Time Scheduling System (v2)
Oct 21, 2023

1 Introduction

Course scheduling is a very complex process. It has three main stages:

1. The course scheduling committee creates a questionnaire for instructors to fill out their preferences. The filled out questionnaire will be used in Stage 2 and become the source of the **InstructPref** file.
2. Based on the results of the questionnaire and other factors, and after checking with instructors, the committee decides which course is offered in which quarter and is taught by which instructor. The output of this stage is the **CourseInstructor** file.
3. The committee runs the scheduling system to determine the time for each course.

For the third stage, we develop a time scheduling system to do time assignment automatically.

1.1 Overivew of the Time Scheduling System

The goal of the time scheduling system is to choose time slots for all the courses in a quarter, while considering various constraints and preferences. Here, we treat the time scheduling task as an integer linear program (ILP) task, and thus the system has two main components:

- An ILP solver: we will use an existing ILP package, PuLP, to solve the ILP problem created by the Python code.
- The Python code, which works as follows:
  1. It takes the input documents described in Section 2 to set up an ILP problem.
  2. It calls the ILP solver to find an optimal solution for the ILP problem.
  3. It processes the optimal solution to generate a time schedule file (Section 3.1), a heat map (Section 3.2) and a log file (Section 3.3).

1.2 To run the system

To run the system, one needs to create the input files described in Section 2 and type the following command on patas:

time-schedule.py config-file

The input files and the output directory are specified in the config-file, and the system will store the output files under the output directory.

For the rest of the documents, we describe the input and the output of our system in Section 2-3, the design of the ILP problem in Appendix A, and the pseudocode for creating the ILP problen in Appendix B.

2 The input to the algorithm

In order to run the scheduling system, the following files should be provided (by the scheduling committee):

1. A **config** file that stores (a) constants and options so that they are not hardcoded in the python code, and (b) the input file names and the output directory (Section 2.1).
2. The **CourseInfo** file which stores the info about all the courses offered by the linguistics department (Section 2.2).
3. The **ConflictCourses** file stores courses that should not be offered at the same time slots (Section 2.3).
4. The **InstructorPref** file w.r.t. time of their courses (Sect 2.4). The file is based on the instructor questionnaire, and is the same for the three quarters in the same academic year.
5. The **CourseInstructor** file is the list of courses offered in a particular and their instructors (Section 2.5). The file is the result of the first two stages of course scheduling.

For those files:

- The 1st file is likely to remain the same except for the file/directory names unless UW's block scheduling and 10% rule has changed.
- The 2nd and 3rd files include all the courses offered by the ling dept, not just the ones from the current year or quarter. They are likely to remain the same unless something about a course change.
- The 4th file is based on the questionnaire from Stage 1 of the scheduling process and will change each year.
- The last file is based on the result of Stage 2 of the scheduling process and is provided for each quarter.

2.1 The config file

The config file is the only argument of the python command. It has three parts. The first part corresponds to the UW block scheduling policy and the 10% rule.[1]

```
# The standard instructional day is from 8:30 am to 6:20 pm.
InstructDayStartsAt = 8:30
InstructDayEndsAt = 18:20

# block scheduling affecting courses ending before 14:30
BlockSchedulingStartsAt = 8:30
BlockSchedulingEndsAt = 14:30

# 10% rule applies to any given time per week (M-F numbers are combined).
# Right now, the percentage is 12%
10PercRuleStartsAt = 8:30
10PercRuleEndsAt = 18:20
```

[1]The policy is at https://registrar.washington.edu/classrooms/learning-spaces-faq/

```
RulePercentage = 0.12

# block scheduling requires the following starting time:
50-min-class-start-time = 8:30 9:30 10:30 11:30 12:30 13:30
80-min-class-start-time = 8:30 10:00 11:30 13:00
110-min-class-start-time = 8:30 10:30 12:30 13:00
170-min-class-start-time = 8:30 11:30
```

