MATH380 Project Report

Lee, Kunwoo

Jung, Juhyun

Contents

1. Introduction

The registrar's office at the Illinois Institute of Technology is in charge of scheduling classes and tests for each semester. To avoid scheduling conflicts, this duty entails choosing acceptable times and places for each course. Even if it seems straightforward, the subtleties reveal how difficult the task actually is. We will create an automated decision-making system to lessen the complexity of this task. In order to provide answers that satisfy the needs of departments, teachers, and students, this system makes use of a mathematical model and solver. By streamlining the process, the technology shortens the time needed to plan classes and tests. Additionally, this automated system makes sure that students can access the courses they need and that teachers can control their classrooms.

(a) Defining the problem Statement

To solve the problem, we divided the optimization into two stages. We have performed time-related optimization (*when* to have certain course) and classroom-related optimization (*where* to have certain course).

For time-optimization, the goal of this project is to come up with course schedule that *satisfies* students, faculty, and the school. As this is a subjective criterion, we came up with our own idea of a "nice" schedule. There is clearly some kind of preference of schedule, that most students agree on. For example, if one student's course has 8:35~9:50 & 3:15~4:30 every weekday, no one likes this schedule. We tried to incorporate this idea into our objective function.

For classroom-optimization, we set the goal to minimize the number of classrooms used. To do this, we modelled using graph, and obtained the chromatic number using greedy coloring algorithm. We will discuss this idea further in chapter 3.

(b) Making Assumptions

In the real world, there are a lot of constraints. However, since it is too complicated, we made some simplifying assumptions. We will talk about this in chapter 3 more specifically.

(c) Defining Variables

For time-optimization, we defined two main binary variables. One is related to course-time, being 1 if certain course is at certain time, and 0 otherwise. The other one is related to course-professor, being 1 if certain course is instructed by certain professor, and 0 otherwise.

For classroom-optimization, for the graph, we defined 22 vertices for courses and edges for the time conflict between the courses. We will give detailed description about variables in chapter 3.

(d) Getting a solution

By using the assumptions, objective function and variables introduced previously, we used PuLP library in python, and greedy coloring algorithm to optimize course schedule.

(e) Model Assessment

From the solution we obtained, we visualized the schedule and verified whether it looks nice or not. Also, we discussed our weaknesses and strengths of our model to improve it.

2. Statement and Analysis of the Problem

The given problem statement was:

"Each semester, the registrar's office at IIT has to coordinate and decide when (timeslot) and where (classroom/lab) to hold each and every course supposed to be offered that semester, and then later that semester, the final exams for those courses also have be scheduled without any conflicts. You have been hired by the President of IIT to design and build an automated decision system (based on a mathematical model and a solver for it) to help the registrar's office with this process so that they get faster solutions that better satisfy the departments, the faculty and the students."

To simply put, it is an optimization problem to decide whether to have certain course at a certain time and at a certain classroom that better ***satisfies*** the school, the faculty, and the students (Note: we didn't have enough time for the final exam scheduling, so we focused on course scheduling). So, to begin with, it was crucial to decide what is a "nice" course schedule that satisfies everyone, as it is a subjective statement.

Thinking from the perspective of students, we thought that having a time schedule for that are evenly distributed along the whole week was a nice time schedule. Having a widely spread classes along the week reduces the workload at specific date of the week, giving students enough time to prepare for the class, and do homework. Also, if students have a concentrated load of class at one day, their concentration level will drop exponentially, reducing the quality of education.

The faculty members are in line with the perspective of students. We thought that if students have trouble with the concentrated workload, it would be much harder for the faculty to

teach the students, lead their participation and get them to do the homework. So, widely spread classes along the week seemed to be a nice option for the faculty as well.

We approached the school's perspective in a more maintenance-cost-wise way. If we use more classrooms for classes, it will cost more money to maintain the classrooms. So, we decided minimizing the number of classrooms used will be a nice option for the school.

So, we restated the given problem statement as:

"Make a conflict-free course schedule (when & where), such that the class schedule is widely spread throughout the week for each student and uses least number of classroom possible."

To perform the desired optimization, we searched for methods that can be used to solve the problem. We divided the problem statement into two parts: assigning optimal time for courses and assigning optimal classroom for courses.

Fort part 1, time-optimization, we couldn't find any literature that has the specific objective of our definition of "nice" course schedule. So, we decided to learn how to use a binary optimization library in Python named PuLP to implement our idea of "nice" schedule. We could find an example of making a best soccer team inside the budget based on cost and value of the soccer player. We could learn the basic grammar for the library, but the example was more about a linear constraint, which was very different from ours. The constraint of our problem was difficult to write in linear constraints. We will discuss these later, but having the specific constraints was very painful as we couldn't find the literature for it, and we had to come up with our own ideas.

For part 2, classroom-optimization, it was relatively easy to perform. Our objective was to minimize the number of classrooms used, which can be obtained from the simple greedy

coloring algorithm we learned in class. We could find a source code that does the greedy coloring algorithm that also includes the part to set constraints for each node (what color can be used for certain node).

3. Description of the Model

3.1. Optimization of time assignment. (Time-optimization)

We will first discuss the problem of assigning optimal time of each course, starting from listing the assumptions. Most of them are simplifying assumptions and we listed the justifications below each assumption.

3.1.1 Assumptions for time-optimization & justification

- There are 6 possible timeslots for each day (8:35 - 9:50, 10:00 - 11:15, 11:25 - 12:40, 13:50 - 15:05, 15:15 - 16:30, 17:00 - 18:15). The duration of each timeslot will be denoted as unit time. (1 unit time = 1 hour 15 minutes)
    - The basic structure of timeslots follows the schedule of IIT, except that all class ends at 18:15. Assuming all class ending before dinner time is in line with the quality of the schedule for students and faculties.
- There is no online class.
    - Most courses are in-person, and adding few online classes after optimization won't affect the quality much.
- We only consider undergraduate courses.

