
INDENG164

Mini Project

AUTHOR

Jiayi Chen
3039832370
marcochen0720@berkeley.edu

May 2, 2024

Contents

1	Introduction	1
2	Modeling	1
2.1	First Thoughts on the Problem	1
2.2	Data Preprocessing	1
2.3	Basic Math Expression of the Model	2
2.4	More Factors Included	3
2.4.1	Time Penalty	4
2.4.2	Capacity Penalty	4
2.4.3	Distance Penalty	4
3	Discussion and Improvements	5
3.1	Results and evaluations	5
3.2	Limitations and Improvements	6
4	Appendix A	I

1 Introduction

The objective of the project is to develop optimization models for the effective allocation and scheduling of Berkeley IEOR courses for Fall 2024 based on the previous enrollment data and curricular trends, combined with real-life convenience and students' preferences.

Restatement of the problem

We have the following classrooms and the corresponding capacity:

Location	Capacity
ETCH 1174	45
ETCH 3107	45
STAN 105	292
VLSB 2040	158
CORY 277	132

Table 1: Classroom and corresponding capacities

The following courses will be offered: 115, 120, 130, 142A, 145, 150, 160, 162, 171, 172, 174.

Courses 130, 142A, 150, 171, 172 have twice-weekly lectures lasting 90 minutes and other courses have twice-weekly lectures lasting 1 hour.

We are asked to schedule these courses on M/W or T/TH between 8 am and 6 pm based on multiple factors.

2 Modeling

2.1 First Thoughts on the Problem

After searching for the related information on the website, we found that all the courses in the undergraduate department that last for an hour are scheduled on M/W, and the ones lasting for one hour and a half are scheduled on T/TH. As a result, to simplify the modeling, we divided the original problem into two parts based on the duration of the different courses. Then, we apply the approach of mixed integer problem by dividing the time into small slots and fill the courses into different slots.

2.2 Data Preprocessing

To determine the number of enrollments, we searched on the course website for the past 5-year data. by taking the mean and making some adjustments based on the trend(e.g. every year 142 reaches

its full capacity for different capacities, so we take the current stable capacity of 200 as the input) , we get the input data used in the coding,

Course_ID	23F	22F	21F	20F	19F	Mean	<i><u>Input</u></i>
115	117	83	100	93	73	93.2	<u>95</u>
120	137					137	<u>137</u>
130	64	52	53	48	35	50.4	<u>50</u>
142	202	200	204		148	188.5	<u>200</u>
145	10	14				12	<u>12</u>
150				26	22	24	<u>24</u>
160	78	90	86	102	100	91.2	<u>85</u>
162	79	70	56	79	90	74.8	<u>75</u>
171	27			40	56	41	<u>35</u>
172	150	63	65	129	82	97.8	<u>100</u>
174	72	77	74	61		71	<u>71</u>

Table 2: Past-year enrollment information and final input

2.3 Basic Math Expression of the Model

Decision Variables

Let $x_{c,r,t}$ be a binary variable where:

$$x_{c,r,t} = \begin{cases} 1 & \text{if course } c \text{ is assigned to room } r \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

Objective Function

Since we want only feasible solutions, we apply the 0 objective function.

Constraints

Ensure that the room capacity is not exceeded:

$$\sum_{c \in C} x_{c,r,t} \cdot \text{students}_c \leq \text{capacity}_r, \quad \forall r \in R, \forall t \in T$$

A room cannot host more than one class at the same time:

$$\sum_{c \in C} x_{c,r,t} \leq 1, \quad \forall r \in R, \forall t \in T$$

A class must be scheduled :

$$\sum_{r \in R} \sum_{t \in T} x_{c,r,t} = 1, \quad \forall c \in C$$

2.4 More Factors Included

To satisfy students' preference for a better commuting and rest experience, we introduce a penalty variable in the objective function, which would be minimized for optimization, to consider factors such as time and distance.