The second part of the file specifies the constants and options we choose when forming the ILP problem.

```
# if the first variable is 1, we strictly follow the block policy (i.e.,  we set the X
#   decision variables to 0 for any slots that do not meet the policy)
# if it is 0, then we do not set X, but the slot in CW with a penalty.

must-follow-block-policy = 1

penalty-for-violating-block-policy = -10
```

The third part specifies the input files and the output directory.

```
# input files for running the code
CourseInfo      = "xxx/CourseInfo"
ConflictCourse  = "xxx/ConflictCourse"
InstructorPref  = "xxx/2024-2025/InstructorPref"
CourseInstructor = "xxx/2024-2025/fall2024/CourseInstructor"

# the output will be stored under this directory:
outputDir = "xxx/2024-2025/fall2024/output/"
```

Among the three parts, the first two parts will remain the same unless the UW policy changes or our treatment of the policy changes (i.e., whether we want to treat it as a hard or a soft constraint). The third part will change for each quarter.

2.2 CourseInfo: all the courses offered by the dept in any year

CourseInfo is a list of all the courses offered by the department, not just the ones offered in a particular quarter or year. The file is likely to remain the same over the years, and needs to be changed only when a course is added or removed or changed.

Each course is represented as a tuple (course name, leng per session, num of sessions per week, large-class, 10-percent-rule-exempted, is-a-TA-session, AllSessionsMustBeOnSameDay, MustOnDay):

- course name: it can be a course number + session id, e.g., "575C". The session id is needed only if different sessions correspond to different courses or TA sessions.
- leng per session: the length of a session in minutes.
- num of sessions per week: how many times a class meets per week
- large-class: it is 1 if the class has 250+ students. Large classes that meet twice per week must meet on Tues and Thurs.

- 10-percent-rule-exempted: some courses (e.g., CL courses) are exempted from the 10% rule.
- is-a-TA-session: TA sessions can be different from a regular course w.r.t. scheduling; e.g., the instructor of an TA session might be unknown, and the TA sessions must all be on the same day.
- AllSessionsMustBeOnSameDay: it is 1 if all the sessions must be on the same day. This can be true for some TA sessions.
- MustOnDay: If all sessions must be on the same day, this field specify which day it is (e.g., "M" for Monday).

Here is an example file:

```
200  50  3  1  0  0 0 - # ling570 is 80-min long, 3 sessions/week,
                        # large class, not exempted from 10% rule.
                        # it is not a TA session
200A 60  4  0  0  1 1 T # an TA session that must meet on Tues only.
200B 60  5  0  0  1 1 R # an TA session that must meet on Thurs only
...
570  80  2  0  1  0 0 - # CLMS courses are exempted from the rule
575C 140 1  0  1  0 0 -
```

2.3 ConflictCourse: courses that should not be taught at the same time

In this file, each line is a list of courses whose time slots should not overlap. For instance, assume that 570 and 571 are in conflict and each is 80-min long. Thus if 570 is at 1-2:20pm, then 571 cannot start between noon-2pm.

The conflict may be due to several reasons:

- Many students plan to take both courses together: e.g., 566 and 570, or 200 and its TA sessions. This type of conflict tends to remain the same over the time.
- The same instructor is teaching both courses. This type does not need to be included in this file as our algorithm can infer that from CourseInstructor.
- The instructor or the TA of one course plans to take the other course. Include this kind of conflict only if necessary. For instance, if LING200 has several TA sessions and one TA is teaching session 200A and plans to take ling500, no need to add (200A, 500) to the list if the TA can switch with another TA who is not planning to take LING 500.

Here is an example:

```
570 571 566 550  # many CLMS students are taking multiple ones from this list
200 200AA 200AB 200AC # 200 and its TA sessions
```

2.4 InstructorPref: the instructors' preferences

InstructorPref is a list of instructors' preferences. Each line represents an instructor's preference as a tuple (instructor name, preferred days, preferred start time, preferred end time, same day or not). All but the first field are optional (represented as a dash). If an instructor teaches multiple courses in a quarter, the last field indicates whether s/he prefers to teach all the courses on the same day.