    - There are co-terminal students who take the graduate courses, but there are not many, and we assume that they are prepared for schedules that are less “nice”.
- We only consider MMAE department, and courses names that start with “MMAE”.
    - To simplify the optimization, we only done it on MMAE department. Adding other departments in the future will be possible.
- For classes consisting of lecture and lab, the materials touched in lab is from previous week’s lecture, so that there is no specific order for lecture and lab to be scheduled.
    - This is true in IIT, so it is justifiable.
- There is only one class for each course. (e.g., there is only MATH 380. No MATH 380-01, MATH 380-02, …)
    - This is mostly true for courses that start with “MMAE”, so it won’t produce too much error. Also, we can always add separated classes later, so we started with the simplest possible case.
- The recommended course schedule on MMAE department website will represent the schedule of students in the department.
    - For MMAE department, due to the prerequisites and the desire to take classes with one’s friends, students follow a coherent path in taking class. We assumed that the recommended schedule on the MMAE department website is the coherent path.

Having listed all the assumptions, we now consider the constraints.

3.1.2 Time-Optimization: Constraints, Mathematical Formulations

- Constraint 1: There are 5 possible cases a class can be held, according to the Fall semester 2022.
- Constraint 2: A course taught by one professor can't be held simultaneously.
- Constraint 3: Students who follow the recommended course schedule from MMAE department website should not experience time conflicts.

Having listed all the constraints, we now consider writing them in mathematical formulation, and implement it in code. To start with, we define variables.

- $i: \ the\ index\ of\ a\ certain\ course. i = 0,1,2,\ldots,22 \ \ (e.g., i = 0: \ MMAE\ 100)$
- $j: \ index\ of\ certain\ timeslot. j = \ 0,1,2,\ldots,30$
  - $(e.g., j = 0: \ Monday\ 8{:}35 - 9{:}50, \ j = 30: \ Friday\ 17{:}00 - 18{:}15)$
- $k: \ index\ of\ certain\ professor\ (e.g., k = 0: \ professor\ Ernhardt\ E\ John)$
- $variable$ "choices" $is\ a\ 22 \times 30\ matrix\ such\ that$

$$c_{ij}(\in choices) = \begin{cases} 1 \ \ if\ course\ i\ is\ in\ timeslot\ j \\ 0\ therwise \end{cases}$$

- $variable$ "prof_const" $is\ a\ 22 \times 18\ matrix\ such\ that$

$$y_{ik}(\in prof\_const): \begin{cases} 1 \ \ if\ course\ i\ is\ instructed\ by\ professor\ k \\ 0 \ \ otherwise \end{cases} \ \ (\sum_k y_{ik} = 1)$$

- $subvariable$ "lecture_temp" $is\ a\ 22 \times 30\ matrix\ such\ that$

$$lec_{ij}(\in lecture\_temp) = \begin{cases} 1 \ \ if\ there\ is\ lecture\ session\ for\ course\ i\ in\ timeslot\ j \\ 0\ therwise \end{cases}$$

- $subvariable$ "lab_temp" $is\ a\ 22 \times 30\ matrix\ such\ that$

$$lab_{ij}(\in lab\_temp) = \begin{cases} 1 \ \ if\ there\ is\ lab\ session\ for\ course\ i\ in\ timeslot\ j \\ 0\ therwise \end{cases}$$

Constraint 1: There are 5 possible cases a class can be held.

Case 1: (denoted "C21")

- 2 lecture sessions, that are each a unit time long.
- Two lectures are either Monday-Wednesday or Tuesday-Thursday.

$for\ i\ corresponding\ to\ case\ 1,\ \sum_j c_{ij} = 2$

$for\ i\ corresponding\ to\ case\ 1,\ if\ j < 12,\ c_{ij} - c_{ij+12} = 0$

$for\ i\ corresponding\ to\ case\ 1, if\ j \geq 24\ ,\ c_{ij} = 0$

Case 2: (denoted "C11L12")

- 1 lecture session of a unit time and 1 lab session with 2 unit time.

$for\ i\ corresponding\ to\ case\ 2,\ \sum_j c_{ij} = 3$

$for\ i\ corresponding\ to\ case\ 2\ and\ j\ in\ interval\ [0,29],$

$there\ is\ at\ least\ one\ value\ of\ j\ that\ makes\ c_{ij} + c_{ij+1} \geq 2$

Case 3: (denoted "C21L12")

- 2 lecture sessions, and 1 lab session with 2 unit time.
- Two lecture sessions are either in Monday-Wednesday set or Tuesday-Thursday set.

$for\ i\ corresponding\ to\ case\ 3,\ \sum_j lec_{ij} = 2$

$for\ i\ corresponding\ to\ case\ 3,\ if\ j < 12,\ lec_{ij} - lec_{ij+12} = 0$

$for\ i\ corresponding\ to\ case\ 3, if\ j\ \geq\ 24\ ,\ lec_{ij}\ =\ 0$

$for\ i\ corresponding\ to\ case\ 3,\ \sum_j lab_{ij} =\ 2$

$for\ i\ corresponding\ to\ case\ 3\ and\ j\ in\ interval\ [0,29],$

$there\ is\ at\ least\ one\ value\ of\ j\ that\ makes\ lab_{ij}\ +\ lab_{ij+1}\ \geq\ 2$

$for\ i\ corresponding\ to\ case\ 3, lec_{ij}\ +\ lab_{ij}\ \leq\ 1\ for\ all\ j$

$for\ i\ corresponding\ to\ case\ 3, c_{ij}\ =\ lec_{ij}\ +\ lab_{ij}\ for\ all\ j$

Case 4: (denoted "C11L23")

- 1 lecture of a unit time, and 2 lab sessions with 3 unit time each.
- Two lab sessions can be in either Monday-Wednesday set or Tuesday-Thursday set.