The objective function for the course scheduling problem is defined as:

$$\begin{aligned} \text{Minimize } Z = & \sum_{c \in C} \sum_{r \in R} \sum_{t \in T} x_{c,r,t} \times (\text{time_penalty}(t) + \text{capacity_penalty}(r)) \\ & + \sum_{c \in C} \sum_{(r_1, r_2) \in D} \sum_{(t_1, t_2) \in T^2} y_{c,r_1,r_2,t_1,t_2} \times \text{distance_penalty}(r_1, r_2) \end{aligned}$$

where:

- C is the set of courses.
- R is the set of rooms.
- T is the set of time slots.
- D is the set of room pairs for which distances are defined.
- T^2 represents all ordered pairs of consecutive time slots.
- $x_{c,r,t}$ is a binary variable that equals 1 if course c is scheduled in room r at time t ; otherwise, it is 0.
- y_{c,r_1,r_2,t_1,t_2} is a binary variable that equals 1 if course c is scheduled consecutively in room r_1 at time t_1 and in room r_2 at time t_2 ; otherwise, it is 0.
- $\text{time_penalty}(t)$ is a function that returns the early time and lunch time penalty if time t is in the above considerations.
- $\text{time_penalty}(t)$ is a function that returns the capacity penalty calculated by multiplying the rate of \$1/seat for every 30 minutes and the capacity of the classroom.
- $\text{distance_penalty}(r_1, r_2)$ is the penalty factor for scheduling courses consecutively in rooms r_1 and r_2 that are geographically distant.

2.4.1 Time Penalty

We set an early class(8 am) and lunch class (12 pm) penalty and minimize it:

$$\text{Minimize } \sum_{c \in C} \sum_{r \in R} \sum_{t \in \text{time}} x_{c,r,t}$$

2.4.2 Capacity Penalty

We incorporate a capacity-based penalty to encourage the use of smaller classrooms unless larger ones are necessary. This penalty is proportional to the capacity of the room used:

$$\text{Minimize } \sum_{c \in C} \sum_{r \in R} \sum_{t \in T} x_{c,r,t} \cdot \text{capacity}_r$$

This formulation adds a cost for using each room that is proportional to its capacity, penalizing the use of larger rooms more heavily unless required by the enrollment numbers of the courses.

2.4.3 Distance Penalty

We minimize the penalty of consecutive classes in distant buildings based on the real-life distance between two buildings as shown in the table:

Location 1	Location 2	Distance(0.1 mile)
ETCH	STAN	2
	VLSB	6
	CORY	1
STAN	VLSB	2
	CORY	2
VLSB	CORY	4

Table 3: Distance between two buildings

3 Discussion and Improvements

3.1 Results and evaluations

Courses	Location	Day	Time
145	ETCH3107	M/W	09:00
160	CORY277	M/W	10:00
115	CORY277	M/W	11:00
174	CORY277	M/W	13:00
162	CORY277	M/W	14:00
120	VLSB2040	M/W	16:00
150	ETCH3107	TU/TH	09:30
171	ETCH1174	TU/TH	11:00
130	CORY277	TU/TH	12:30
142A	STAN105	TU/TH	14:00
172	VLSB2040	TU/TH	15:30

Table 4: Final schedule

After coding and modeling using Pyomo, we obtained the result in [4](#).

It is shown that on Monday and Wednesday, the class begins in Etcheverry Hall, then goes to CORY Hall, and finally to Valley Life Science Building. The route is well-planned, commuting time for CORY and VLSB considered as well as early and lunchtime prevented.

For class on Tuesday and Thursday, we can only satisfy one of the early and lunchtime constraints due to the limited time block(given considerations of the starting time for 1.5h classes goes as 8 am,9.30 am, and so on), in which case we prioritize early class. The route is also reasonable, going as ETCH-CORY-STAN-VLSB.

All of the classes are scheduled without conflict and considerations for students' preferences and real-life convenience are satisfied. We believe this is a good version of the schedule.

3.2 Limitations and Improvements

There are also some limitations and potential improvements for a better model of course scheduling:

- **Stochastic and Robust Concerns**

The past-year data is incomplete. We can see that the number of enrollments and scheduled classes change due to different department policies and real-world trends. These factors lead to difficulty dealing with input data and uncertainties and should be disclosed more clearly.

- **Difficulties to Determine Proper Parameters**

As we mentioned above, we have different parameters for penalty terms. However, it is difficult to set proper weights or values for different terms due to personal preferences and limited data.

- **Building Models with Better Generalization Ability**

The courses considered are limited to the ones in the undergraduate part, while in fact, students may enroll in graduate-level courses and teachers may teach undergrad, grad, or maybe both. What's more, the current model follows a convention of separate dates for classes with different durations. When more classes are considered, there grows a need for overlapping classes and a mixed version in which case classes of different durations are scheduled on the same day. We can expect a more comprehensive model given more information.