Here is an example:

```
naja    -   9:30  14  1 # no teaching after 2:30, Same day(s) for my courses.
fei     TR 10:00  -   - # prefer to teach on T or R, starting after 10:30am
qi      -   9:00  -   1 # prefer same day teaching and no early morning class
emily   TRF -     -   1 # prefer to avoid MW, and stack classes on the same day
richard TR  -     -   1 # like my classes to be on the same day and on T Th
```

2.5 CourseInstructor: courses offered in a quarter

CourseInstructor is a list of courses offered in that quarter, with instructor and other information. Each course is represented as a tuple (course name, instructor name, must on days, must start time, must end time).

- For large courses, their time slots might have been selected already, before running this system. Their time slots are specified in the last three fields.
- For seminars, they must start at 3:30pm, so must-start-time should be 15:30; must-on-days can be unspecified (meaning any day is fine), which is represented as a dash.
- For the TA session, the instructor's name might be unavailable, represented as a dash.

Here is an example:

```
200  laura  MWF 10:30 11:20  # ling200 is scheduled already
200AA -                      # the TA session's instructor is unknown
200AB -
...
570  fei
575C gina
```

CourseInstructor is created for each quarter for which we will run the course scheduling system. It is the result of Stage 2 of the course scheduling process.

3 The output files produced by the algorithm

3.1 TimeSchedule: schedule for the quarter

TimeSchedule is a list of courses with scheduled time. Each line has the format (course name, instructor name, days, startTime, endTime, length, blockPolicy, instructor preference).

Here is an example:

```
200  laura   MWF 11:30 12:20 50  y n  # ling200 is at 11:30-12:20 on MWF
                                      # meet block policy, but not instructor pref
200AA -      M   15:30 16:30 60  - -
200AB -      T   13:30 14:30 60  y -
...
570  fei     TR  13:00 14:20 80  y y
575C gina    R   15:30 17:50 140 - y
```

More information about the fields:

- The first two elements are the same as in CourseInstructor.
- days and startTime come from Matrix X
- length and endTime come from CourseInfo
- blockPolicy indicates whether the block scheduling policy is satisfied: "y" for yes, "n" for no, or "-" for irrelevant (e.g., when the class starts after 2:30pm). The value is derived by comparing the cells in Matrix X and CW.
- The last field indicates whether the instructor's preference is satisfied: "y" for yes, "n" for no, or "-" for irrelevant (e.g., the instructor did not specify the preference). The value is derived by comparing the cells in Matrix X and IW.

3.2 The Heat Map for the 10% rule

Our system will produce a heat map, similar to the one at the UW policy page.[2] The heat map will show how well we follow the 10% rule. Our map is created from Matrix Y (see Appendix A.5).

Here is what the heat map looks like:

```
       M    T   W   R   F  Hourly total    Hourly Target
8:30   1.5  1   1   1   0     4.5              10
9:30   5    3.5 4   3.5 1.5  17.5              10
...
17:30  0    3   0   3   0     6                10
```

3.3 The log file produced by the Python code

This file should contain important results from running the code, such as

- TotalCoursesNum: including exempted courses
- ExemptedCourses: the list of exempted courses
- TotalHours: the total number of hours of the nonexempted courses
- objective function's value
- If needed, the system can also dump out Matrix X, Y, CW, IW, etc.

[2]https://registrar.washington.edu/classrooms/learning-spaces-faq/

A Treating time scheduling as an ILP task

To form the ILP problem, we define four matrices X, Y, CW, and IW:

- All of them are three dimensional arrays of size C*D*T, where C is the number of courses, D is 5 (for 5 days), and T is 20 (the number of time slots per day).
- The cells of X and Y, represented as $x_{c,d,t}$ and $y_{c,d,t}$ are decision variables; but Y can be determined by X.
- CW and IW are the weight matrices, CW is for course preferences and IW is for instructor preferences.
- The objective function is: $\sum_{c,d,t}(\lambda_1 * CW_{c,d,t} + \lambda_2 * IW_{c,d,t}) * x_{c,d,t}$.
- The constraints are defined in Appendix B.8.

