CMU Math Department TA Assignment Problem
Operations Research II, 21-393

Selena Ma, Xiaohua Shi, Katherine Li, Kimberly Bau, Jong Won Lim
Carnegie Mellon University

Abstract— **This project mainly concerns the TA (teaching assistant) assignment at the Mathematics Department of Carnegie Mellon University (CMU). Every semester, the math department at CMU needs to recruit and assign students to work as teaching assistants for over 20 courses for the upcoming semester. While the traditional assignment is done by hand, there are many opportunities for us to improve the process to set up the assignment more effectively. In this paper, we're going to work with student application data and schedule data to formulate an optimization problem that aim to minimize assignment cost or maximize faculty and applicants' satisfaction value. We will briefly discuss our feature engineering and optimization method, of which the source code is publicly available at `https://github.com/pawsagainsthoomanity/TA_Assignment_Problem`.**

I. INTRODUCTION

Every semester, the CMU math department faces the needs to recruit and assign both graduate and undergraduate teaching assistants to approximately 30 courses department-wide. Although it has been a routine, there are still persisting problems that make it a complicated task: specifically, aligning students' needs (course preference, students' schedules) with course requirement (GPA, recitation time). Since the selection process has been manual and no computer-aided means is available at the moment, we decided to develop an automated tool that helps alleviate the workload and facilitate the faculties make decisions on interview scheduling, preferable student ranking, etc.

A. Current State

After interviewing Professor Howell, the faculty member who is responsible for TA selection, we gathered the following insights: (1) Every semester, approximately 90 undergraduate and 30 graduate students apply for TA positions for 30 courses; (2) Professor Howell first manually picks eligible applicants, then matches the applicants' schedules to available recitations one by one, which needs frequent adjustments.

The interview reveals some pain points that the current method is laborious in terms of scheduling, ranking, and matching. Therefore, we propose an assignment problem solver that tailors to the needs of the math department.

B. Goals

Our end goal is to formulate the TA assignment to a non-regular assignment problem. It should provide an easy-to-use interface, ranking of preference, filtering functionalities, and most importantly providing an assignment solution given students' preferences, course schedules, students' schedules, grades and interview feedback.

II. DATA COLLECTION

Our project requires a lot of data, used as inputs or constraints in our assignment problem. We received students application information from professor Jason Howell in our meetings, which contains the courses that are hiring TA, as well as students' preferences. However due to privacy concerns some information such as student names was concealed. The table below is an example of the preferences data.

#ID	Course Interested In Teaching
1	21-111: Calculus I , 21-112: Calculus II , 21-120: Differential and Integral Calculus
2	21-228: Discrete Mathematics , 21-241: Matrices and Linear Transformations
3	21-301: Combinatorics , 21-341: Linear Algebra , 21-355: Principles of Real Analysis I
4	21-370: Discrete Time Finance
5	21-241: Matrices and Linear Transformations
6	21-256: Multivariate Analysis
7	21-217: Concepts of Mathematics, 21-241 Matrices and Linear Transformations, 21-259: Calculus in Three Dimensions
...	... ...

TABLE I: Application Information

This data gave us information such that: there are in total of 92 undergraduate applicants, and there are in total of 23 distinct courses that are hiring TAs. With this initial data in hand we retrieved some information from the Schedule of Course (SOC) system and constructed a total of 4 data structures for our program: (1) Recitation Schedules (**R**); (2) Applicants Schedules (**S**); (3) Applicants Grades (**G**); (4) Applicants Preferences (**P**)

A. Recitation Schedule Data

We only consider undergraduate math courses in our problem, so there were a total of 23 courses. Some courses had multiple recitation sections, so the total number of recitation sections to consider was 44. We constructed a data structure of the recitation schedules based on the information in the CMU website on the course schedules for Spring 2020.

A partial screenshot of the data we constructed for the recitation schedules is on the next page. We basically converted the information into a 2D matrix where each of

the single entries is a list of size 5. Each row in the matrix represents a recitation section, where if a TA is assigned to that section, he/she would be obligated to teach during that duration. The columns represent the time of a single day, divided into 1 hour blocks. The list of size 5 represents each weekday of the week, Monday through Friday.

#Course	8:30-9:30	9:30-10:30	10:30-11:30	11:30 12:30	...
21-111	[0,1,0,1,0]	[0,0,0,0,0]	[0,0,0,0,0]	[0,0,0,0,0]	...
21-112	[0,0,0,0,0]	[0,0,0,0,0]	[0,0,0,0,0]	[0,0,0,0,0]	...
21-120	[0,1,0,1,0]	[0,1,0,1,0]	[0,0,0,0,0]	[0,0,0,0,0]	...
21-122	[0,1,0,1,0]	[0,1,0,1,0]	[0,1,0,1,0]	[0,1,0,1,0]	...
21-124	[0,0,0,0,0]	[0,0,0,0,0]	[0,1,0,1,0]	[0,1,0,1,0]	...
...	...	...	...	...	...

