Optimization of Assignments for Teaching Assistants at UW

Chongyi Xu, Weifan Jiang
University of Washington
MATH 381 Project Draft
chongyix@uw.edu

December 4, 2018

Abstract

For this project, we are focusing on developing a method to assign teaching-assistant candidates to different courses. Assigning candidates to their preferred teaching positions is important to course coordinators at UW for a long time. Our purpose is to recommend a method of teaching-assistant assignments to solve this problem. We used simulated data to compare different methods that we would use in the project.

1 Problem Description

The assignment of different teaching positions is a complicated task. The word "teaching position" includes teaching assistants, graders and instructors here at UW. Each type of position has its unique qualifications and requirments. Some positions require teaching, while other do not. Most students are deemed to be certified to teach. Those whose first language is not English must pass the SPEAK test to be certified. The position qualifications do not only appear in different roles but also in different courses. For instance, the instructor positions would mostly be restricted to graduate students or faculty. Meanwhile, the teaching assistant positions could open to both undergraduate students and graduate students.

The UW introduction courses to programming, such as CSE 142(Computer Programming I), are always popular. There are over 700 students registered for around 50 sections each quarter. On the other hand, there are also tiny-sized courses that designed for under 10 students. Therefore, the assignments of teaching positions must satisfy the requirements for every single course each quarter such that everyone who registered for the course could have equal opportunity and fairly distributed teaching resources.

During the process of assignments, the course coordinators, who are in charge of assigning student candidates to appropriate roles, must consider the preference lists submitted by the candidates. Taking the example of UW CSE TA application form, as the candidates apply for CSE TA positions, they needs to choose their preferences ("prefer not" or"neutral" or "prefer") for 12 distinct categories:

- AI and Robotics
- Architecture
- Computational Biology
- Databases, Information Retrieval
- Graphics, Vision, Animation
- Hardware
- Human-Computer Interaction
- Introductory (CSE100,14x, 190)
- Languages, Compilers, Software Engineering
- Systems, Networks
- Theory
- Uncategorize

Following the information provided by CSE department, they initialize the preference value for each course that candidates prefer, are neutral to, or prefer not to TA to 0.8, 0.5, and 0.2, respectively. This helps "push" candidates assignment towards courses in areas they prefer and away from courses in areas they do not prefer. Without choosing preferences directly, candidates could establish their course preferences as well. If candidates choose to make up their own list, they would be asked to fill in a numerical number between 0.0 and 1.0 that represents their preference to teach each courses that they are certified to TA. Be-

sides preferences from the candidates side, instructors preferences should also be considered. Instructors would be asked to fill in a form of preferred students.

Our motivation for the project is from the interview with undergraduate TAs and graduate TAs about their teaching experience in early quarters. (Hongtao Huang, hongth@cs.washington.edu, undergraduate teaching assistant at CSE; Tejas Devanur, tdevanur@uw.edu, graduate teaching fellow at Math Department) They noticed that many times, even though they self-report their preferences, they got assigned to a course which indicated as "less-preferred". Therefore, we would like to recommend a method that assigns candidates to courses, in such way that respects the following considerations:

- Each candidate must be assigned to at most one course.
- Each course must be assigned an appropriate number of candidates.
- Each candidate must be assigned only to the courses for which they are qualified.
- Both candidates' and professors' preferences will be satisfied as much as possible.

2 Simplification

For the project draft, we simulated several groups of small-scale data and compare their results

- 50 candidates with 5 courses
- 100 candidates with 10 courses
- 500 candidates with 20 courses
- 800 candidates with 25 courses
- 1000 candidates with 30 courses

2.1 Input

We simulated student candidates' following a quantified metric:

- Preference for each role (grader, teaching assistant, instructor) for each course as numerical values between 0.0 and 1.0 where 0.0 indicates minimal preference and 1.0 indicates maximal preference. For the project draft, we would only consider teaching assistant roles at first.
- Qualification for each role for each course, which could be represented as an indicating matrix, where 1 entry indicates qualified for the role and 0 indicates not qualified.

We also need to have each professor's preferences towards each candidate, which we implemented as a numerical matrix like preference matrix of candidates' preference.
As for the department's requirements, we chose a number between 1 and 5 for each course that representing the required number of teaching assistants for corresponding course.

2.2 Assumption

There were some assumptions we considered about in the draft phase

1. The quantifications of candidates who are applying for the same course were the same. In the other word, we would not take account into candidates' past teaching experiences, as well as their GPA when they took that course at this time.
2. Student candidates do not care about any factors other than their preferences, such as payment and work time.
3. Candidate's time conflict with other courses they are taking is not considered.
4. All candidates are legally registered UW students.

3 Mathematical Model

Let $X = x_1, ..., x_m$ represent m student candidates, let $Y = y_1, ..., y_n$ represent n courses.
Let c_j represent the number of teaching assistants required for course y_j for $1 \leq j \leq n$.
Let

$$q_{ij} = \begin{cases} 1, \text{ if } x_i \text{ is qualified to teach } y_j \\ 0, \text{ otherwise} \end{cases}$$

The goal is to produce a **stable assignment** of candi-

dates to courses, represented as

$$a_{ij} = \begin{cases} 1, \text{ if } x_i \text{ is assigned to } y_j \\ 0, \text{ otherwise} \end{cases}$$

, subject to the following hard constraints:

1. Each candidate must be assigned to at most one course:
$$\forall x_i \in X, \sum_{j=1}^{n} a_{ij} = 1$$
.
2. Each course must be assigned an appropriate number of candidates:
$$\forall y_j \in Y, \sum_{i=1}^{m} a_{ij} = c_j$$
.
3. Each candidate must be assigned only to the courses for which they are qualified:
$$\forall x_j \in X \forall y_j \in Y q_{ij} \geq a_{ij}$$
.