A.1 Course, day, and time slot

In the system, all names will be mapped to integer ids:

- Course: each course has a unique course name (e.g., 575C), which will be mapped to unique course id (e.g., 0 for the 1st course in CourseInstructor file, 1 for the 2nd course, and so on).
- Day: each day has a unique name (e.g., "M" for Monday), and a unique day id (e.g., 0 for Monday).
- Time slot: The instructional day (8:30am-6:20pm) is broken into twenty 30-min time slots, each slot has a unique slot name (e.g., 8:30-9:00) and a unique slot id t, e.g., 8:30-9:00 is slot 0, 9:00-9:30am is slot 1, and so on.
- In this document, we will use c, d, and t to refer to course id, day id, and time slot id, representively. They are the indices for the four matrices, ddescribed below.

For the rest of the Appendix, we describe the matrices and other data structures.

A.2 Course info

We use an array, **CourseInfo**, to store course information. This array is set according to **CourseInfo** and **CourseInstructor** files. Its element, CourseInfo[courseId], is a structure with the following fields:

```
### the following info comes from the CourseInstructor file
instructorId: int    ## the id of the instructor, -1 is unspecified
mustOnDays: intArray ## the list of days for the course, [] if unspecified.
mustStartSlot: int   ## the slot id for the must-start-time
mustEndSlot: int     ## the class must ends by this time slot
                     ## e.g., if must-end-time is 11:20, the class must end
                     ## by the time slot for 11-11:30

### the following info comes from the CourseInfo file
lengPerSession: int  ## length of a session in minutes
sessionsPerWeek: int ## how often a class meets per week
largeClass: int      ## 1 (if it is a large class) or 0
```

```
exempted: int         ## 1 (if exempted from 10\% rule) or 0
is-TA-session: int    ## 1 (if it is an TA session) or 0
mustBeOnSameDay: int  ## 1 (if all sessions have to be on the same day) or 0
mustOnWhichDay: int   ## the day id or -1 (if unspecified)

slotNum: int          ## it is ceiling(lengPerSession / 30)
```

A.3 Matrix X which marks the starting times of courses

Matrix X represents the start time of a course; that is, $X[c][d][t] = 1$ means that class c starts at time t on Day d; 0 otherwise. The matrix includes all the 'independent" decision variables as Matrix Y depends on Matrix X.

Some cells in X are set to 0 to account for hard constraints:

- If **must-follow-block-policy** is set to 1 in the config file, all the time slots that do not meet the block scheduling policy are set to 0 (e.g., 9am, 10am, ..., 2pm for a 50-min course).
- If the **CourseInstructor** file states that some courses must meet on a particular day or time, then we set other time slots to 0.

A.4 Matrix Y which marks the durations of courses

Matrix Y marks the duration of each course. That is, $Y_{c,d,t}$ is 1 if the time span of the course includes this time slot. For instance, if a 80-min course starts at Monday 8:30, then 8:30, 9:00, and 9:30 slots are all set to 1 in Y, but only the 8:30 slot is set to 1 in X.

We introduce Matrix Y in order to handle conflictCourse constraints and the 10%-rule in the ILP task. we treat $Y_{c,d,t}$ as decision variables too. But once matrix X is set, since the length of each course is known, matrix Y is set too. In order to keep X and Y consistent, we add the following constraints:

```
for each c:
    slotNum = CourseInfo[c].slotNum
    $\sum_{d,t} X_{c,d,t} * slotNum = \sum_{d,t} Y_{c,d,t}$
    for each d and each t:
        for j in range(t, min(slotNumPerDay, t+slotNum)):
            $Y_{c,d,j} \geq X_{c,d,t}$
```

The last line says that, if $X_{c,d,t}$ is 1, then $Y_{c,d,j}$ must be 1 too when j=t, t+1, ..., t + slotNum - 1. SlotNumPerDay is the total number of slots per day, which is 20 if the instruction day is from 8:30am to 6:30pm.