TABLE II: Course Schedule Data

For example, the top left entry [0,1,0,1,0] for row Calculus 1 and column 8:30-9:30 means that a section for a recitation for Calculus 1 meets from 8:30-9:30 on Tuesday and Thursday.

B. Applicant Schedules Data

We needed to remove applicants that have other academic obligations during a recitation period, so we also needed all the applicants academic schedules. We organized the application schedules into the same data structure we used for the course schedules. However unlike the course schedules, we did not have concrete data for applicant schedules due to privacy issues. So for our program purposes, we created random schedules for each applicant. The structure is the same as course schedule we previously discussed.

C. Applicant Grades Data

Much like the applicant schedules, we were given no information regarding the applicants' grades of the math courses. However, we felt that the grades were essential for the assignment problem since they provide an objective measure of part of the applicant's ability to perform well for the job.

At first, we generated the grades uniformly randomly from A to B. However we felt that this was rather an unrealistic generation, since most people who apply for the position apply because they received relatively good grades. So we generated the grades for the 100 and 200 level courses (which were the majority of the courses) uniformly randomly from A to B, and the others from A to C.

#Course	Stu1	Stu2	Stu3	Stu4	Stu5	Stu6	...
21-111	3	3	4	3	4	3	...
21-112	3	4	4	3	4	3	...
21-120	4	4	3	3	4	4	...
...	...	...	...	...	...	...	...

TABLE III: Applicants Grade Data

Above is part of the data we constructed. Each entry is a number from 0 to 4, representing the grades R to A respectively. Each row is a math course, and the column is the applicant.

D. Applicant Preferences Data

Finally we have created a data structure representing the course preferences for each applicant. We received a list of courses that the applicants were interested to teach from Professor Howell, but we additionally gave each course a preference score for each applicant.

#Course	Stu1	Stu2	Stu3	Stu4	Stu5	Stu6	...
21-111	0	1	0	0	4	3	...
21-112	0	0	0	5	1	0	...
21-120	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...

TABLE IV: Applicants Preference Data

The rows represent the courses, the columns represent each applicant, and finally each entry represents the preference score. The preference scores are from 0-5. A preference score of 0 means that the applicant did not apply to teach for the course. Regarding scores from 1 to 5, a lower score means that applicant prefers the course more. The process of calculating the scores were as follows: we first give courses that the applicant did not apply to all score 0. Then we randomly sorted the courses that he/she did apply to and ranked them in ascending order. However, since the number of courses applied were different for each applicant, we normalized the generated rankings by the total number of applied courses so that all rankings ranged from 1 to 5.

III. Problem Formulation

Let's formulate the previous dataset with the following variables

$$R_{ij} = \begin{cases} 1, & \text{if recitation } i \text{ is taking place at time } j \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$(i = 1, 2, ...43, j = 1, 2, ...9)$$

$$S_{ij} = \begin{cases} 1, & \text{if student } i \text{ is available at time } j \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

$$(i = 1, 2, ...92, j = 1, 2, ...9)$$

G_{ij}= student j's grade for recitation i, grade = 0-4

P_{ij}= student j's preferences for recitation i, pref = 0-5

$$(i = 1, ...43, j = 1, ...92)$$

A. Feature Engineering

There are many operations we can perform on the data structures to make the data more valuable for us. First of all, we use Recitation Schedule and Applicant Schedule to create a new array that tells us a student's availability for each classes.

$$\text{Availability}_{ij} = \begin{cases} 1, & \text{if } (RS^T)_{ij} = 1 \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

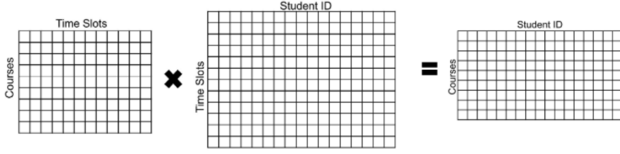

Fig. 1: Computation of Availability Array

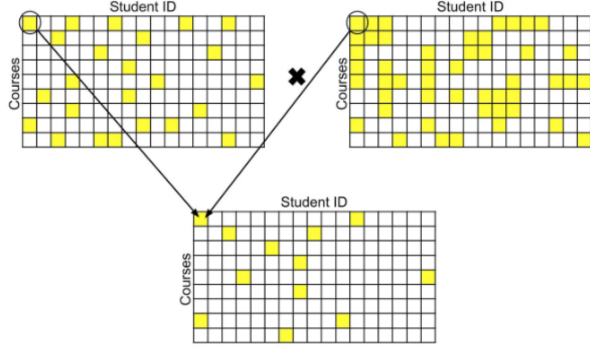

Fig. 2: Computation of Eligibility Array

$$(i = 1, 2, ...43, j = 1, 2, ...92)$$

After that, we formulate the grade constraint by getting an entry-wise product of the availability matrix and grade matrix. We will treat the entries of students who - although available and interested in teaching a course - received a grade lower than B, as not eligible. This operation gives a finalist of eligible students.

