where w_i is some positive weight. Obviously, a positive penalty increases the value of the objective function, so solutions containing violated soft constraints will have a larger value of the objective function.

The soft constraints are formalized as follows:

S_1) **Minimize use of payable classrooms.** Some classrooms are available for lecturing, but for an additional payment. It is desired to minimize the use of such classrooms. This constraint can be represented using existing variables, by adding element (2) to the objective function. Given a classroom $r \in R$ and a timeslot $t \in T$, we denote by $w_{S_1,r,t}$ the “cost” of using classroom at timeslot t . Observe that the weight $w_{S_1,r,t}$ depends on the choice of the room r and the timeslot t . In the case of UP FAMNIT the timeslot t has no influence to the weight value at the time of this writing, although weights defined by both indices are more general and can easily be adopted in case that choice of timeslot becomes important for the cost of classroom. A constraint is represented by:

$$\sum_{r \in R} \sum_{t \in T_r} \sum_{m \in M} w_{S_1,r,t} \cdot x_{m,t,r}. \quad (2)$$

S_2) **Compact timetable.** Timetable compactness can have more forms. One of them is from the lecturers' point of view. This constraint is related to grouping of teaching obligations of teaching staff, since it is not desirable for one teacher to have some teaching hours in the morning and then again at the evening, with a long break in between. Since the number of teachers who teach just one course, that is, who are incident just with one meeting, is not too small, here it makes sense to refer just to teachers teaching more than one session. Denote set of corresponding teachers by L_+ . Then we introduce binary variable $z_{S_2,\ell,d}$, for every $(\ell, d) \in L_+ \times D$, where S_2 represents the type of constraint. As already mentioned, sets T_{AM} and T_{PM} represent morning and afternoon timeslots, respectively. Constraints representing S_2 are the following:

$$\begin{aligned} \sum_{m \in M_\ell} \sum_{t \in T_d \cap T_{PM}} \sum_{r \in R_m} x_{m,t,r} &\leq 0, \quad \forall d \in D, \forall \ell \in L_+, \\ \sum_{m \in M_\ell} \sum_{t \in T_d \cap T_{AM}} \sum_{r \in R_m} x_{m,t,r} &\leq 0, \quad \forall d \in D, \forall \ell \in L_+. \end{aligned}$$

Using variables $z_{S_2,\ell,d}$ we get:

$$\begin{aligned} \sum_{m \in M_\ell} \sum_{t \in T_d \cap T_{PM}} \sum_{r \in R_m} x_{m,t,r} &\leq B_1 z_{S_2,\ell,d}, \quad \forall d \in D, \forall \ell \in L_+, \\ \sum_{m \in M_\ell} \sum_{t \in T_d \cap T_{AM}} \sum_{r \in R_m} x_{m,t,r} &\geq B_2 (z_{S_2,\ell,d} - 1), \quad \forall d \in D, \forall \ell \in L_+. \end{aligned}$$

Constants B_1 and B_2 have to have sufficiently large values. In this case it is sufficient for them to be equal to the number of morning and afternoon timeslots in a day, respectively. Thus, we define B_1 and B_2 to have values τ_{PM} and τ_{AM} , respectively.

For every variable $z_{S_2,\ell,d}$ there is also a corresponding weight $w_{S_2,\ell,d}$ used in objective function:

$$\sum_{\ell \in L_G} \sum_{d \in D} w_{S_2,\ell,d} \cdot z_{S_2,\ell,d}. \quad (3)$$

S_3) **Requirements related to students.** As mentioned in the description of the teaching process at the institution, there are some Master's study programmes that are supposed to offer lectures just at afternoons timeslots. Meetings relevant to these programmes belong to the set M_{PM} , and number of such meetings scheduled for earlier timeslots should be minimised. Timeslots defined as afternoon timeslots are contained in the set $T_{PM} \subset T$. If it is not possible to put all meetings from M_{PM} in afternoon slots, there can be some measure that decides which of the requirements are preferred to be satisfied. For that reason we introduce weights $w_{S_3,m}$ for every meeting $m \in M_{PM}$, describing the importance of meeting. A greater weight means that the course is more desirable to be scheduled in the afternoon. The number of undesirable assignments is minimized by adding the following element to the objective function:

$$\sum_{m \in M_{PM}} \sum_{t \in T \setminus T_{PM}} \sum_{i \in H_m} w_{S_3,m} \cdot y_{m,t,i}. \quad (4)$$

Another constraint related to students' preferences concerns minimization of lectures scheduled at Friday afternoon. Most students go home during the weekend so such lectures are undesirable. Timeslots contained here can be determined by $T_5 \cap T_{PM}$. Since there are some additional properties that can influence the priority of scheduling lectures at described timeslots (e.g., the number of students attending some meeting), here we define the corresponding weights, $w_{S_3,m,t}$, for every meeting $m \in M$ and timeslot $t \in T_5 \cap T_{PM}$. These preferences can be modelled by adding the following sum to the objective function:

$$\sum_{m \in M} \sum_{t \in T_5 \cap T_{PM}} \sum_{r \in R_m} w_{S_3,m,t} \cdot x_{m,t,r}. \quad (5)$$