$for\ i\ corresponding\ to\ case\ 4,\ \sum_j lec_{ij} =\ 1$

$for\ i\ corresponding\ to\ case\ 4\ and\ j\ <12\ ,\ \sum_j lab_{ij} =\ 3$

$for\ i\ corresponding\ to\ case\ 4\ and\ j\ \geq 12\ ,\ \sum_j lab_{ij} =\ 3$

$for\ i\ corresponding\ to\ case\ 4\ and\ j\ in\ interval\ [0,9],$

$there\ is\ at\ least\ one\ value\ of\ j\ that\ makes\ lab_{ij}\ +\ lab_{ij+1} +\ lab_{ij+2}\ \geq\ 3$

$for\ i\ corresponding\ to\ case\ 4\ and\ j\ in\ interval\ [12,28],$

$there\ is\ at\ least\ one\ value\ of\ j\ that\ makes\ lab_{ij}\ +\ lab_{ij+1} +\ lab_{ij+2}\ \geq\ 3$

$for\ i\ corresponding\ to\ case\ 4, lec_{ij}\ +\ lab_{ij}\ \leq\ 1\ for\ all\ j$

$for\ i\ corresponding\ to\ case\ 4, c_{ij} = lec_{ij} + lab_{ij}\ for\ all\ j$

Case 4: (denoted "C12")

- 1 lecture session of 2 unit time.

$for\ i\ corresponding\ to\ case\ 4, \sum_j c_{ij} = 3$

$for\ i\ corresponding\ to\ case\ 4\ and\ j\ in\ interval\ [0,29],$

$there\ is\ at\ least\ one\ value\ of\ j\ that\ makes\ c_{ij} + c_{ij+1} \geq 2$

Constraint 2: A course taught by same professor can't be held simultaneously.

$for\ all\ values\ of\ j,\ all\ the\ elements\ in\ vector\ \sum_i c_{ij} \times y_{ik} \leq 1$

For example, if $j = 0$ and $i = 0$,

c_{ij} will be 1 if course 0 (MMAE 100) is in timeslot 0 (Monday 8:35 - 9:50), and 0 otherwise. And y_{ik} will represent a vector for who teaches MMAE 100. For instance, if y_{ik} = [1,0,0,...0], it means that professor Ernhardt E John will be teaching MMAE 100. So, if we add vectors of corresponding to i values $c_{ij}y_{ik}$ for a constant j, all element in it should be less or equal to 1.

Constraint 3:

Students who follow the recommended course schedule from MMAE department website should not experience time conflicts.

The recommended course schedule looks like below.

Bachelor of Science in Mechanical Engineering Curriculum

YEAR 1			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 100	3	MS 201	3
MATH 151	5	MATH 152	5
CHEM 124	4	PHYS 123	4
Humanities 200-level Course	3	CS 104	2
		Social Sciences Elective	3
	15		17
YEAR 2			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 202	3	MMAE 350	3
MMAE 232	3	MATH 252	4
MATH 251	4	Free Elective	3
PHYS 221	4	Humanities Elective (300+)	3
Humanities or Social Science Elective	3	Social Sciences Elective (300+)	3
	17		16
YEAR 3			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 302	3	MMAE 319	4
MMAE 305	3	MMAE 321	3
MMAE 313	3	MMAE 323	3
MMAE 320	3	MMAE 332	3
Humanities Elective (300+)	3	Social Sciences Elective (300+)	3
	15		16
YEAR 4			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 419	4	MMAE 432 or 433	3
MMAE 443	3	Technical Elective[2]	3
MMAE 485	3	Technical Elective[2]	3
Technical Elective[1]	3	IPRO Elective II	3
IPRO Elective I	3	Free Elective	3
	16		15
Total Credit Hours: 127			

Bachelor of Science in Aerospace Engineering Curriculum

YEAR 1			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 100	3	MS 201	3
MATH 151	5	MATH 152	5
CHEM 124	4	PHYS 123	4
Humanities 200-level Course	3	CS 104	2
		Social Sciences Elective	3
	15		17
YEAR 2			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 202	3	MMAE 313	3
MATH 251	4	MMAE 320	3
PHYS 221	4	MATH 252	4
Humanities or Social Sciences Elective	3	Free Elective	3
Humanities Elective (300+)	3	Social Sciences Elective (300+)	3
	17		16
YEAR 3			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 311	3	MMAE 304	3
MMAE 312	3	MMAE 305	3
MMAE 315	4	MMAE 352	3
MMAE 350	3	MMAE 372	3
Free Elective	3	Humanities Elective (300+)	3
	16		15
YEAR 4			
SEMESTER 1	CREDIT HOURS	SEMESTER 2	CREDIT HOURS
MMAE 419	3	MMAE 412	3
MMAE 411	3	MMAE 415	4
MMAE 414	3	IPRO Elective II	3
MMAE 443	3	Technical Elective[2]	3
IPRO Elective I	3	Social Sciences Elective (300+)	3
	15		16
Total Credit Hours: 127			

I extracted the name of the required courses in Fall semester,

```
req_courses_name = [[202, 232], [302, 305, 313, 320], [419, 443,
485], [311, 312, 315, 350], [410, 411, 414, 443], [202, 232], [320,
365, 370, 373], [470,476,485]]
```

Translating course names into i values we get,

```
 set = [[1, 2], [3, 4, 7, 9], [17, 18, 21], [5, 6, 8, 10],
[14, 15, 16, 18], [1, 2], [9, 11, 12, 13], [19, 20, 21]]
```

Let $combination \in set$. Then, for all combination,

$$for\ all\ j, \sum_{i \in combination} c_{ij} \leq 1$$

3.1.3 Time-Optimization: Mathematical Formulation of Problem Objective

"Make a conflict-free course schedule such that the class schedule is widely spread throughout the week for each student."