A.5 Matrix CW which stores the weights based on course requirements

Matrix CW stores preferences based on the course requirements. Currently, the preferences only comes from the UW block scheduling policy. Later, other kinds of preferences (if any) can be easily added to the matrix. The value of $CW_{c,d,t}$ is:

- 1 if t is one of the slots allowed by the policy (e.g., 8:30am for a 50-min class)
- **penalty-for-violating-block-policy** if t violates the policy (e.g., 9am for a 60-min class).
- 0 if t is outside the policy window (e.g., t is after 2:30pm)

A.6 Matrix IW which stores the instructors' preference

In the **InstructorPref** file, an instructor can specify PrefDays, PrefStartTime and PrefEndTime. From the **CourseInstructor** file, we know what courses an instructor is teaching in that quarter. Thus, the preference is passed from the instructors to the courses that they teach.

The value of $IW_{c,d,t}$ is:

- **1/CourseInfo[c].sessionsPerWeek** if d is one of the PrefDays and t falls inside [PrefStartTime, PrefEndTime] (i.e., t is no earlier than PrefStartTime and class c starting at t will end no later than PrefEndTime[3]).[4]
- **0** otherwise (i.e., the instructor of the course does not indicate any preference)

B Implementation of the System

The system reads the input files (Section 2) and produces the output files (Section 3). This section shows the main steps of the system.

B.1 Read the config file

From the config file, set the values of the following variables:

- Based on **InstructDayStartsAt** and **InstructDayEndsAt**, set **TimeSlotName2Id**, **TimeSlotId2Name**, and **SlotNumPerDay**. e.g., if the instruction day starts at 8:30 and ends at 18:20, the slot id for 8:30 is 0, and SlotNumPerDay is 20.
- Map **BlockStartsAt** and **BlockEndsAt** to their slot ids and store those slot ids in variables of the same names. Do the same for **10PercRuleStartsAt** and **10PercRuleEndsAt**. Store the value for **RulePercentage**.
- Map the slot names in **50-min-class-start-time** to slot ids, and store the slot ids into a list. Do the same for **80-min-class-start-time**, etc.
- Store the values for **meet-follow-block-policy** and **penalty-for-violating-block-policy**.
- Store the names of the four input files and the output dir.

B.2 Read the CourseInstructor file

This file lists courses that are offered in this quarter. We use the info to set **Instructor2Courses**, the first four attributes of **CourseInfo**, the map between course/instructor names and their ids, etc:

- Each line has the format: "course-name instructor-name must-on-days must-start-time must-end-time". The last three fields can be empty.
- Create a hash table **CourseName2Id** and an array **CourseId2Name** to map between course names and course ids. Do the same for instructor names and their ids.
- Based on the first two fields of each line, set **CourseInfo**[courseId].instructorId = instructorId. If the instructor name is not specified (e.g., in a TA session), the instructorId is -1.[5]

[3]we need to use CourseInfo[c].lengPerSession to determine whether t falls inside that range.

[4]The weight is set to be 1 divided by sessionsPerWeek because we want to treat all instructors' preferences equally regardless of how frequently a course meets per week.

[5]See Appendex A.2 for the definition of the array **CourseInfo**.

- We create an array, **Instructor2Courses**, that stores the course(s) that an instructor teaches (an instructor can teach more than one course in a quarter).
- If the last three fields of the line are provided, use them to set the corresponding fields in **CourseInfo**[courseId]. For instance, if must-end-time is 11:20, the class must end by the time slot for 11-11:30.
- set **TotalCourseNum** to be the number of courses in this file.