A third constraint in this group of constraints concerns upper bound on number of teaching hours related to one student group in a day. These are parameters, say $\rho_S(s)$, which define the numbers representing the desirable upper bounds. Here we define variables $z_{S_3,i}$, for each $i \in I_{S_3}$, with S_3 representing this constraint, and I_{S_3} being set of pairs $(s, d) \in S \times D$. It means that for every pair

representing a student group s and a day d there is a variable $z_{S_3,s,d}$, which determines if constraint of type S_3 is satisfied for (s,d) . If constraint is not satisfied, the variable gets value 1, and 0 otherwise. The constraint is originally represented by the inequality

$$\sum_{m \in M_s} \sum_{t \in T_d} \sum_{r \in R_m} x_{m,t,r} \leq \rho_S(s), \quad \forall s \in S, \forall d \in D.$$

From this we evaluate conditions for $z_{S_3,s,d}$ as follows:

$$\sum_{m \in M_s} \sum_{t \in T_d} \sum_{r \in R} x_{m,t,r} \leq \rho_S(s) + B_1 z_{S_3,s,d}, \quad \forall s \in S, \forall d \in D,$$

$$\sum_{m \in M_s} \sum_{t \in T_d} \sum_{r \in R} x_{m,t,r} \geq \rho_S(s) + 1 - B_2(1 - z_{S_3,s,t}), \quad \forall s \in S, \forall d \in D.$$

In the above equations, constants B_1 and B_2 can be determined in a few different ways. One possibility is to define them to have values $B_1 = \tau - \rho_S(s)$ and $B_2 = \rho + 1$, so that the corresponding constraint S_3 is satisfied for $s \in S$ and $d \in D$ whenever the variable $z_{S_3,s,d}$ has value 0, and violated whenever the variable $z_{S_3,s,d}$ has value 1. Given a variable $z_{S_3,s,d}$ we define a corresponding weight $w_{S_3,s,d}$, for normalization with other weights of the model. If there is no need for weights, they can be set to have value 1. Minimization of violations is represented by the sum (6), which is added to the objective function:

$$\sum_{d \in D} \sum_{s \in S} w_{S_3,s,d} \cdot z_{S_3,s,d}. \quad (6)$$

S₄) Requirements related to lecturers. The sets $T(\ell)$, for each $\ell \in L$, represents available timeslots for lecturer ℓ . Even if timeslots are contained in the set $T(\ell)$, there are some of them that might be preferred by the lecturer. For that reason we introduce a soft constraint representing a measure of lecturers' preferences with respect to the timeslots that are assigned to teaching hours. For each pair of lecturer $\ell \in L$ and timeslot $t \in T(\ell)$ we define a weight $w_{S_4,\ell,t}$ representing the measure of preferences. A modelled constraint has the form:

$$\sum_{\ell \in L} \sum_{m \in M(\ell)} \sum_{t \in T} \sum_{r \in R_m} w_{S_4,\ell,t} \cdot x_{m,t,r}.$$

4.5 The objective function

Putting together the sums described above, we can formulate the objective function of the ILP model as follows:

$$\begin{aligned} & \sum_{t \in T_r} \sum_{m \in M} \sum_{r \in R_m} w_{S_1,r,t} \cdot x_{m,t,r} + \sum_{\ell \in L_+} \sum_{d \in D} w_{S_2,\ell,d} \cdot z_{S_2,\ell,d} + \\ & \sum_{m \in M_{PM}} \sum_{t \in T \setminus T_{PM}} \sum_{i \in H_m} w_{S_3,m} \cdot y_{m,t,i} + \sum_{m \in M} \sum_{t \in T_5 \cap T_{PM}} \sum_{r \in R_m} w_{S_3,m,t} \cdot x_{m,t,r} + \\ & \sum_{d \in D} \sum_{s \in S} w_{S_3,s,d} \cdot z_{S_3,s,d} + \sum_{\ell \in L} \sum_{m \in M(\ell)} \sum_{t \in T} \sum_{r \in R_m} w_{S_4,\ell,t} \cdot x_{m,t,r}. \end{aligned}$$

5 Implementation of ILP model - ZIMPL

Zimpl is a little language to translate the mathematical model of a problem into a linear or nonlinear (mixed-) integer mathematical program expressed in .lp or .mps file format which can be read and (hopefully) solved by an LP or MIP solver. Zimpl is a command line program written in plain C and released under GNU LGPL. It has been tested to compile under Linux/Intel, Solaris, Tru64, HPUX, IRIX, AIX and MacOS-X. Probably it will compile and run wherever GMP is available. Zimpl has even been successfully compiled for Windows using MinGW and the GCC as a cross compiler and also directly using VisualStudio 2010. Zimpl has the complete documentation available at <https://zimpl.zib.de/download/zimpl.pdf>.