4 Appendix A

Listing 1: Pyomo Code

```
1 from pyomo.environ import *
2 import itertools
3
4 # Data Definitions
5 courses = ['115', '120', '145', '160', '162', '174']
6 enrollments = {'115': 95, '120': 137, '145': 12, '160': 85, '162': 75, '174':
7               71}
8 # courses = ['130', '142A', '150', '171', '172']
9 # enrollments = {'130': 50, '142A': 200, '150': 24, '171': 35, '172': 100}
10 rooms = {'ETCH1174': 45, 'ETCH3107': 45, 'STAN105': 292, 'VLSB2040': 158, '
11          CORY277': 132}
12 times = [f"{hour}:00" for hour in range(8, 18)]
13 prohibited_time = "12:00" # No classes at this specific time
14 room_capacity_penalty = 1.0 / 30 # $1 per seat per 30 minutes
15 early_times = [f"{hour}:00" for hour in range(8, 9)]
16 early_time_penalty = 10
17 distance_penalty = 5
18
19 # Assumed distances between buildings (matrix)
20 distances = {
21     ('ETCH1174', 'ETCH3107'): 0,
22     ('ETCH1174', 'STAN105'): 2,
23     ('ETCH1174', 'VLSB2040'): 6,
24     ('ETCH1174', 'CORY277'): 1,
25     ('ETCH3107', 'STAN105'): 2,
26     ('ETCH3107', 'VLSB2040'): 6,
27     ('ETCH3107', 'CORY277'): 1,
28     ('STAN105', 'VLSB2040'): 4,
29     ('STAN105', 'CORY277'): 2,
30     ('VLSB2040', 'CORY277'): 4,
31 }
32 # Make distances symmetric
33 for (r1, r2), d in list(distances.items()):
34     distances[(r2, r1)] = d
35
36 # Model
```

```
35 model = ConcreteModel()
36
37 # Indices
38 model.courses = Set(initialize=courses)
39 model.rooms = Set(initialize=rooms.keys())
40 model.times = Set(initialize=times)
41
42 # Decision Variables
43 model.x = Var(model.courses, model.rooms, model.times, within=Binary)
44
45 # Define variable y for consecutive course room assignment, assuming it needs
    different room and time pairs
46 model.y = Var(model.courses, model.rooms, model.rooms, model.times, model.times
    , within=Binary)
47
48 # Objective Function
49 def objective_rule(model):
50     total_penalty = sum(model.x[c, r, t] * (early_time_penalty if t in
        early_times else 1) for c in courses for r in rooms for t in times)
51     total_penalty += sum(model.x[c, r, t] * room_capacity_penalty * rooms[r]
        for c in courses for r in rooms for t in times)
52     for c in courses:
53         for t1, t2 in itertools.permutations(times, 2):
54             if abs(int(t1.split(':')[0]) - int(t2.split(':')[0])) == 1: #
                Consecutive times
55                 for (r1, r2), dist in distances.items():
56                     total_penalty += model.y[c, r1, r2, t1, t2] * (
                        distance_penalty * dist)
57     return total_penalty
58 model.objective = Objective(rule=objective_rule, sense=minimize)
59
60 # Constraints
61 def room_capacity_rule(model, r, t):
62     return sum(model.x[c, r, t] * enrollments[c] for c in courses) <= rooms[r]
63 model.room_capacity = Constraint(model.rooms, model.times, rule=
    room_capacity_rule)
64
65 def course_schedule_rule(model, c):
66     return sum(model.x[c, r, t] for r in rooms for t in times) == 1 # Each
    course once a week
```

```
67 model.course_schedule = Constraint(model.courses, rule=course_schedule_rule)
68
69 def no_overlap_rule(model, r, t):
70     return sum(model.x[c, r, t] for c in courses) <= 1
71 model.no_overlap = Constraint(model.rooms, model.times, rule=no_overlap_rule)
72
73 def consecutive_assignment_rule(model, c, r1, r2, t1, t2):
74     return model.y[c, r1, r2, t1, t2] <= model.x[c, r1, t1]
75     return model.y[c, r1, r2, t1, t2] <= model.x[c, r2, t2]
76     return model.y[c, r1, r2, t1, t2] >= model.x[c, r1, t1] + model.x[c, r2, t2
    ] - 1
77 model.consecutive_assignment = Constraint(model.courses, model.rooms, model.
    rooms, model.times, model.times, rule=consecutive_assignment_rule)
78
79 # Solver configuration
80 solver = SolverFactory('cbc')
81 result = solver.solve(model, tee=True)
82
83 # Display results
84 for c in courses:
85     for r in rooms:
86         for t in times:
87             if model.x[c, r, t].value == 1:
88                 print(f"Course {c} is scheduled in room {r} at time {t}")
```