B.3 Read the CourseInfo file

Read the file and set **CourseInfo** and **TotalNonExemptedHours**:

- The line has the format: "course-name leng-per-session sessions-per-week is-large-class exempted-from-10perc-rule is-TA-session must-be-on-same-day must-on-which-Day"
- If the course is not taught this quarter (i.e., course name is not a key in CourseName2Id), skip that line.
- Set the second part of the fields in **CourseInfo[courseId]** (see Appendix A.2).
- **TotalNonExemptedHours** stores the total number of hours subject to the 10% rule. It is set to 0 initially.
- **NonExemptedC** stores all the non-exempted courses.
- If the course is not exempted, TotalNonExemptedHours += CourseInfo[courseId].slotNum * CourseInfo[courseId].sessionsPerWeek / 2

B.4 Read the ConflictCourse file

In this file, each line is a list of courses whose time slots should not overlap. We use those lines to set a variable called **ConflictCoursePairs**:

- Each line is a list of course names: e.g., "570 571 566 550"
- For each input line, keep only the ones that are taught in this quarter and map them to course ids. If the number of ids is 2 or more, add all the pairs of ids to **ConflictCoursePairs**.
- Go over each element in **Instructor2Courses**; if an instructor is teaching more than one course, add all the course pairs to **ConflictCoursePairs**.
- When add a pair, choose the smaller id to be the first element.

B.5 Read the InstructorPref file

Read the file and set SameDayPairs and the matrix IW.

- Each line has the format: "instructor-name prefDays pref-start-time, pref-end-time same-day-or-not".
- If the instructor is not teaching that quarter based on **Instructor2Courses**, skip that line.
- We use **SameDayPairs** to store the pairs of courses that should be taught on the same day. The set is empty initially.

- If same-day-or-not is set to 1 and the instructor is teaching multiple courses based on **Instructor2Courses**, add all the course pairs to SameDayPairs.
- For the sake of simplicity when adding constraints, when we add a pair (id1, id2) to SameDayPairs, we choose the order of the two courses so that CourseInfo[id1].sessionsPerWeek $\leq$ CourseInfo[id2].sessionsPerWeek.
- If any of 2rd-4th fields is specified, for each course that the instructor is teaching this quarter, set the corresponding row in IW (see below).

Here is how we set matrix IW:

```
if none of prefDayStr, prefStartTime, and prefEndTime is provided:
   return   # no preference

# set default values
if prefStartTim is not specified:
   prefStartSlot = 0
else:
   prefStartSlot = slotName2Id(prefStartTime)

if prefEndTime is not specified:
   prefEndSlot = slotNumPerDay - 1
else:
   prefEndSlot = slotName2Id(prefEndTime)

if prefDays is not specified:
   prefDayList = [0, 1, 2, 3, 4]
else:
   prefDayList = DayNames2Ids(prefDays)  # map "MF" to [0, 4]

# set IW(c, d, t)
for each course c that the instructor teaches:
  set IW(c, d, t) based on prefStartSlot, prefEndSlot, prefDayList.
  Note that the length of the course matters, and
  the value of the preferred slot is 1/CourseInfo[c].sessionsPerWeek.
```

B.6 Set the Matrix CW according to the UW policy and course length

See Appendix A.5 for the definition of CW. Go over each course in CourseInfo and set the row in CW accordingly. For instance, if a course is 50-min long, the slots specified in **50-min-class-start-time** will be set to 1, other times before **BlockSchedulingEndsAt** will be set to **penalty-for-violating-block-polic**, and slots at **BlockSchedulingEndsAt** or later are set to 0.

B.7 Create an ILP problem

The decision variables (all are binary) are the cells in X and Y. The weights are CW and IW.

The objective function is: $\sum_{c,d,t}(\lambda_1 * CW_{c,d,t} + \lambda_2 * IW_{c,d,t}) * x_{c,d,t}$.

B.8 Add the constraints to the ILP problem

Let's define a few functions:

- CourseHasSameScheduleOnDays(c, d_1, d_2, ..., d_m) means the course c has the same schedule on a list of days; i.e., all these days have the class at the same time or no class on that day:

```
for slot t in range (0, slotNumPerDay):
    $x_{c,d_1,t} = x_{c,d_2,t} = ... = x_{c,d_m,t}$.
```

- CourseMustMeetOnDays(c, d_1, d_2, ..., d_m) means the course must be taught on those days (not necessarily at the time slot):

```
for each $d_i$ in the argument list:
    $\sum_t x_{c,d_i,t} \geq 1$
```