In the rest of this section we present the file model.zpl that can be evaluated using Zimpl. The file model.zpl contains all ingredients of the ILP model, and receives as input .txt files containing information about parameters of ILP, presented in Section 4.1. The result of evaluation of model.zpl using Zimpl will be .lp file model.lp that represents the integer linear program formulation in standard form and is accepted for any ILP Solver Program to (eventually) solve the proposed problem.

5.1 Input .txt files

Here we describe the .txt files that are used as input for Zimpl program. Zimpl reads each .txt file line by line, so every line represents one new data. In the following we list all files that are used in program, explain what do they represent, and give an example of one line belonging to the proposed file.

- **meetings.txt** - Every line contains Meeting ID and Meeting name, separated by comma. When reading the following example, we have two courses: Analysis 1 and Algebra 1, with IDs 1 and 2, respectively.

```
1, Analysis 1
2, Algebra 1
```

- **rooms.txt** -Every line contains Room ID and Room name, separated by comma. When reading the following example, we have two rooms: Big classroom 1 and Computer classroom 4, with IDs BC1 and CC4, respectively.

```
BC1, Big classroom 1
CC4, Computer classroom 4
```

- **lecturers.txt** - Every line contains Lecturer ID and Lecturer name, separated by comma. When reading the following example, we have two lecturers: Marc Anthony and Jennifer Lopez, with IDs MA and JL, respectively.

```
MA, Marc Anthony
JL, Jennifer Lopez
```

- **students.txt** - Every line contains Student group ID and Student group name, separated by comma. When reading the following example, we have two student groups: Mathematics 1 and Computer Science 2, with IDs MA1 and CS2, respectively.

```
MA1, Mathematics
CS2, Computer Science 2
```

- **afternoonMeetings.txt** - every line contains a single meeting ID. Every meeting with ID in that list should be scheduled in afternoon slots. In example below we have that the meeting with ID MA1 should be scheduled afternoon, while any other meeting not contained in this file does not have to satisfy that requirement.

```
MA1
```

- **noLecturerTimeslots.txt** - every line contains a pair of a lecturer ID, and a timeslot, such that the lecturer cannot teach in that timeslot. When reading the following example, we get that lecturer with ID MA (Marc Anthony) cannot teach on timeslots 5 and 6.

```
MA, 5
MA, 6
```

- `predefined.txt`- every line contains one event, for which the name of meeting, the classroom and the timeslot are known, and separated by comma. When reading the following example, we have two predefined meetings, where the first of them is meeting with ID 1, that is Analysis 1, and is scheduled to room CC4 (Computer classroom 4) and timeslot 13, and the second one is meeting with ID 2, that is Algebra 1, scheduled to room BC1(Big classroom 1) on timeslot 4.

```
1, CC4, 13
2, BC1, 4
```

- `meetingLecturers.txt` - every line contains a pair of a meeting ID, and a lecturer ID, separated by space, or by comma, such that the corresponding meeting and lecturer are incident, meaning that the lecturer is supposed to teach that meeting. When reading the following example, we get that lecturer with ID MA (Marc Anthony) teaches a course with id 1, and that the lecturer with id JL (Jennifer Lopez) teaches the course with ID 2.

```
1, MA
2, JL
```

- `meetingStudents.txt` - every line contains a pair of a meeting ID, and a student group ID, separated by space, or by comma, such that the corresponding meeting and student group are incident, that is, the student group should attend the meeting. When reading the following example, we get that the student groups with ID MA1 and CS2 are supposed to attend the course with ID 1, and that the student group with ID CS2 is supposed to attend the course with ID 2.

```
1, MA1
1, CS2
2, CS2
```

- `sections.txt` - every line contains a triple (separated by space or comma) consisting of a meeting ID, a number a , and a length i (that is, a number 1,2,3,4 or 5 - representing the number of hours), where a is the number representing how many sections of length i the corresponding meeting has. Every meeting has one line for each length, that is, every meeting has 5 lines. For example, if a meeting with ID 1 has two sections of length 3, then the corresponding lines are

```
1,0,1
1,0,2
1,2,3
1,0,4
1,0,5
```

- `roomsMeeting.txt` - every line contains a pair of a room ID, and a meeting ID, separated by space, or by comma, such that the corresponding meeting can be scheduled in that room. When reading the following example, we get that the meeting with ID 1 can be scheduled in room with ID BC1, and that the meeting with ID 2 can be scheduled in rooms with ID BC1 or CC4.

```
1, BC1
2, BC1
2, CC4
```

- `payableRooms.txt` - every line contains a pair of a room ID, and a number a , where a is a weight corresponding to a price of the classroom: if it is for free, then the price is 0, otherwise larger price gives the larger number a . In the example below we have that the BC1 is for free, while the classroom CC4 is payable. The weights should be determined depending on the current conditions.

```
BC1, 0
CC4, 2
```