1. Define $Vector\ Z\ = \sum_{i\ \in\ combination} \overrightarrow{c_i}$,
2. Append 1 in each end of $\vec{Z}$.
3. Create a new $vector\ Z\_idx$, which contain the indices of 1 in $\vec{Z}$.

   $(e.g., if\ vector\ Z\ =\ [1,0,1,0,0,1,0,1,0,1], Z\_idx\ =\ [1,3,6,8,10])$
4. Create a new vector $idx\_distance$, which contains the difference of two neighboring values of vector Z_idx. $(e.g., if\ vector\ Z\_idx\ =\ [1,3,6,8,10], idx\_distance\ =\ [2,3,2,2])$
5. Then, get all the values of $idx\_distances$ divided by the length of $idx\_disances$.
6. Square the values and add them up, then append the value to list "$loss$"
7. Repeat step 1- 6 for all $combination\ \in\ set$.

Objective Function: Minimize $\sum loss$

3.2. Optimization of classroom assignment. (Classroom-optimization)

3.2.1 Assumptions for classroom-optimization & justification

- All classrooms are close enough that it is within 10 minutes of walking distance. Students having consecutive classes can take the later class even if two classrooms are the furthest possible.
    - IIT is relatively small campus, and this is true.
- For the MMAE department, we can use 3 auditorium and 5 classrooms.

- As we are only discussing MMAE department, they shouldn't have too many available classrooms considering other departments. Based on our experience, there seems to be 3 auditorium and 5 classrooms available for MMAE.

3.2.2 Classroom Optimization: Constraints & Mathematical Formulation

Constraint 1: The same classroom can't be assigned for two classes at overlapping time.

As we learned in class about modeling with graphs, we put classes on nodes and the time conflict between classes in edges. Then, by using the greedy coloring algorithm, we could find the minimal number of classrooms needed.

Constraint 2: Due to the limited capacity of classrooms, there are classes that require auditorium.

We collected the data of maximum occupancy of each course, then classified it into two types, which are courses that need auditorium and courses that classroom is sufficient. Then, according to the data, we assigned each node with possible classroom/auditorium. (e.g., MMAE 100 has maximum occupancy of 40 people, so we assigned it with possible auditoriums ['Auditorium1', 'Auditorium 2', 'Auditorium 3']).

4. Analysis and Testing of the Model

1. Input data

We used course information of 2022 Fall semester obtained from MMAE department webpage.

MMAE	Way	Classroom	Lab	Professor		maximum seats
100	C21	RE 141		Ruiz Francisco		40
202	C21	RE 104		Cesarone C John		85
232	C11L12	HH 002	RE141	Cesarone C John		60
302	C21	PH 131		Saghaian M Sayed		85
305	C21	RE 104		Parvizi Roohollah		85
311	C21	PH 108		Naqib M. Hassan		42
312	C21	RE 258		Bernhardt E John		42
313	C21	RE 104		Dawson Scott		80
315	C21L12	PS 152	RE 142	Bernhardt E John		48
320	C21	SH 118	PS 152	Ostrogorsky Aleks	Heidarinejad Moha	60
350	C21	RE 104		Srivastava Ankit		90
365	C21	SB 220		Chen Wei		12
370	C11L23	RE 053		Wang Heng		10
373	C21L12	RE 018		Wang Heng		18
410	C21	PS 152		Bernhardt E John		45
411	C21	HH 003		Pervan S. Boris		45
414	C21	RE 104		Williams R. David		45
419	C21L12	RE104	RE 018	Vural Murat		100
443	C21	PH 131		HomChaudhuri Bai		105
470	C12	WH 116		Shah R. Anand		15
476	C11L23	RE 018	RE 053	Wang Heng		10
485	C21	RE 104		Mostafaei Amir		100

Also, from the IIT sample curriculum, we made sets of required courses to use for constraint and objective function.

```
req_courses_name = [[202, 232], [302, 305, 313, 320], [419, 443, 485], [311, 312, 315, 350], [410, 411, 414, 443], [202, 232], [320, 365, 370, 373], [470,476,485]]
```

2. Binary Optimization Algorithm

A method for resolving optimization issues involving binary decision variables is the binary optimization algorithm. In this algorithm, the issue domain is represented by a collection of binary choice

variables, and the objective function is assessed for every conceivable combination of the binary variables until the best solution is identified.

The basic steps of binary optimization algorithm are as follows:

1. Initialize the binary decision variables to a random state.
2. Evaluate the objective function for the current set of binary decision variables.
3. If the objective function meets the stopping criteria, return the solution.
4. If the stopping criteria are not met, update the binary decision variables based on a set of rules.

Repeat steps 2-4 until the stopping criteria are met.

3. PuLP library

PuLP is a Python library for linear and mixed-integer programming that provides high-level functions to build models and algorithms that make it simple and effective for users to construct and solve linear programming issues.

The basic steps of PuLP algorithm are as follows:

1. Establishing a new instance of the optimization problem.
2. Specifying the objective function.
3. Incorporating constraints into the problem.
4. Determining the type of optimization problem (maximization or minimization).
5. Solving the problem with the available optimization algorithms.
6. Accessing and examining the results.