- CourseMustNotMeetOnDays(c, d_1, d_2, ..., d_m) means the course must NOT be taught on those days (not necessarily at the time slot):

```
for each $d_i$ in the argument list:
    $\sum_t x_{c,d_i,t} = 0$
```

We add the following constraints to the ILP problem:

1. Matrix Y must be consistent with Matrix X:

```
for each course c:
    slotNum = CourseInfo[c].slotNum
    for each day d in range(0, 5):
        $\sum_t X_{c,d,t} * slotNum = \sum_t Y_{c,d,t}$
        for each slot t in range(0, slotNumPerDay):
            for each j in range(t, min(t+slotNum, slotNumPerDay)):
                $Y_{c,d,j} \geq X_{c,d,t}$
```

2. Each course must meet the correct number of times per week:

```
for each course c:
    $\sum_{d,t} X_{c,d,t} = CourseInfo[i].sessionsPerWeek$
    If (CourseInfo[c].mustBeOnSameDay):
        then d1 = CourseInfo[c].mustOnWhichDay
                $\sum_t X_{c,d1,t} = CourseInfo[i].sessionsPerWeek$
        else for each day d in range(0, 5):
                $\sum_t x_{c,d,t} \leq 1$ # meet at most once per day
```

3. Each non-TA course that meets twice per week must be taught on MW or TR:[6]

```
for each course c that meets twice:
    if (CourseInfo[c].mustBeOnSameDay): continue
    SameSchedule(c, 1, 3) # T and R have the same schedule
    SameSchedule(c, 0, 2) # M and W have the same schedule
    courseMustNotMeetOnDays(c, 4) # not meet on Friday
    if (CourseInfo[c].isLargeClass):
        courseMustMeetOnDays(id, 1, 3) # must meet on T and R
```

4. Each non-TA course that meets three times per week must be taught on MWF:

```
for each course c that meets three times per week:
    if (CourseInfo[c].mustBeOnSameDay): continue
    CourseMustMeetOnDays(c, 0, 2, 4)
    CourseHasSameScheduleOnDays(c, 0, 2, 4)
```

5. Conflicting courses should not overlap in time:

```
for each (c1, c2) in ConflictCoursePairs:
    for each day d in range(0, 5):
        for each slot d in range(0, slutNumPerDay):
            # at most one course is assigned at this time slot
            $Y_{c1,d,t} + Y_{c2,d,t} \leq 1$
```

6. Meet the 10%-rule requirement:

```
target = RulePercentage * TotalNonExemptedHours
for each t in range(10PercRuleStart, 10PercRuleEndHour, 2):
    t1 = $\sum_{c \in NonExemptedC} \sum_d Y_{c,d,t}$
    t2 = $\sum_{c \in NonExemptedC} \sum_d Y_{c,d,t+1}$
    $(t1 + t2)/2 \leq target$
```

7. The SameDay preferences are treated as hard constraints:

```
for each (c1, c2) in SameDayPairs:
    # we assume CourseInfo[c1].SessionsPerWeek $\leq$ CourseInfo[c2].sessionsPerWeek
    for each day d:
        # if c1 is taught on d, so must c2
        $\sum_t X_{c1,d,t} \leq \sum_t X_{c2,d,t}$
```

[6] While they can meet on MF or WF, we prefer not to meet on F as that is the day for lab meetings etc.

B.9 Call the ILP solver to find the optimal solution

Call the ILP solver to find the optimal solution. This step should just be one or a few lines of code.

B.10 Generate the time schedule output file

Once the optimal solution is provided, use Matrix X, CW, and IW to produce the scheduling file. See Section 3.1 for detail.

B.11 Generate the heat map for the 10% rule

The format of the heat map file is in Section 3.2. To generate the file, use Matrix Y.

- For instance, to get the value for (8:30, M), add the column total for M 8:30 and M 9:00 in Matrix Y.
- The Hourly Total column is just the row sum of M-F.
- The Hourly Target column has the value floor(RulePercentage * TotalNonExemptedHours).