5.2 ILP parameters

```
set Days := {1, 2, 3, 4, 5};
set Timeslots := {1..65};
```

```

set MorningTimeslots := {1, 2, 3, 4, 14, 15, 16, 17, 27, 28, 29, 30, 40, 41, 42, 43, 53, 54, 55, 56};
set NoonTimeslots := {5, 6, 7, 8, 18, 19, 20, 21, 31, 32, 33, 34, 44, 45, 46, 47, 57, 58, 59, 60};
set AfternoonTimeslots := {9, 10, 11, 12, 13, 22, 23, 24, 25, 26, 35, 36, 37, 38, 39, 48, 49, 50, 51, 52, 61, 62, 63, 64, 65};

set Types := {"Lec", "Se"};

set DaySlots := {< 1, 1 >, < 1, 2 >, < 1, 3 >, < 1, 4 >, < 1, 5 >, < 1, 6 >, < 1, 7 >, < 1, 8 >, < 1, 9 >,
< 1, 10 >, < 1, 11 >, < 1, 12 >, < 1, 13 >, < 2, 14 >, < 2, 15 >, < 2, 16 >, < 2, 17 >, < 2, 18 >, < 2, 19 >,
< 2, 20 >, < 2, 21 >, < 2, 22 >, < 2, 23 >, < 2, 24 >, < 2, 25 >, < 2, 26 >, < 3, 27 >, < 3, 28 >, < 3, 29 >,
< 3, 30 >, < 3, 31 >, < 3, 32 >, < 3, 33 >, < 3, 34 >, < 3, 35 >, < 3, 36 >, < 3, 37 >, < 3, 38 >, < 3, 39 >,
< 4, 40 >, < 4, 41 >, < 4, 42 >, < 4, 43 >, < 4, 44 >, < 4, 45 >, < 4, 46 >, < 4, 47 >, < 4, 48 >, < 4, 49 >,
< 4, 50 >, < 4, 51 >, < 4, 52 >, < 5, 53 >, < 5, 54 >, < 5, 55 >, < 5, 56 >, < 5, 57 >, < 5, 58 >, < 5, 59 >,
< 5, 60 >, < 5, 61 >, < 5, 62 >, < 5, 63 >, < 5, 64 >, < 5, 65 >};

set DurationLengths := {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
set Length := {1, 2, 3, 4, 5};
set Meetings := {read "meetings.txt" as " < 1s > "};
set Rooms := {read "rooms.txt" as " < 1s > "};
set Lecturers := {read "lecturers.txt" as " < 1s > "};
set Students := {read "students.txt" as " < 1s > "};
set AfternoonMeetings := {read "afternoonMeetings.txt" as " < 1s > "};
set lecturerTimeslots := {read "lecturerTimeslots.txt" as " < 1s, 2n > "};
set NotLecturerTimeslots := {read "noLecturerTimeslots.txt" as " < 1s, 2n > "};
set Predefined := {read "predefined.txt" as " < 1s, 2s, 3n > "};
set MetLen := Meetings * Length;
set FridayTimeslots := {53..65};
set MeetingsLecturers := {read "meetingLecturers.txt" as " < 1s, 2s > "};
set MeetingStudents := {read "meetingStudents.txt" as " < 1s, 2s > "};

param sections[Meetings * Length] := read "sections.txt" as " < 1s, 3n > 2n";
param roomsMeetings[Rooms * Meetings] := read "roomsMeeting.txt" as " < 1s, 2s > 3n";

set indexForX := Meetings * Timeslots * Rooms;
set indexForY := Meetings * Timeslots;
set Attr := Rooms + Lecturers;

param payableRooms[Rooms] := read "payableRooms.txt" as " < 1s > 2n";

set MeetingsSections := {< m, s > in Meetings * Length with sections[m, s] > 0};
var x[indexForX] binary;
var y[< m, s, t > in MeetingsSections * Timeslots] binary;

```

5.3 Objective function

```
minimize cost : sum < m, t, r > inAfternoonMeetings * MorningTimeslots * Rooms : 2 * x[m, t, r] +
               sum < m, t, r > inAfternoonMeetings * NoonTimeslots * Rooms : 2 * x[m, t, r] +
               sum < m, t, r > inMeetings * Timeslots * Rooms : x[m, t, r] * payableRooms[r] +
               sum < m, t, r > inMeetings * FridayTimeslots * Rooms : x[m, t, r];
```