$$\text{Eligibility}_{ij} = \begin{cases} 1, & \text{if } ((RA^T) \circ G)_{ij} \geq 3 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$(i = 1, 2, ...43, j = 1, 2, ...92)$$

It's important to know that the preferences value is directly related to a student's satisfaction. That is, the lower the preference value, the higher the satisfaction. So in order to formulate the objective function, we want to find out the preferences among the students who are eligible to teach by, again, taking the entry-wise product of eligibility matrix and the preferences array. With the new matrix, maximizing satisfaction can be effectively expressed with minimizing preferences score.

We introduce a new variable "Cost", which is 10 times the value of preferences. Because we want to create a larger gap between distinct satisfaction/preference values. By doing so, we are also allowing Prof.Howell to come into the picture. For example, after he interviews a few candidates, if he decides to assign student j to teach recitation i, we will shrink the corresponding preference value by 90%, which can almost guarantee the assignment.

$$\text{Cost} = 10 * (\text{Eligibility} \circ \text{Preferences})$$

B. Intermediate Output

With the cost matrix, there already are a few valuable insights we are able to produce. For example, we can return a list of students who are capable to teach over 4 courses, in which case it's more necessary for prof.Howell to interview these candidates. In addition, we are able to identify the classes for which we found no eligible applicants; We can also identify the applicants that are not eligible to teach any classes. More importantly, because these courses and/or students are no longer part of the assignment problem, we need to set them as dummy variables so that our solver can run without error.

Algorithm 1 Feature Engineering Output

0: **procedure** OUTPUT1($Cost$)
0: **Initialize** L1, L2
0: $m = nrow(Cost)$
0: $n = ncol(Cost)$
0: **Initialize** C'
0: $(C')_{ij} = 1 \ if \ Cost_{ij} > 0 \ else \ 0$
0: $C1 \leftarrow \sum_i^m (Cost_{ij})$
0: $C2 \leftarrow \sum_j^n (Cost_{ij})$
0: $C3 \leftarrow \sum_i^m (C'_{ij})$
0: **for** `i in (1,2, ... m)` **do**
0: **if** $C2_i == 0$ **then**
0: $Cost_{ij}$ `= 999999 for all j`
0: **end if**
0: **end for**
0: **for** `j in (1, 2, ... n)` **do**
0: **if** $C1_j == 0$ **then**
0: `L1.append(j)`
0: $Cost_{ij}$ `= 999999 for all i`
0: **end if**
0: **if** $C3_j > 4$ **then**
0: `L2 append j`
0: **end if**
0: **end for**
0: **return** L1, L2, Cost
0: **end procedure**=0

A sample output from the program is the following:

```
Hi Jason, we are unable to find
qualified or available candidates
for the following courses:
Course #21-124
Course #21-236
Course #21-238
Course #21-254
Course #21-261
Course #21-269
Course #21-292
Course #21-369
Course #21-469

Additionally,
```

```
Found 5 candidates for course 21-111.
Found 15 candidates for course 21-112.
Found 16 candidates for course 21-120.
Found 21 candidates for course 21-122.
Found 27 candidates for course 21-127.
Found 11 candidates for course 21-228.
Found 9 candidates for course 21-240.
Found 23 candidates for course 21-241.
Found 1 candidates for course 21-242.
Found 8 candidates for course 21-256.
Found 9 candidates for course 21-259.
Found 6 candidates for course 21-260.
Found 1 candidates for course 21-268.
Found 1 candidates for course 21-270.

Here are the people who are qualified
to teach more than 4 courses.
Candidate #16: 21-111; 21-112;
21-120; 21-122; 21-256;
Candidate #21: 21-111; 21-112;
21-120; 21-228; 21-240;
Candidate #31: 21-111; 21-112;
21-120; 21-122; 21-241; 21-259;
Candidate #54: 21-112; 21-122;
21-127; 21-228; 21-241; 21-260;
Candidate #79: 21-111; 21-112;
21-127; 21-228; 21-240; 21-241; 21-260;

You may consider an interview for them.
If you do, please fill out the
evaluation form with the COURSE NUMBER
of your preferred assignment in the
column next to student ID.
 Good Luck!
```

With this information, Jason Howell can select students to interview and then dynamically change the value from an auto-generated evaluation form to express his preferences. Prof.Howell will have the ability to practically determine the assignments of many students when he fills out the evaluation form. Let's say Prof.Howell has completed the evaluation file after the interviews as such:

#Student	Course
16	21-256
21	21-228
31	21-241
54	21-260
79	21-127

TABLE V: Prof. Howell's Evaluation

As we've discussed earlier, the cost of assigning a student to a course will shrink by 90% if prof.Howell prefer the assignment. Thus, after the evaluation, we can modify the cost matrix again using the following method:

Algorithm 2 Post Interview Cost

0: **procedure** NEWCOST($Cost, Eval, courseDict$)
0: **for** `row in Eval` **do**
0: `i` $\leftarrow$ `courseDict[`row_1`]`
0: `j` $\leftarrow row_0$
0: `curr` $\leftarrow Cost_{ij}$
0: $Cost_{ij} \leftarrow$ `curr/10`
0: **end for**
0: **return** Cost
0: **end procedure**=0

C. Problem Setup

Let M be the assignment matrix, M_{ij} = 1 if student j is assigned to lead recitation i. We want to

$$Min. \quad \sum_{i}^{43}\sum_{j}^{92}(Cost \circ M)_{ij}$$

$$s.t. \sum_{i}^{92} M_{ij} = 1 \ (each\ recitation\ has\ one\ TA)$$

$$\sum_{j}^{43} M_{ij} = 1 \ (each\ TA\ teaches\ one\ recitation)$$

IV. SOLUTION

A. Tools

We tried to use Google Sheets to solve the assignment problem first. By using integer programming tool, the number of variables is too big and the runtime for getting the solution is quite slow. Also, the result is still not ideal, saying that the constraints are not linear.

B. Assignment Solver

We then switched to the python distribution of Google OR-Tools, an open source OR package that handles vehicle routing, flows, integer and linear programming, and constraint programming. We customized its assignment solver in following ways:

(i) The solver is built for minimization problem, whereas ours is a maximization. Hence, we re-ranked the cost matrix and fit it to the solver.

(ii) The solver is designed for standard assignment problems that take in equal number of workers and tasks. However, in our case, we have more applicants than available recitations. In order to align with the input format, we created dummy applicants with extreme costs that would not make the algorithm consider them as eligible applicants.

Our modification is robust - it gives sensible successful assignments in very short run-time (less than 0.05 seconds).

C. *Output*

```
Hello Professor Howell! Here’s your TA assignment result.
Applicant #1 is assigned to Recitation 21-228-2.  Cost = 10
Applicant #2 cannot be assigned to any recitation.
Applicant #3 cannot be assigned to any recitation.
Applicant #4 is assigned to Recitation 21-260-1.  Cost = 20
Applicant #5 is assigned to Recitation 21-112.  Cost = 10
Applicant #6 cannot be assigned to any recitation.
Applicant #7 cannot be assigned to any recitation.
Applicant #8 cannot be assigned to any recitation.
Applicant #9 cannot be assigned to any recitation.
Applicant #10 cannot be assigned to any recitation.
Applicant #11 cannot be assigned to any recitation.
Applicant #12 is assigned to Recitation 21-241-1.  Cost = 10
Applicant #13 cannot be assigned to any recitation.
Applicant #14 cannot be assigned to any recitation.
Applicant #15 cannot be assigned to any recitation.
Applicant #16 is assigned to Recitation 21-111.  Cost = 10
Applicant #17 cannot be assigned to any recitation.
Applicant #18 cannot be assigned to any recitation.
Applicant #19 cannot be assigned to any recitation.
Applicant #20 is assigned to Recitation 21-259-3.  Cost = 10
Applicant #21 is assigned to Recitation 21-228-1.  Cost = 3
Applicant #22 cannot be assigned to any recitation.
Applicant #23 cannot be assigned to any recitation.
Applicant #24 is assigned to Recitation 21-241-3.  Cost = 20
Applicant #25 is assigned to Recitation 21-127-1.  Cost = 20
Applicant #26 cannot be assigned to any recitation.
Applicant #27 cannot be assigned to any recitation.
Applicant #28 is assigned to Recitation 21-268.  Cost = 50
Applicant #29 cannot be assigned to any recitation.
Applicant #30 is assigned to Recitation 21-256-3.  Cost = 10
Applicant #31 is assigned to Recitation 21-241-2.  Cost = 1
Applicant #32 cannot be assigned to any recitation.
Applicant #33 is assigned to Recitation 21-259-4.  Cost = 20
Applicant #34 cannot be assigned to any recitation.
Applicant #35 cannot be assigned to any recitation.
Applicant #36 is assigned to Recitation 21-127-4.  Cost = 20
Applicant #37 is assigned to Recitation 21-122-3.  Cost = 10
Applicant #38 cannot be assigned to any recitation.
Applicant #39 cannot be assigned to any recitation.
Applicant #40 cannot be assigned to any recitation.
Applicant #41 cannot be assigned to any recitation.
Applicant #42 cannot be assigned to any recitation.
Applicant #43 cannot be assigned to any recitation.
Applicant #44 cannot be assigned to any recitation.
Applicant #45 cannot be assigned to any recitation.
Applicant #46 is assigned to Recitation 21-240.  Cost = 10
Applicant #47 cannot be assigned to any recitation.
Applicant #48 cannot be assigned to any recitation.
Applicant #49 is assigned to Recitation 21-270-3.  Cost = 10
Applicant #50 is assigned to Recitation 21-122-2.  Cost = 10
Applicant #51 cannot be assigned to any recitation.
Applicant #52 cannot be assigned to any recitation.
Applicant #53 is assigned to Recitation 21-259-1.  Cost = 10
```

```
Applicant #54 is assigned to Recitation 21-260-2.  Cost = 3
Applicant #55 cannot be assigned to any recitation.
Applicant #56 cannot be assigned to any recitation.
Applicant #57 cannot be assigned to any recitation.
Applicant #58 is assigned to Recitation 21-122-4.  Cost = 20
Applicant #59 cannot be assigned to any recitation.
Applicant #60 is assigned to Recitation 21-256-2.  Cost = 20
Applicant #61 cannot be assigned to any recitation.
Applicant #62 is assigned to Recitation 21-256-1.  Cost = 20
Applicant #63 cannot be assigned to any recitation.
Applicant #64 cannot be assigned to any recitation.
Applicant #65 cannot be assigned to any recitation.
Applicant #66 cannot be assigned to any recitation.
Applicant #67 cannot be assigned to any recitation.
Applicant #68 cannot be assigned to any recitation.
Applicant #69 cannot be assigned to any recitation.
Applicant #70 is assigned to Recitation 21-127-5.  Cost = 20
Applicant #71 is assigned to Recitation 21-122-1.  Cost = 20
Applicant #72 is assigned to Recitation 21-242.  Cost = 40
Applicant #73 cannot be assigned to any recitation.
Applicant #74 cannot be assigned to any recitation.
Applicant #75 is assigned to Recitation 21-120.  Cost = 10
Applicant #76 is assigned to Recitation 21-259-2.  Cost = 20
Applicant #77 cannot be assigned to any recitation.
Applicant #78 cannot be assigned to any recitation.
Applicant #79 is assigned to Recitation 21-127-3.  Cost = 4
Applicant #80 cannot be assigned to any recitation.
Applicant #81 cannot be assigned to any recitation.
Applicant #82 cannot be assigned to any recitation.
Applicant #83 cannot be assigned to any recitation.
Applicant #84 cannot be assigned to any recitation.
Applicant #85 cannot be assigned to any recitation.
Applicant #86 cannot be assigned to any recitation.
Applicant #87 is assigned to Recitation 21-127-2.  Cost = 10
Applicant #88 cannot be assigned to any recitation.
Applicant #89 cannot be assigned to any recitation.
Applicant #90 cannot be assigned to any recitation.
Applicant #91 cannot be assigned to any recitation.
Applicant #92 cannot be assigned to any recitation.
```