Creating a new optimization problem instance: Create new optimization problem instance before utilizing PuLP. A new PuLP Problem object, which acts as a representation of the optimization issue, can be created to achieve this.

```
# 2D Decision variable
COURSE_NAME = Data["MMAE"][:22]
COURSE = list(range(22)) # course slot : 22
TIMESLOT = list(range(30)) # time slot : 6 x 5 = 30
#print(TIMESLOT)
choices = pulp.LpVariable.dicts("Choice", (COURSE, TIMESLOT), lowBound = 0, upBound = 1, cat= "Binary") #  if course is in timeslop, then 1. otherwise 0.
# print(choices.keys())
lecture_temp = pulp.LpVariable.dicts("temp_course", (list(range(22)), list(range(30))), cat = 'Binary')
lab_temp = pulp.LpVariable.dicts("temp_lab", (list(range(22)), list(range(30))), cat = 'Binary')



prob  = pulp.LpProblem( "Best_Schedule", pulp.LpMinimize )

req_courses_name = [[202, 232], [302, 305, 313, 320], [419, 443, 485], [311, 312, 315, 350], [410, 411, 414, 443], [202, 232], [320, 365, 370, 373], [470,476,485]]

req_courses = []
for req_c in req_courses_name:
    temp = []
    for course in req_c:
        temp.append(np.where(COURSE_NAME==course)[0][0])
    req_courses.append(temp)
```

Defining the objective function: The objective function is the function that the optimization algorithm will try to maximize or minimize.

```
loss = []
for req_c in req_courses:
    req_courses_timeslot=[]
    for course in req_c:
        req_courses_timeslot.append(list(choices[course].values())) ## row vector 2x30
    z=[1]       ## add one in beginning
    for i in range(len(req_courses_timeslot[0])): # column
        temp=0
        for j in range(len(req_courses_timeslot)): # row
            temp = temp+req_courses_timeslot[j][i] # column wise sum
        z.append(temp)
    z.append(1)  ## add one in end
    z_idx = list(filter(lambda x:z[x]==1,range(len(z)))) # [1,1,1,0,0,1] - > 01235

    idx_distance = []
    for i in range(len(z_idx) - 1):
        idx_distance.append(z_idx[i+1] - z_idx[i])          # front index  - back index - > EX) 01235 - > 1 1 1 2

    loss.append(sum([(k/len(idx_distance))**2 for k in idx_distance]))


prob += pulp.lpSum(loss)
```

Adding constraints to the problem: Constraints indicate the necessary conditions that must be fulfilled to validate the optimization problem. **(APPENDIX)**

Setting the optimization problem type: Establish the type of optimization problem by indicating whether it is a maximization or minimization problem.

```
prob  = pulp.LpProblem( "Best_Schedule", pulp.LpMinimize )
```

Solving the optimization problem: PuLP is equipped with an array of potent optimization algorithms that can be employed to solve linear programming problems.

```
### Solve
prob.solve()
```

4. Greedy coloring

The greedy coloring algorithm's main goal is to use a heuristic technique to solve graph coloring problems. The algorithm seeks to reduce the number of colors needed to color a graph while preventing nearby vertices from sharing a color.

The basic steps of Greedy coloring algorithm are as follows:

1. The vertices are arranged in ascending order, based on the number of adjacent vertices they have, during the greedy coloring algorithm. This sorting procedure is implemented to prioritize the vertices with higher degrees initially since they tend to have more color constraints.

```
# use selection sort
for i in range(len(degree)): #0~21
    _max = 0
    j = 0
    for j in range(len(degree)): ##0~21
        if j not in indeks:
            if degree[j] > _max:# _max=2
                _max = degree[j]
                idx = j
    indeks.append(idx)
    sortedNode.append(node[idx])
```

2. Before applying the greedy coloring algorithm, a dictionary and an empty color list are established. The list of colors contains the assigned colors for each vertices, while the dictionary contains the colors used by adjacent vertices for each vertices. The greedy coloring procedure iterates over each vertex in the sorted list. For each vertex, the algorithm selects the smallest color that hasn't been used by any of the neighboring vertices. If all of the colors used by nearby

vertices have already been assigned, the method assigns a new color. This step ensures that adjacent vertices have different colors.

```
# The main process
theSolution={}
for n in sortedNode:

    setTheColor = colorDict[n]
    theSolution[n] = setTheColor[0]
    adjacentNode = G[t_[n]]
    for j in range(len(adjacentNode)):
        if adjacentNode[j]==1 and (setTheColor[0] in colorDict[node[j]]):
            colorDict[node[j]].remove(setTheColor[0])
```