5.4 Constraints

Here we write constraints explained in Section 4.3.

```
subto Aone :
    forall <l> in Lecturers do
        sum <m,t,r> in Meetings cross Timeslots cross Rooms with <m,l> in
MeetingsLecturers and <l,t> in NotLecturerTimeslots : x[m,t,r] <=0;
|
subto Atwo :
    forall <r> in Rooms do
        sum <m,t> in Meetings cross Timeslots with <r,t> in RoomsTimeslots : x[m,t,r] <=0;

subto Athree :
    forall <m> in Meetings do
        sum <t,r> in Timeslots cross Rooms with roomsMeetings[r,m] == 0 : x[m,t,r] == 0;

subto Bone :
    forall <s,t> in Students cross Timeslots do
        sum <m,r> in Meetings cross Rooms with <m,s> in MeetingStudents: x[m,t,r] <= 1;

subto Btwo :
    forall <l,t> in Lecturers * Timeslots do
        sum <m,r> in Meetings cross Rooms with <m,l> in MeetingsLecturers : x[m,t,r] <= 1;

subto Bthree :
    forall <r,t> in Rooms * Timeslots do
        sum <m> in Meetings : x[m,t,r] <=1;

subto Cone :
    forall <m> in Meetings do
        sum <t,r> in Timeslots * Rooms : x[m,t,r] - sum <s> in Length with <m,s> in
MeetingsSections : s*sections[m,s]==0;

subto Ctwo :
    forall <m, s, d> in MeetingsSections * Days do
        sum <t> in Timeslots with <d, t> in DaySlots and ( t - 1 ) mod 13 >= 14 - s : y[m,s,t] == 0;

subto Cthree :
    forall <m, d> in Meetings * Days do
        sum <s, t> in Length * Timeslots with <d, t> in DaySlots and <m, s> in MeetingsSections: y[m,s,t] <= 1;

subto Cfour :
    forall <m,s> in MeetingsSections do
        sum <t> in Timeslots : y[m,s,t] == sections[m,s];

subto Cfive :
    forall <m,s,d> in MeetingsSections * Days do
        sum <r,t> in Rooms*Timeslots with <d,t> in DaySlots: x[m, t, r ] >= s * sum <t> in Timeslots
|with <d,t> in DaySlots: y[m,s,t];

subto Csix :
    forall <m, s, t, i> in MeetingsSections * Timeslots * DurationLengths with ( t - 1 ) mod 13 < 14 - s and
i <= s do
        y[m,s,t] <= sum <r> in Rooms : x[m, t+i-1, r ];

subto Cseven :
    forall <m, t, r1, r2> in Meetings cross Timeslots cross Rooms cross Rooms with r1 != r2 and (t-1) mod
13 < 12 do
        x[m, t, r1] + x[m, t+1, r2 ] <= 1;
```

```

subto D :
    forall <m,r,t> in Predefined: x[m,t,r]==1;

subto Eone:
    forall <l,d> in Lecturers cross Days do
        sum <m,t,r> in Meetings cross Timeslots cross Rooms with <d,t> in DaySlots and <m,l> in
        MeetingsLecturers : x[m,t,r]<=9;

```

5.5 Zimpl evaluation

Once the model.zpl file is prepared as explained in previous section, we navigate in cmd to directory containing the file model.zpl and relevant .txt files described in previous section, and run the following command:

```
zimpl model.zpl
```

The result is a file model.lp that can be used to evaluate ILP using Gurobi Optimizer.

```

nevena@wanker:~$ zimpl model.zpl
*****
* Zuse Institute Mathematical Programming Language *
* Release 3.3.9 Copyright (C)2018 by Thorsten Koch *
*****
* This is free software and you are welcome to *
* redistribute it under certain conditions *
* ZIMPL comes with ABSOLUTELY NO WARRANTY *
*****

Reading model.zpl
Reading lecturers.txt
Reading noLecturerTimeslots.txt
Reading predefined.txt
Reading meetingLecturers.txt
Reading meetingStudents.txt
Reading sections.txt
Reading payableRooms.txt
Instructions evaluated: 27261106
Name: model.zpl Variables: 13455 Constraints: 82859 Non Zeros: 293736
writing [model.lp]
writing [model.tbl]
nevena@wanker:~$

```

6 Evaluation of ILP model - Gurobi Optimizer

Once the model.lp file was produced, in order to solve the problem we run the command

```
gurobi_cl model.lp
```

Additionally, we can save the resulting .sol file produced by Gurobi that contains the vector of variables. Assume we want to name the resulting file model.sol:

```
gurobi_cl ResultFile=model.sol model.lp
```