V. Conclusion

We have by now completed a program that report list of courses with no candidates, allows for Jason Howell to dynamically interview candidates to modify cost value, and returns a complete assignment.

A. Value Of Use

One of the more pressing matters, according to Jason Howell, when assigning TAs is scheduling conflicts. Before the TAs are assigned in our program Prof. Howell is able to see the number of eligible students that are available to teach a certain recitation and which students those are. This solves the scheduling conflict problem in a matter of seconds. This list also ranks the eligible students based on their grades and preferences. Prof. Howell can then use this information to decide which students he wants to interview. Once done his personal interview score can be added into the program and run to get an assignment. This program can make the application cycle easier and smoother for Prof. Howell.

B. Limitations and Future Opportunities

Although we have created a model to use for TA assignments there were many limitations that prevented us from modeling real life data. Due to privacy concerns, we were not able to gather actual data such as student grades, schedules, and preference rankings. In real life the person using this model would be Jason Howell, who does have access to all this information, so that would make this program more practical.

We've completed a comprehensive GitHub documentation that serves as a user guide for Prof.Howell so it can be as simple as possible when he uses our tool. However, Command line is not the most user-friendly interface, it leaves us some room to further improve on this model in the future we would try to create a web based solution to use.

Another limitation in our model are the constraints. The current program runs a 1 to 1 match up between the students and recitations. This means only one student gets assigned to one course recitation. In real life, students will often teach multiple recitations within the same course or more than one TA will be assigned to a section. In the future we would be able to change this in the program to model the real life situation. Furthermore, TAs need to include extra hours to hold office hours. These need to be convenient to the course, most likely before homework and exam dates. In the future we can find a way to consider this aspect. Finally, in our simulation only undergraduates were considered. In the math department many TAs are actually doctoral students, since teaching experience may be part of their curriculum. There may be a preference to give positions to doctoral students. This can either decrease the positions available or add more students into the program. In the future we would like to be able to fix all these limitations.

References

1. Google OR-Tools
https://developers.google.com/optimization/assignment/assignment_min_cost_flow

Appendix

You can use our program by downloading the code and data from github (https://github.com/pawsagainsthoomanity/TA_Assignment_Problem) or copy the code from Appendix AB, then follow the procedures below.