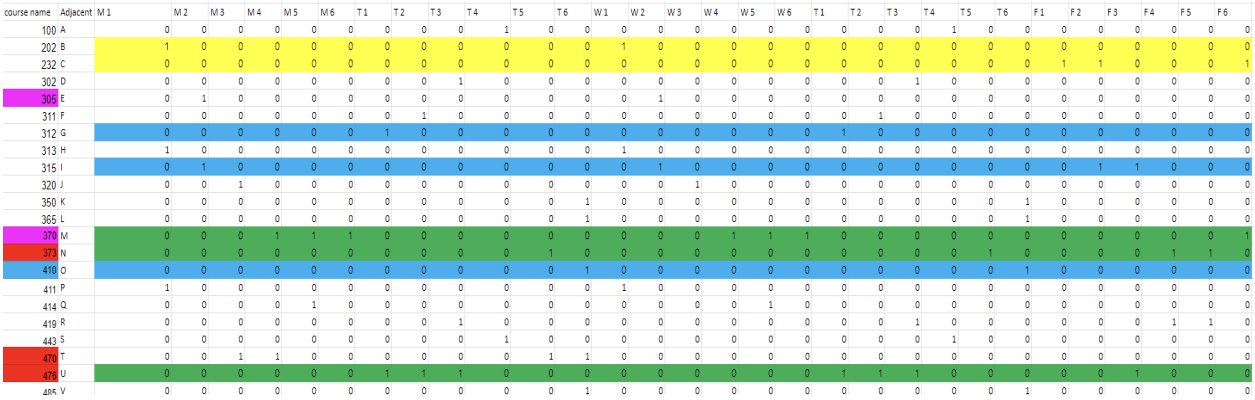
3. The number of colors used after assigning a unique color to each vertex in the graph satisfies the criteria that no two adjacent vertices have the same color. This number is the bare minimum needed to color the graph.

```
for t,w in sorted(theSolution.items()):
    print("MMAE",t," = ",w)
```

5. Results and Discussion

Result

We run our model based on the fall semester sample curriculum.



course name	Adjacent	M 1	M 2	M 3	M 4	M 5	M 6	T 1	T 2	T 3	T 4	T 5	T 6	W 1	W 2	W 3	W 4	W 5	W 6	T 1	T 2	T 3	T 4	T 5	T 6	F 1	F 2	F 3	F 4	F 5	F 6
100	A	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
202	B	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
232	C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
302	D	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
305	E	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
311	F	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
312	G	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
313	H	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
315	I	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
320	J	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
350	K	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
365	L	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
370	M	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1
373	N	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
410	O	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
411	P	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
414	Q	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
419	R	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0
443	S	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
470	T	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
476	U	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0
485	V	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

This is our result, and we made sample schedules for recommended course combination on the MMAE webpage.

3rd Mechanical	Monday	Tuesday	Wednesday	Thursdays	Friday
8:35 ~ 9:50	MMAE313(A2)		MMAE313(A2)		
10:00 ~ 11:15	MMAE305(A2)		MMAE305(A2)		
11:25 ~ 12:40	MMAE320(A1)	MMAE302(A2)	MMAE320(A1)	MMAE302(A2)	
1:50 ~ 3:05					
3:15 ~ 4:30					
5:00 ~ 6:15					

3rd Material	Monday	Tuesday	Wednesday	Thursdays	Friday
8:35 ~ 9:50					
10:00 ~ 11:15					
11:25 ~ 12:40	MMAE320(A1)		MMAE320(A1)		
1:50 ~ 3:05	MMAE370(C2)		MMAE370(C2)		
3:15 ~ 4:30	MMAE370(C2)	MMAE373(C2)	MMAE370(C2)	MMAE373(C2)	
5:00 ~ 6:15	MMAE370(C2)	MMAE365(C2)	MMAE370(C2)	MMAE365(C2)	MMAE370(C2)

3rd Aerospace	Monday	Tuesday	Wednesday	Thursdays	Friday
8:35 ~ 9:50		MMAE312(A1)		MMAE312(A1)	
10:00 ~ 11:15	MMAE315(A1)	MMAE311(A1)	MMAE315(A1)	MMAE311(A1)	MMAE315(A1)
11:25 ~ 12:40					MMAE315(A1)
1:50 ~ 3:05					
3:15 ~ 4:30					
5:00 ~ 6:15		MMAE350(A1)		MMAE350(A1)	

This is Junior's schedule. It looks "Nice schedule", because the time gap between classes is not too much and classes are evenly distributed.

4th Material	Monday	Tuesday	Wednesday	Thursdays	Friday
8:35 ~ 9:50		MMAE476(C1)		MMAE476(C1)	
10:00 ~ 11:15		MMAE476(C1)		MMAE476(C1)	
11:25 ~ 12:40	MMAE470(C1)	MMAE476(C1)		MMAE476(C1)	MMAE476(C1)
1:50 ~ 3:05	MMAE470(C1)				
3:15 ~ 4:30					
5:00 ~ 6:15		MMAE485(A3)		MMAE485(A3)	

4th Material	Monday	Tuesday	Wednesday	Thursdays	Friday
8:35 ~ 9:50		MMAE476(C1)		MMAE476(C1)	
10:00 ~ 11:15		MMAE476(C1)		MMAE476(C1)	
11:25 ~ 12:40	MMAE470(C1)	MMAE476(C1)		MMAE476(C1)	MMAE476(C1)
1:50 ~ 3:05	MMAE470(C1)				
3:15 ~ 4:30					
5:00 ~ 6:15		MMAE485(A3)		MMAE485(A3)	

4th Aerospace	Monday	Tuesday	Wednesday	Thursdays	Friday
8:35 ~ 9:50	MMAE411(A3)		MMAE411(A3)		
10:00 ~ 11:15					
11:25 ~ 12:40					
1:50 ~ 3:05		MMAE443(A2)		MMAE443(A2)	
3:15 ~ 4:30	MMAE414(A1)		MMAE414(A1)		
5:00 ~ 6:15		MMAE410(A2)		MMAE410(A2)	

This is senior's schedule. It looks "Nice schedule" except for the aerospace students.

If we see the total schedule, it looks nice except for the 4th aerospace student who has a "Bad schedule" that we think. However, it is impossible to make a "nice" schedule for everyone considering limited classroom, professor, and time.