The resulting file model.sol is used in the back-end of our timetable visualization app in order to display the events.

```
nevena@wanker:~$ gurobi_cl ResultFile=model.sol model.lp
Using license file /home/nevena/gurobi.lic
Set parameter LogFile to value gurobi.log
Academic license - for non-commercial use only

Gurobi optimizer version 9.0.2 build v9.0.2rc0 (linux64)
Copyright (c) 2020, Gurobi Optimization, LLC

Read LP format model from file model.lp
Reading time = 0.43 seconds
cost: 82859 rows, 13455 columns, 293736 nonzeros
Optimize a model with 82859 rows, 13455 columns and 293736 nonzeros
Model fingerprint: 0xac52b060
Variable types: 0 continuous, 13455 integer (0 binary)
Coefficient statistics:
  Matrix range [1e+00, 4e+00]
  Objective range [1e+00, 8e+00]
  Bounds range [1e+00, 1e+00]
  RHS range [1e+00, 2e+01]
Presolve removed 68543 rows and 3225 columns
Presolve time: 3.38s
Presolved: 14316 rows, 10230 columns, 170718 nonzeros
Variable types: 0 continuous, 10230 integer (10230 binary)

Deterministic concurrent LP optimizer: primal and dual simplex
showing first log only...

Concurrent spin time: 0.00s

Solved with primal simplex

Root relaxation: objective 0.000000e+00, 1455 iterations, 0.58 seconds
Total elapsed time = 5.01s

  Nodes      |      Current Node      |      Objective Bounds      |      Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent  BestBd   Gap   | It/Node Time
-----
H   0   0   0.000000  0  28   -          0.000000  -       -   | -     5s
H   0   0   30.00000000  0  28   30.00000000  0.000000  100%  -   | -     5s
H   0   0   24.00000000  0  28   24.00000000  0.000000  100%  -   | -     5s
H   0   0   4.00000000  0  28   4.00000000  0.000000  100%  -   | -     6s
   0   0   4.000000  0  60   4.000000  0.000000  100%  -   | -     6s
   0   0   0.000000  0  42   4.000000  0.000000  100%  -   | -     8s
   0   0   0.000000  0  40   4.000000  0.000000  100%  -   | -     9s
   0   0   0.000000  0  23   4.000000  0.000000  100%  -   | -    10s
H   0   0   0.000000  0  39   0.000000  0.000000  0.00%  -   | -    11s
   0   0   0.000000  0  39   0.000000  0.000000  0.00%  -   | -    11s

Cutting planes:
  MIR: 9
  StrongCG: 1
  Zero half: 9

Explored 1 nodes (7601 simplex iterations) in 11.70 seconds
Thread count was 32 (of 80 available processors)

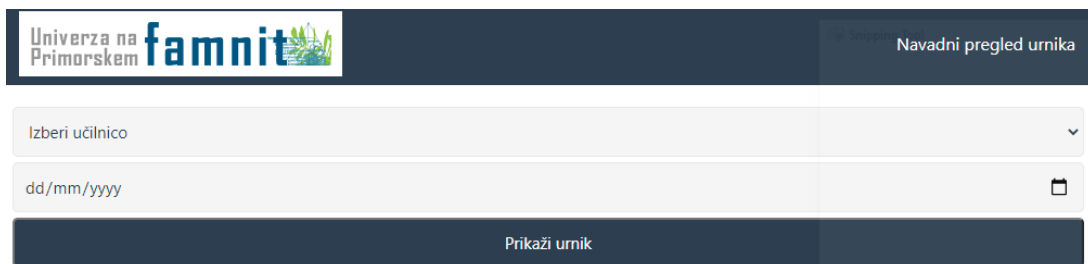
Solution count 4: 0 4 24 30

Optimal solution found (tolerance 1.00e-04)
Best objective 0.000000000000e+00, best bound 0.000000000000e+00, gap 0.0000%

wrote result file 'model.sol'
```

7 Timetable app - visualisation

The timetable app runs on local server in development mode, while for production it is supposed to run on web server. The default view of app, stored in file `index.html` displays the select form where user can select a study program, a course, a teacher, and a date, and after submitting the form, the corresponding timetable shows. The other view, stored in `rooms.html` displays the select form where a user can select a room, and a date, and after submitting a form gets the corresponding timetable displayed. For both views, see figures below. The features of the app are explained in next section.