1. Install ortools package (make sure you're using Python3)

```
python -m pip install --upgrade
--user ortools
```

2. Run this command to get cost matrix:

```
python feature_engineering.py
course_schedule.csv
student_schedule.csv
undergrad_preferences.csv
grades.csv cost.csv
```

3. Fill out the file eval.csv as instructed by the output file if you plan to interview candidates.
4. If no interview. Run this command to get TAs assigned (You can replace the cost.csv with your own cost matrix):

```
python assign_ta.py cost.csv
```

5. If interview, run this command instead:

```
python assign_ta_w_interview.py
cost.csv evaluation.csv
```

Check out the output.txt file in your current directory

A: Feature Engineering Code

```
import pandas as pd
import random
import numpy as np
import sys
import csv
import time

def replace_all(text, dic):
    for k, v in dic.items():
        text = text.replace(k, v)
    return text

def cleanPreference(filename, rep):
    data = dict()
    data[’course’] = [’21-111’, ’21-112’, ’21-120’, ’21-122’, ’21-124’, ’21-127’,
                        ’21-228’, ’21-236’, ’21-238’, ’21-240’, ’21-241’, ’21-242’,
                         ’21-254’, ’21-256’, ’21-259’, ’21-260’, ’21-261’, ’21-268’,
                        ’21-269’, ’21-270’, ’21-292’, ’21-369’, ’21-469’]
    course_index = dict()
    for i in range(len(data[’course’])):
        course_name = data[’course’][i]
        course_index[course_name] = i

    with open(filename, ’r’) as csvfile:
        prefs = list(line for line in csvfile)
        prefs.pop(0)
        nrow, ncol = len(prefs), len(data[’course’])
        for i in range(nrow):
            data[str(i+1)] = [0]*ncol

        for i in range(len(prefs)):
            pref = prefs[i].replace(’"’, "").strip().split(’,’)[2:-6]
            selection = [k+1 for k in range(len(pref))] #number of choices
            total_preferences = len(pref)

            for course in pref:
                number = course[:7]
                if number[0] == ’␣’: number = number[1:]
                else: number = number[:6]
                if number in course_index:
                    choice = random.choice(selection)
                    data[str(i+1)][course_index[number]] = choice #student i, course j
                    selection.remove(choice)
            data[str(i+1)] = np.ceil(np.array(data[str(i+1)])*5/total_preferences)
        # print(data[str(69)])
        course_name =  data[’course’]
        for key in data:
            data[key] = pd.Series(data[key]).repeat(rep)

    return pd.DataFrame(data), course_name


#read course schedule file
def getCatalog(filename):
    schedule = pd.read_csv(filename).dropna()
```

```
    course_name = [name for name in schedule.Number]
    course_catalog = dict() #key: course name, val: how many recitations, typically less th
    for i in range(len(schedule.Number)):
        name = schedule.Number.iloc[i]
        name_ = str(name)[0:6]
        if name_ in course_catalog:
            course_catalog[name_] += 1
        else: course_catalog[name_] = 1

    return course_name, course_catalog

def cleanSchedule(filename, nrow, col_start):
    days = [’Monday’, ’Tuesday’, ’Wednesday’, ’Thursday’, ’Friday’]
    schedule = dict()
    for key in days:
        schedule[key] = list()
    with open(filename, ’r’) as file:
        rawdata = list(file)
        rawdata.pop(0)
        rawdata = rawdata[0:nrow]
        rep = {’"’: "", "[": "", "]": "", "␣":""}
        for row in rawdata:
            times = replace_all(row, rep)
            times = times.strip().split(’,’)[col_start:col_start+45]
            for i in range(5):
                sequence = [int(times[int(i+j*5)]) for j in range(9)]
                schedule[days[i]].append(sequence)
        return schedule

def cleanGrades(filename, rep):
    res = list()
    with open(filename, ’r’) as file:
        rawdata = list(file)
        rawdata.pop(0)
        for i in range(len(rawdata)):
            row = rawdata[i]
            vals = row.strip().split(’,’)[1:]
            seq = list()
            for val in vals:
                if int(val) > 3:
                    seq.append(1) #pass if student got more than b
                else:
                    seq.append(0) #disqualified otherwise
            for t in range(rep[i]):
                res.append(seq)
    return res

def main(courseScheduleFile, studentScheduleFile, preferenceFile, gradesFile):

    #get catalog: return course number and index alignment, repition of recitation
    recitation_name, course_vol = getCatalog(courseScheduleFile)
    repeat = [v for k, v in course_vol.items()]

    print("Processing␣schedules␣..␣")
    #clean course schedule, return NxT matrix
    #1 if class takes place, 0 otherwise
    N = len(recitation_name)
```

```
courseSchedule = cleanSchedule(courseScheduleFile, N, 2)

#clean student schedule, return MT matrix