Possible Modifications

- Our model is only takes MMAE classes into account, excluding all other courses. If we add in more classes, we can't assure that we can get the same optimal result. We will have to collect more input data and run the optimization and evaluate the result for modification.
- We assumed that there is no division of classes. This is a good way to relax the model, as for now, we only have one class for each course, constraining the model very hard. To handle this, we will have to collect more data of the class division and create more rows in matrix "choices" and "prof_const" to account for the increased courses. Also, we will have to modify the third constraint in time-optimization into:

  there is one possible combination such that $for\ all\ j, \sum_{i\ \in\ combination} c_{ij}\ \leq\ 1$

- Our optimization is based on recommended course schedule, which means that students who are not following it might experience many time conflicts. However, we believe this is the limitation of college system, as we can't make every single possible combination of courses to have no time conflict. So, this problem can't be resolved, unless we use different time for every single courses.
- During the process of reducing the total loss of our model, some students (like the aerospace students) can have relatively bad. This is inevitable, but the total inconvenience can be reduced by assigning the "bad schedule" for departments with least students (although there may be ethical issues), by assigning weight to loss from each department based on the number of students.
- We assumed that all classes end before 6:15 PM. However, if we consider All classes, can consider having classes after 6:15PM, to make "nicer" schedules.
- We did not consider that courses are usually held in buildings related to the department of that course (e.g., Engineering-related: RE, WH, … / Computer Science: Stuart, … / HUM: Siegel, …). This can be modified by inputting possible building for each node, rather than saying "auditorium 1, auditorium 2, ...".

- We did not consider distance between classrooms. For example, it is not optimal for students to take consecutive classes that are held in IIT tower and Stuart building. This can be improved by adding a new objective function for classroom-optimization, rather than having least number of classrooms.

6. References

Hands On Integer (Binary) Linear Optimization using Python : Soccer Team Optimization Example

Optimization with PuLP : PuLP document

Solve Graph Coloring Problem with Greedy Algorithm and Python : Greedy Coloring Source Code

PuLP doesn't take if-else : PuLP doesn’t take if-else, giving us enlightenment.

