Berkeley Industrial Engineering & Operations Research
COLLEGE OF ENGINEERING

INDENG164

Mini Project

AUTHOR

Jiayi Chen
3039832370
marcochen0720@berkeley.edu

July 27, 2024

Contents

1 Introduction

The client for this project is a bus agency. We will call it Bay Area Buses - covering the same service area as clients like AC Transit, Muni, etc, but in a fictional universe. The agency would like to develop a database it can use to keep track of several interrelated facets of its operations.

First and foremost, the agency would like to keep track of where its buses are scheduled to operate. It would like to ensure that every single trip is assigned exactly one bus to operate it, and one driver to drive that bus. And in this way, it would be able to know in one place where each bus and driver should be at any given point in time.

The agency will also be tracking maintenance needs. It will be able to easily access information about when the last time each bus has had any of a number of services done, and when it will be needed next, thus able to schedule maintenance ops well in advance.

The agency will also keep track of its depots, and the mix of buses assigned to each. It will know where those buses are going to and from each morning (dead-heading). It will know where it has spare capacity to base new buses.

The agency will be able to track its employees across its departments - drivers, maintenance, and perhaps white-collar workers as well (though we omit these from our initial consideration). It will be able to see who has how much experience, and is therefore vested into its pension system.

The agency will be able to track all its stops, and see which has a bench or a full shelter. It will be able to see which stops have more boardings, and perhaps prioritize those for improvements.

The agency will need to gather abundant data for this database. For each of its bus stops, it would need their addresses, the amenities available, and which bus routes serve them.

For each bus, it would need to know that busâs capacity, what fuel type it uses, its license plate, its age, and what depot it is assigned to. For each bus route, we will need to know what stops it starts and ends at, and where else it stops along the way. We would also need records of when the last time it was maintained is.

For each employee, we would need to know their social security number (SSID), how many years theyâve worked for the agency, and what depot theyâre based at. We will also collect the driver licenses of our bus drivers, as this info may be useful. For each type of maintenance task, we will need to know how frequently it is done.

Given this, the agency will be well-equipped to assign its buses, drivers, and maintenance workers to tasks that utilize them efficiently.

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

Table 1: 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:

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

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.
- time_penalty(t) is a function that returns the early time and lunch time penalty if time t is in the above considerations.
- 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.
- 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:

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

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:

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

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 2: 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 3: Final schedule

After coding and modeling using Pyomo, we obtained the result in 3.

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

```
from pyomo.environ import *
import itertools

# Data Definitions
courses = ['115', '120', '145', '160', '162', '174']
enrollments = {'115': 95, '120': 137, '145': 12, '160': 85, '162': 75, '174': 71}
# courses = ['130', '142A', '150', '171', '172'']
# enrollments = {'130': 50, '142A': 200, '150': 24, '171': 35, '172': 100}
rooms = {'ETCH1174': 45, 'ETCH3107': 45, 'STAN105': 292, 'VLSB2040': 158, 'CORY277': 132}
times = [f"{hour}:00" for hour in range(8, 18)]
prohibited_time = "12:00"  # No classes at this specific time
room_capacity_penalty = 1.0 / 30  # $1 per seat per 30 minutes
early_times = [f"{hour}:00" for hour in range(8, 9)]
early_time_penalty = 10
distance_penalty = 5

# Assumed distances between buildings (matrix)
distances = {
    ('ETCH1174', 'ETCH3107'): 0,
    ('ETCH1174', 'STAN105'): 2,
    ('ETCH1174', 'VLSB2040'): 6,
    ('ETCH1174', 'CORY277'): 1,
    ('ETCH3107', 'STAN105'): 2,
    ('ETCH3107', 'VLSB2040'): 6,
    ('ETCH3107', 'CORY277'): 1,
    ('STAN105', 'VLSB2040'): 4,
    ('STAN105', 'CORY277'): 2,
    ('VLSB2040', 'CORY277'): 4,
}
# Make distances symmetric
for (r1, r2), d in list(distances.items()):
    distances[(r2, r1)] = d

# Model
```

```
model = ConcreteModel()

# Indices
model.courses = Set(initialize=courses)
model.rooms = Set(initialize=rooms.keys())
model.times = Set(initialize=times)

# Decision Variables
model.x = Var(model.courses, model.rooms, model.times, within=Binary)

# Define variable y for consecutive course room assignment, assuming it needs different room and time pairs
model.y = Var(model.courses, model.rooms, model.rooms, model.times, model.times, within=Binary)

# Objective Function
def objective_rule(model):
    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)
    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)
    for c in courses:
        for t1, t2 in itertools.permutations(times, 2):
            if abs(int(t1.split(':')[0]) - int(t2.split(':')[0])) == 1:  # Consecutive times
                for (r1, r2), dist in distances.items():
                    total_penalty += model.y[c, r1, r2, t1, t2] * (distance_penalty * dist)
    return total_penalty
model.objective = Objective(rule=objective_rule, sense=minimize)

# Constraints
def room_capacity_rule(model, r, t):
    return sum(model.x[c, r, t] * enrollments[c] for c in courses) <= rooms[r]
model.room_capacity = Constraint(model.rooms, model.times, rule=room_capacity_rule)

def course_schedule_rule(model, c):
    return sum(model.x[c, r, t] for r in rooms for t in times) == 1  # Each course once a week
```

```
model.course_schedule = Constraint(model.courses, rule=course_schedule_rule)

def no_overlap_rule(model, r, t):
    return sum(model.x[c, r, t] for c in courses) <= 1
model.no_overlap = Constraint(model.rooms, model.times, rule=no_overlap_rule)

def consecutive_assignment_rule(model, c, r1, r2, t1, t2):
    return model.y[c, r1, r2, t1, t2] <= model.x[c, r1, t1]
    return model.y[c, r1, r2, t1, t2] <= model.x[c, r2, t2]
    return model.y[c, r1, r2, t1, t2] >= model.x[c, r1, t1] + model.x[c, r2, t2
        ] - 1
model.consecutive_assignment = Constraint(model.courses, model.rooms, model.
    rooms, model.times, model.times, rule=consecutive_assignment_rule)

# Solver configuration
solver = SolverFactory('cbc')
result = solver.solve(model, tee=True)

# Display results
for c in courses:
    for r in rooms:
        for t in times:
            if model.x[c, r, t].value == 1:
                print(f"Course {c} is scheduled in room {r} at time {t}")
```