#1 if student is available, 0 otherwise
M = 92
studentSchedule = cleanSchedule(studentScheduleFile, M, 1)

print("Processing grades ... ")
#clean grades, return NxM matrix
grades_arr = cleanGrades(gradesFile, repeat)
grades_matrix = np.array(grades_arr)

print("Processing undergrad preferences ... ")
#clean preference matrix, return dataframe, then return N*M matrix
pref_df, class_name = cleanPreference(preferenceFile, repeat)
# pref_df.to_csv(r'pref.csv', sep = ',', index = False)
pref_matrix = np.array(pref_df.iloc[:, 1:])
cost_mat = np.multiply(pref_matrix, 10)

print("Retrieving eligibility and scores ... ")
#compute availability matrix NxM, 1 if student can lead recitation, 0 otherwise
availability = np.zeros((N, M))
rowtotal = np.zeros(N)
for day in ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']:
    course, student = np.array(courseSchedule[day]), np.array(studentSchedule[day])
    rowsum = np.sum(course, axis = 1)
    rowsum[rowsum>0] = 1 #indicator of whethere or not there's class at all
    rowtotal += rowsum
    avail = np.dot(course, np.transpose(student))
    avail[ avail > 0] = 1
    availability = availability + avail
availability[ availability<1 ] = 0
rowtotal[rowtotal<1] = 1 #prevent division overflow

# TA who apply should be available for both days if there're 2 recitation a week
for col in range(len(availability[0])):
    availability[:, col] = np.divide(availability[:, col], rowtotal)
availability[ availability<1 ] = 0

#21-236, 21-238, 21-270 don't have recitation/ or recitation in the evening
#assume everyone is available for now
availability[15:17, ] = np.ones(92)
availability[-8:-4, ] = np.ones(92)

#apply grades as a filter
eligibility = np.multiply(availability, grades_matrix)

#compute cost
cost = np.multiply(eligibility, cost_mat)
print("Finished!")

#filter unqualified people, collapse vertically
candidates = np.sum(cost, axis = 0)
#filter unpopular class, collapse horizontally
recitations = np.sum(cost, axis = 1)

for i in range(N):
```

```
        for j in range(M):
            if candidates[j] == 0 or recitations[i] == 0:
                cost[i, j] = 999999

    #make dataframe
    out = dict()
    out['Course'] = recitation_name
    for i in range(M):
        out[str(i+1)] = [int(val) for val in cost[:, i]]
    #add dummies
    for i in range(M-N):
        for key in out:
            if key == 'Course': out[key].append('dummy'+str(i+1))
            else: out[key].append(int(999999))

    cost_df = pd.DataFrame(out)
    cost_df.to_csv(r'cost.csv', sep = ',', index = False)

    cost = cost[[0,1,2,3,7,8,13, 15, 16, 17, 18, 21, 22, 23,
     26, 30, 32, 33, 34, 35, 39, 40, 42], :]

    cost[cost == 999999] = 0
    cost[cost > 0] = 1

    interviewee = np.sum(cost, axis = 0) #it's different from candidates!!!
    classes = np.sum(cost, axis = 1) #it's different from recitation!

    outputFile = open("fe_report.txt", "w+")
    outputFile.write("Hi␣Jason,␣we␣are␣unable␣to␣find␣qualified␣or␣available␣candidates␣for
    for i in range(len(classes)):
        if classes[i] == 0:
            outputFile.write("Course␣#" + class_name[i])
            outputFile.write('\n')
    outputFile.write('\n')

    outputFile.write("Additionally,␣\n")
    for i in range(len(classes)):
        print("Found␣%d␣candidates␣for␣course␣%s."% (sum(cost[i]), class_name[i]))
        if classes[i] != 0:
            result = "Found␣%d␣candidates␣for␣course␣%s."% (sum(cost[i]), class_name[i])
            outputFile.write(result)
            outputFile.write('\n')
    outputFile.write('\n')

    eval_form = dict()
    eval_form["student"] = list()
    eval_form['you_preferred_assignment'] = list()

    outputFile.write("Here␣are␣the␣people␣who␣are␣qualified␣to␣teach␣more␣than␣4␣courses.␣\
    for i in range(len(interviewee)):
        if interviewee[i] >4:
            eval_form["student"].append(str(i+1))
            eval_form["you_preferred_assignment"].append("21-XXX")
            outputFile.write("Candidate␣#" +str(i+1)+':␣')
            for j in range(len(cost[:, i])):
                if cost[j, i] >0:
                    outputFile.write(class_name[j]+';␣')
```

```
            outputFile.write('\n')
    pd.DataFrame(eval_form).to_csv(r'evaluation.csv', sep = ',', index = False)

    conclusion = "You␣may␣consider␣an␣interview␣for␣them.␣" +\
            "If␣you␣do,␣please␣fill␣out␣the␣evaluation␣form␣" +\
            "with␣the␣COURSE␣NUMBER␣of␣your␣preferred␣assignment␣" +\
            "in␣the␣column␣next␣to␣student␣ID.␣\n␣Good␣Luck!␣\n"
    outputFile.write(conclusion)
    outputFile.close()

if __name__ == '__main__':
    start_time = time.perf_counter()

    course = sys. argv [1]
    student = sys. argv [2]
    pref = sys. argv [3]
    grades = sys. argv [4]
    main(course, student, pref, grades)
    print("Finished␣in␣", time.perf_counter() - start_time, "seconds")
```