7. Appendix

In [ ]:
```
import numpy as np
import pulp
import pandas as pd
from itertools import accumulate
import operator

Data = pd.read_excel("MATH380_Project_Data.xlsx")

# 2D Decision variable
COURSE_NAME = Data["MMAE"][:22]
COURSE = list(range(22)) # course slot : 22
TIMESLOT = list(range(30)) # time slot : 6 x 5 = 30
#print(TIMESLOT)
choices = pulp.LpVariable.dicts("Choice", (COURSE, TIMESLOT), lowBound = 0, upBound = 1, cat= "Binary") #  if course is in timeslop, then 1. otherwise 0.
# print(choices.keys())
lecture_temp = pulp.LpVariable.dicts("temp_course", (list(range(22)), list(range(30))), cat = 'Binary')
lab_temp = pulp.LpVariable.dicts("temp_lab", (list(range(22)), list(range(30))), cat = 'Binary')



prob  = pulp.LpProblem( "Best_Schedule", pulp.LpMinimize )

req_courses_name = [[202, 232], [302, 305, 313, 320], [419, 443, 485], [311, 312, 315, 350], [410, 411, 414, 443], [202, 232], [320, 365, 370, 373], [470,476,485]]

req_courses = []
for req_c in req_courses_name:
    temp = []
    for course in req_c:
        temp.append(np.where(COURSE_NAME==course)[0][0])
    req_courses.append(temp)

print(req_courses)

########################################## Constraint 1 ##################################################################
"""
course time consist of (Monday, Wednesday),(Tuesday, Thursday), (Theory 2hours, lab 2 hours), (Theory 1hour, lab 3hours x 2)
ex) x_ij = 1 ->  x_i(j+6) = 1

"""

time_const = Data['Time'][:22]

lab = TIMESLOT
for i in range(5, 30, 6):
    lab.remove(i)
# lab.remove(5, 11, 17, 23, 29) # possible consecutive course, and the every 6th time cannot be consecutive so remove


TIMESLOT = list(range(30))
lab2 = TIMESLOT[:22]

delete_list_lab2 = [4,5,10,11,16,17]
for delete in delete_list_lab2:
    lab2.remove(delete) # for three hours consecutive lab


for i in range(len(COURSE)):

    if time_const[i] == "C21":
        prob += pulp.lpSum( [ choices[i][l] for l in range(24) ] ) == 2 # total 2 hours
        prob += pulp.lpSum([choices[i][l] for l in range(24,30)]) == 0
        for j in range(12):
            prob += choices[i][j] - choices[i][j+12] == 0 # MW or TT


    elif time_const[i] == "C11L12":
        prob += pulp.lpSum( [ choices[i][l] for l in range(len(choices[0])) ] ) == 3 # theory 1hour, lab 2hours
        temp_2_1 = [choices[i][l] for l in range(len(choices[i]))]
        temp_2_2 = [temp_2_1[l] for l in range(1, len(temp_2_1))]
        temp_2_1 = np.array(temp_2_1[:-1]); temp_2_2 = np.array(temp_2_2)
        temp2 = temp_2_1 + temp_2_2
        print(type(np.any(temp2 >= 2)))
        prob += bool(np.any(temp2 >= 2)) == True



    elif time_const[i] == "C21L12":

        # # Lecture
        prob += pulp.lpSum([lecture_temp[i][l] for l in range(24)]) == 2 # total course hour = 2
        prob += pulp.lpSum([lecture_temp[i][l] for l in range(24,30)]) == 0
        for j in range(12):
            prob += lecture_temp[i][j] - lecture_temp[i][j+12] == 0 # course time should be (Monday Wednesday) , (Tuesday, Thursday)
        # # Lab
        prob += pulp.lpSum( [lab_temp[i][l] for l in range(len(lab_temp[i]))] ) == 2
        temp_3_1 = [lab_temp[i][l] for l in range(len(lab_temp[i]))]
        temp_3_2 = [temp_3_1[l] for l in range(1, len(temp_3_1))]

        temp_3_1 = np.array(temp_3_1[:-1]); temp_3_2 = np.array(temp_3_2)
        temp_3 = temp_3_1 + temp_3_2
        prob += bool(np.any(temp_3 >= 2)) == True

        for j in range(30):
            prob += lecture_temp[i][j] + lab_temp[i][j] <= 1   # for overlap
        for j in range(30):
            prob += choices[i][j] == lecture_temp[i][j] + lab_temp[i][j] # choices


    elif time_const[i] == "C11L23":
        # # Lecture
        prob += pulp.lpSum([lecture_temp[i][l] for l in range(30)]) == 1 # total course time =1

        # # Lab
        prob += pulp.lpSum( [lab_temp[i][l] for l in range(12)] ) == 3
        prob += pulp.lpSum( [lab_temp[i][l] for l in range(12,30)] ) == 3
        temp_4_1 = [lab_temp[i][l] for l in range(12)]
        temp_4_2 = [temp_4_1[l] for l in range(1,12)]
        temp_4_3 = [temp_4_1[l] for l in range(2,12)]

        temp_4_1 = np.array(temp_4_1[:-2]); temp_4_2 = np.array(temp_4_2[:-1]); temp_4_3 = np.array(temp_4_3)
        temp_4_front = temp_4_1 + temp_4_2 + temp_4_3

        temp_4_4 = [lab_temp[i][l] for l in range(12,30)]
        temp_4_5 = [temp_4_4[l] for l in range(17)]
        temp_4_6 = [temp_4_4[l] for l in range(16)]

        temp_4_4 = np.array(temp_4_4[:-2]); temp_4_5 = np.array(temp_4_5[:-1]); temp_4_6 = np.array(temp_4_6)
        temp_4_back = temp_4_4 + temp_4_5 + temp_4_6

        prob += bool(np.any(temp_4_front>=3)) == True
        prob += bool(np.any(temp_4_back>=3)) == True

        for j in range(12):
            prob += lab_temp[i][j] - lab_temp[i][j+12] == 0

        for j in range(30):
            prob += lecture_temp[i][j] + lab_temp[i][j] <= 1   # for overlap

        for j in range(30):
            prob += choices[i][j] == lecture_temp[i][j] + lab_temp[i][j]
    elif time_const[i] == "C12":
        prob += pulp.lpSum( [ choices[i][l] for l in range(len(choices[i])) ] ) == 2 # theory 1 hour , lab 2 hours
        temp_5_1 = [choices[i][l] for l in range(len(choices[i]))]
        temp_5_2 = [temp_5_1[l] for l in range(1, len(temp_5_1))]
        temp_5_1 = np.array(temp_5_1[:-1]); temp_5_2 = np.array(temp_5_2)
        temp5 = temp_5_1 + temp_5_2
        prob += bool(np.any(temp5 >= 2)) == True

    else:
        print("????")

#################################################################################################



########################################## Constraint 2 ##################################################################

#  same time for one course, not same professor
professors = list(set(Data["Professor"][:22]))

for name in Data['Prof_extended'][:22]:
    if type(name) == str:
        professors.append(name)

professors.sort()
print(professors)

# Make an 22 x 18 matrix. Course x Professor
prof_const = np.zeros((22,len(professors)))

for i in range(len(COURSE_NAME)):
    for j in range(len(professors)):
        if Data['Professor'][i] == professors[j]:
            prof_const[i][j] = 1
        if Data["Prof_extended"][i] == professors[j]:
            prof_const[i][j] = 1




for j in TIMESLOT:
    time_conflict_matrix = np.zeros((len(COURSE), len(professors)), dtype = 'object')
    is_time_conflict = np.zeros((len(professors)), dtype = 'object')
    for i in COURSE:
        for k in range(len(professors)):
            time_conflict_matrix[i,k] = choices[i][j] * prof_const[i,k]

    for i in range(time_conflict_matrix.shape[0]):
        is_time_conflict += time_conflict_matrix[i, :]

    for i in range(len(is_time_conflict)):
        prob += is_time_conflict[i] <= 1

# #################################################################################################################

# ########################################## Constraint 3 ##########################################


for set in req_courses:
    for time in TIMESLOT:
        x = np.empty(30, dtype = 'object')
        for course in set:
            dic = choices[course]
            x += np.array(list(dic.values()))
        prob += bool(np.all(x <= 1)) == True


# #################################################################################################


########################################## Objective Function ##########################################

loss = []
for req_c in req_courses:
    req_courses_timeslot=[]
    for course in req_c:
        req_courses_timeslot.append(list(choices[course].values())) ## row vector 2x30
    z=[1]           ## add one in beginning
    for i in range(len(req_courses_timeslot[0])): # column
        temp=0
        for j in range(len(req_courses_timeslot)): # row
            temp = temp+req_courses_timeslot[j][i] # column wise sum
        z.append(temp)
    z.append(1)  ## add one in end
    z_idx = list(filter(lambda x:z[x]==1,range(len(z)))) # [1,1,1,0,0,1] - > 01235

    idx_distance = []
    for i in range(len(z_idx) - 1):
        idx_distance.append(z_idx[i+1] - z_idx[i])          # front index  - back index - > EX) 01235 - > 1 1 1 2

    loss.append(sum([(k/len(idx_distance))**2 for k in idx_distance]))


prob += pulp.lpSum(loss)

### Solve
prob.solve()

for j in range(22):
    print('\n')
    for i in range(30):
        print(pulp.value(choices[j][i]), end = '  ')
```