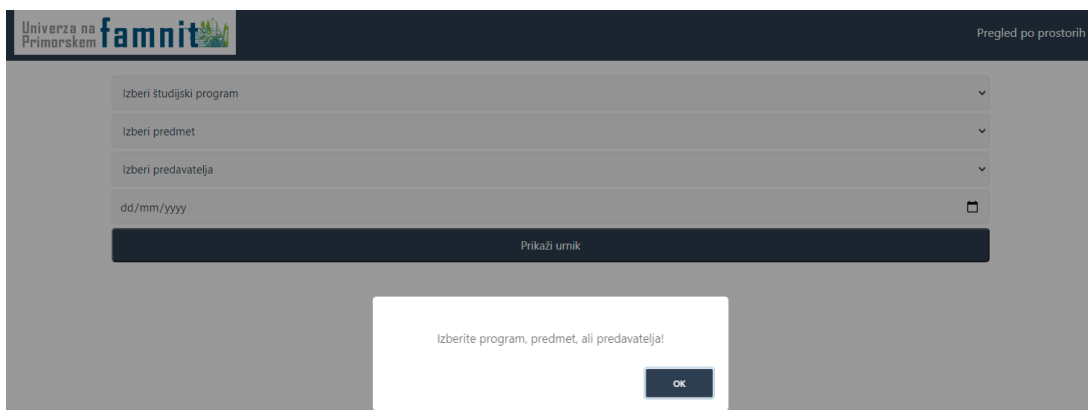


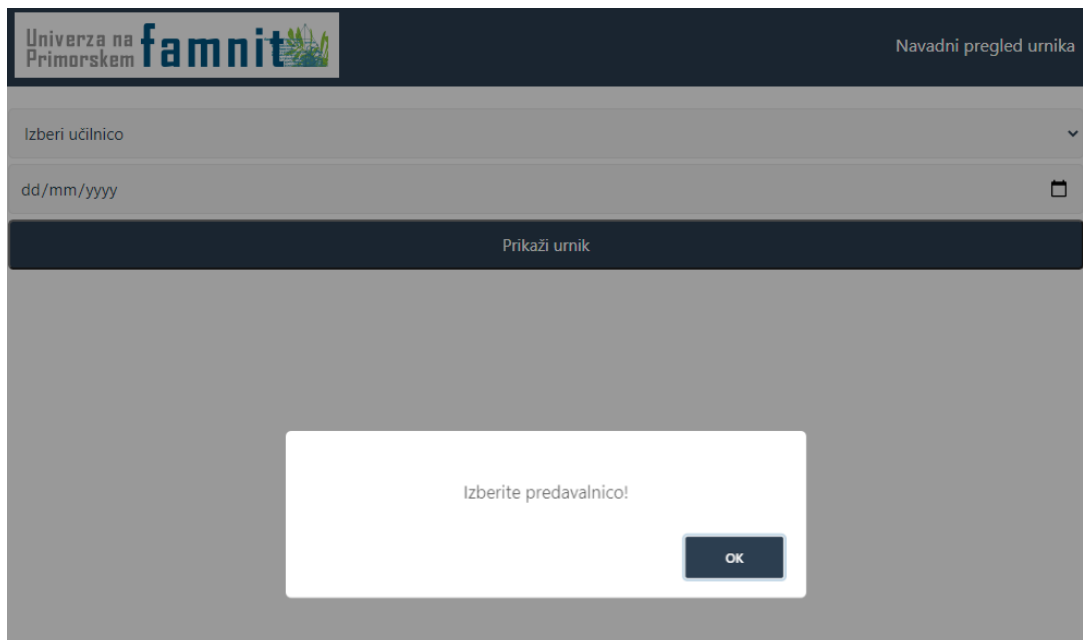
7.1 Features

The aim of the app is to display the timetable produced by Gurobi Optimizer.

7.1.1 Nothing was selected

If it happens that a user does not make any selection in select form, and wants to display a timetable, then he gets the alert box with information that no data was selected (figures below).



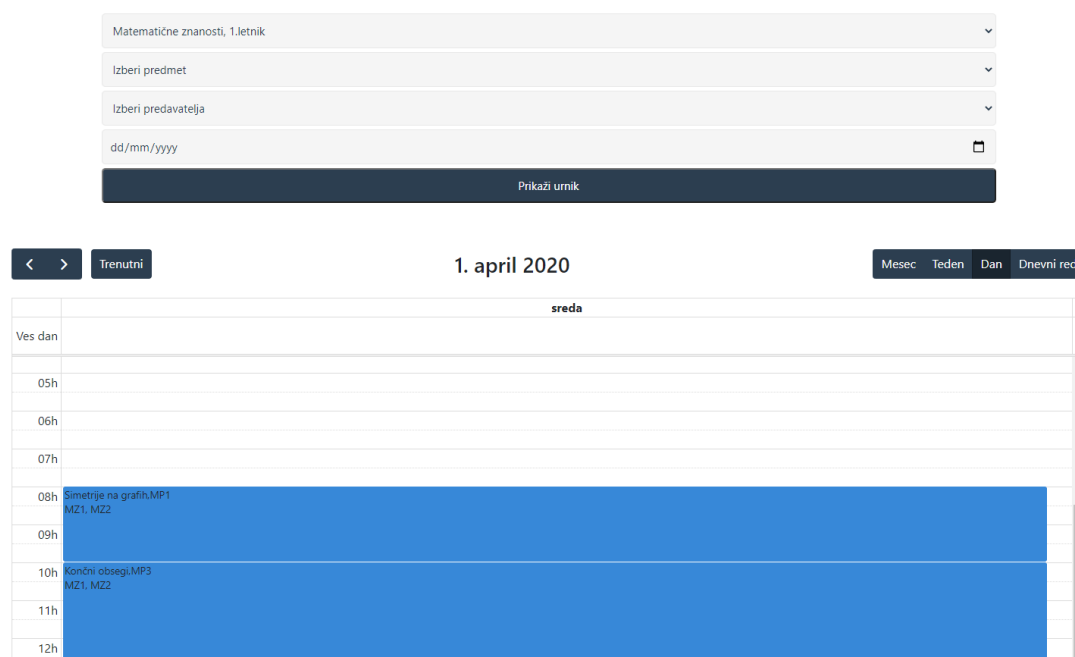


7.1.2 Display relevant events

Once the user submits the select-form, selected data are sent to server, and among all scheduled events selects which events are to be displayed. When clicking on submit button the calendar is displayed, and there we can select among the month, week, or day view. Also, we can show the so-called task list, that is, a list of all events taking place in a current week. If we select a date in the select form, then the calendar is displayed for the selected date. Otherwise, if we don't select any date, the calendar is displayed for the current day (=“today”). In any calendar view, we can navigate to previous, or next day(week, month), or to today, so the calendar is interactive.