B: Solver Code

```
from __future__ import print_function
from ortools.graph import pywrapgraph
import time
import numpy as np
import sys
import csv

inputArgs = sys.argv
FILENUM = 3
THRESHOLD = 999999

def main():

  dumped_costs = 0
  cost = createDataArray()
  course_dict = createCourseDict()
  rows = len(cost)
  cols = len(cost[0])
  outputFile = open("output.txt", "w+")
  outputFile.write("Hello␣Professor␣Howell!␣Here’s␣your␣TA␣assignment␣result.␣\r\n")

  assignment = pywrapgraph.LinearSumAssignment()
  for ta in range(rows):
    for recitation in range(cols):
      if cost[ta][recitation]:
        assignment.AddArcWithCost(ta, recitation, cost[ta][recitation])

  solve_status = assignment.Solve()

  if solve_status == assignment.OPTIMAL:
    print(’Total␣cost␣=␣’, assignment.OptimalCost())
    for i in range(0, assignment.NumNodes()):
      if assignment.AssignmentCost(i) < THRESHOLD:
        result = ’Applicant␣#%d␣is␣assigned␣to␣Recitation␣%s.␣␣Cost␣=␣%d’ % (
              i+1,
              course_dict[assignment.RightMate(i)],
              assignment.AssignmentCost(i))
      else:
        result = ’Applicant␣#%d␣cannot␣be␣assigned␣to␣any␣recitation.’ %(i+1)
        dumped_costs += 999999
      print(result)
      outputFile.write(result + "\r\n")
    print()
    print(’Total␣cost␣=␣’, (assignment.OptimalCost()-dumped_costs))

  elif solve_status == assignment.INFEASIBLE:
    print(’No␣assignment␣is␣possible.’)
  elif solve_status == assignment.POSSIBLE_OVERFLOW:
    print(’Some␣input␣costs␣are␣too␣large␣and␣may␣cause␣an␣integer␣overflow.’)

  outputFile.close()
  print()
  print("Please␣check␣out␣output.txt␣in␣your␣current␣directory.")

def checkFileExistence(testFile):
```

```
  try:
      openedFile = open(testFile)
      openedFile.close()
  except:
      raise Exception('File␣cannot␣be␣opened.')

def isValidCommand():
  if len(inputArgs) != FILENUM:
    raise Exception("The␣number␣of␣input␣files␣is␣incorrect.␣A␣sample␣command␣looks␣like␣th
    return False
  return True

def convertInput():
  if isValidCommand():
    inputFile = inputArgs[1]
    checkFileExistence(inputFile)
    with open(inputFile, 'r') as csvfile:
      rawCost  = list(csv.reader(csvfile, delimiter=','))
      print("Cost␣matrix␣is␣created.")
      return rawCost

def createCourseDict():
  if isValidCommand():
    inputFile = inputArgs[1]
    checkFileExistence(inputFile)
    res = dict()
    with open(inputFile, 'r') as csvfile:
      rawCost  = list(csv.reader(csvfile, delimiter=','))
      rawCost.pop(0)
      for i in range(len(rawCost)):
        res[i] = rawCost[i][0] #recitation number
      return res

def getEval():
  if isValidCommand():
    inputFile = inputArgs[2]
    checkFileExistence(inputFile)
    with open(inputFile, 'r') as csvfile:
      scores = list(csv.reader(csvfile, delimiter=','))
      scores.pop(0)
      student = [scores[i][0] for i in range(len(scores))]
      course = [scores[i][1] for i in range(len(scores))]
      return student, course

def createDataArray():
  rawCost = convertInput()
  for row in rawCost:
    del row[0]
  rawCost.pop(0)

  ref = createCourseDict()
  student, course = getEval()
  for i in range(len(student)):
    s = student[i]
    s_index = int(s)-1
    c = course[i]
    for k, v in ref.items():
```

```
      if v[:6] == c:
        temp = rawCost[k][s_index]
        rawCost[k][s_index] = float(temp)/10

  transposedCost = [[int(string) for string in inner] for inner in rawCost]
  cost = [[row[i] for row in transposedCost] for i in range(len(transposedCost[0]))]


  return cost

if __name__ == "__main__":
  start_time = time.perf_counter()
  main()
  print("Finished␣in␣", time.perf_counter() - start_time, "seconds")
```