7.1.3 Day view

If the calendar is displayed in a day-view (figure below), then we can see which day is selected, and for each event (meeting) we see the course name, the groups of students that are supposed to attend that event, and the classroom. Also, we can see a starting and end-time of a meeting.



7.1.4 Week view

If the calendar is displayed in a week-view (figure below), then we can see a whole selected week, and for each day we see all the events that take place on that day. For each event (meeting) we see the course name, the groups of students that are supposed to attend that event, and the classroom. Also, we can see a starting and end-time of a meeting.

	pon. 30. 3.	tor. 31. 3.	sre. 1. 4.	čet. 2. 4.	pet. 3. 4.	sob. 4. 4.	ned. 5. 4.
Ves dan							
05h							
06h							
07h							
08h			Simetrije na grafih.MP1 MZ1, MZ2				
09h							
10h	Matematični seminar.MP1 MZ1, MZ2	Algebra.VP2 MZ1	Končni obsegi.MP3 MZ1, MZ2				
11h	Diskretna matematika.MP4 MZ1, MZ2, PZ1						

7.1.5 Month view

If the calendar is displayed in a month-view (figure below), then we can see which month is selected, and we can see the whole month at once, where each week is displayed in one line. For each day of a week a list of relevant events is displayed, and for each event we see the course name. If we want to see more details for each event, then we should select a day view or a week view.

pon.	tor.	sre.	čet.	pet.	sob.	ned.
30. Matematični seminar Diskretna matematika	31. Algebra Linearno programiranje	1. Simetrije na grafih Končni obsegi	2. Numerika	3.	4.	5.
6. Matematični seminar Diskretna matematika	7. Algebra Linearno programiranje	8. Simetrije na grafih Končni obsegi	9. Numerika	10.	11.	12.

7.1.6 Tasks view

If the calendar is displayed in a task-view (figure below), then we can see which month is selected, and we can see the list of events taking place in a selected week. We don't see the grid view day by day, but a list

of events for each day, one by one. For each event we see the starting and ending time, and the name of the meeting. Additionally, we can display the relevant groups of students, or classroom, if we prefer.

Univerza na Primorskem tamnit Pregled po prostorih

Matematične znanosti, 1.letnik

Izberi predmet

Izberi predavatelja

dd/mm/yyyy

Prikaži urnik

< > Trenutni 30. mar. – 5. apr. 2020 Mesec Teden Dan Dnevni red

ponedeljek	30. marec 2020
10:00 - 11:00	• Matematični seminar
11:00 - 15:00	• Diskretna matematika
torek	31. marec 2020
10:00 - 14:00	• Algebra
19:00 - 21:00	• Linearno programiranje
sreda	1. april 2020
08:00 GMT+2 - 10:00	• Simetrije na grafih
10:00 - 13:00	• Končni obsegi
četrtek	2. april 2020
15:00 - 18:00	• Numerika

7.2 Full Calendar

Full Calendar (www.fullcalendar.io) is an open-source JavaScript calendar. It has connectors for React, Vue, and Angular, and provides high-quality TypeScript definitions. All code is available on Github, and is easy to implement in existing project. In documentation on webpage of Full Calendar we can find few ways for installing it, the easiest two are using npm, or just by downloading the ZIP archive and using the script tag. In this project we used the second option, so the calendar was created using the script tag. In the definition of calendar we set the variable `events` that is a JSON array, and each element of it is a json object representing one event. Once the all events to display are defined, we use the function `calendar.render()` that creates a calendar. All existing methods for FullCalendar are explained in the documentation of FullCalendar, and if there is any need to use some of them, one can find it there, so we will not repeat the FullCalendar documentation here.

7.3 Methods

Main methods are described below.

`getAllEvents()` - a method that reads the `.sol` file (resulting file from Gurobi) and produces all events for which the variable x has a value 1. Method returns the JSON array, where each element of array has the following attributes: `ID`, `title`, `day`, `start`, `end`, and these attributes represent the meeting ID, meeting title, day in a week when meeting is scheduled, and starting and ending time of a meeting.

`sendData(data)` - a method that receives as input the data from select form in `index.html` and among all events selects those that are supposed to be visible in timetable, based on submitted data. The output of a method is object similar as in previous method.

`sendRoomData(data)` - a method that receives as input the data from select form in `rooms.html` and among all events selects those that are supposed to be visible in timetable, based on submitted data. The output of a method is object similar as in previous method.

`showCalendar(data)` - a method that executes on click of a submit button in the form. The method receives as input the data from select form and uses one of previous two methods (depending on the view) and sets the relevant events to be displayed. In the last step of execution method executes the line `calendar.render()` so that is displays the calendar once all parameters are set.