4 Solution of the Mathematical Problem

4.1 Stable Marriage Algorithm

In the field of computer science and mathematics, the stable match problem or stable marraige problem states that given N men and N women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no 2 people of opposite sex who would both rather have each other than their current partners. If there are no such people, all the marriages are “stable”.

In 1962, D. Gale and L. S. Shapley, proved that, for any equal number of men and women, it is guaranteed that there is a stable matching. In their paper "College Admission and the Stability of Marriage", they defined the stability as following, an assignment of applications to colleges will be called unstable if there are two applicants α and β who are assigned to colleges A and B, respectively, although β prefers A to B and A prefers β to α. They considered a stable assignment to be optimal if every applicant is at least as well off under it as under any other stable assignments.

```
*Gale-Shapley Algorithm*
INPUT: preference list for men and
women
INITIALIZE matching set S to an empty
set
WHILE (some woman w in W is still
    unmatched and hasn't proposed
    to every man in M)
    m <- first man on w's preference
        list to whom w has not yet
        proposed
    IF (m is unmatched)
        ADD pair (m, w) to S
    ELSE IF (m prefers w to existing
            pair w')
        REPLACE (m, w') with (m, w)
        FREE w'
    ELSE
        w REJECT m
RETURN: matching S
```

In this project, we have to slightly modify the algorithm in order to achieve our goal. Since each course may have need of more than one candidate to be assigned, such changes will be made to the original Gale-Shapley Algorithm:

1. During each round of proposing, a currently unmatched candidate proposes to his/her top-choice course which he/she has not proposed to yet.
2. After candidates finish proposing to courses, each course takes the new proposers, put them into the same "set" with other candidates that are already matched with this course, to form a "temporary" waitlist.
3. If the waitlist’s length exceeds the course capacity, the waitlist will be sorted by course’s preference to waitlist’s members, and only the top k ones will be kept, with k being the capacity of that course.
4. The algorithm terminates when there are no un-

matched candidates or all candidates have proposed to all courses.

By using `Python`, we implemented the following functionalities. By inputting a `num_candidate` and a `num_course` parameters, our program can:

1. Generate m candidates and n courses, which `num_candidate = m` and `num_course = n`.
2. Generate random preference levels of each candidate for courses, and each course to candidates, which are floats between 0 and 1 inclusively.
3. Generate capacity for each course, which is an integer between 1 and 5 inclusively.
4. Generate qualification for each student for each course, which is an integer either 0 or 1, with 1 indicates being qualified.
5. Use the modified algorithm mentioned above to generate an assignment.

4.2 Hungarian Algorithm

The Hungarian Algorithm is a combinatorial optimization algorithm that solves the assignment problem in polynomial time. It was developed and published in 1955 by Harold Kuhn, who gave "Hungarian Algorithm" its name according to the previous works of two Hungarian mathematicians.

```
*Hungarian Algorithm*
INPUT: n*n cost matrix A
FOR EACH (row R_A in A)
    SUBSTRACT min(R_A) from R_A
FOR EACH (column C_A in A)
    SUBSTRACT min(C_A) from C_A
LABEL appropriate entries so that all
      zero entries are covered and
      minimum number of labels are
      used
IF (# labels = n)
    RETURN: labels as assignment
ELSE:
    SUBSTRACT min(A) from unlabeled
              R_A
    ADD min(A) to unlabeled C_A
    REPEAT from LABEL
```

4.3 Maximum Matching Algorithm

Consider an undirected graph $G = (V, E)$. A matching M is said to be maximal if M is not properly contained in any other matching. Formally, $M \notin M^{'}$ for any matching $M^{'}$ of G. Intuitively, this is equivalent to saying that a matching is maximal if we cannot add any edge to the existing set. And a matching M is said to be Maximum if for any other matching $M^{'}$, $|M| \geq |M^{'}|$. Generally, maximum matching applied to unweighted graph more but for this project, we would like to modify the algorithm with weights in order to meet our propose. With researching, we decided to implement the method introduced by *Zvi Galil*, Department of Computer Science, Columbia University, in 1986. In his study "Efficient Algorithms for Maximum Matching in Graphs", he developed this method based on Berge's Theorem, "the matching M has maximum cardinality if and only if there is no augmenting path with respect to M."

Given a graph, $G = (V, E)$ and a matching $M \subset E$, a path P is called an augmenting path for M if:

- The two end points of P are unmatched by M
- The edges of P alternate between edges $\in M$ and edges $\notin M$.

```
*Maximum Matching*
INPUT: Graph G
M <- random selected matching
WHILE (there is a blossom and there
       is an augmenting path in M)
    GROW the forest, labeling the
         vertices even/odd
    IF (there is a blossom in the
        graph)
        SHRINK the blossom to obtain
               a new graph G'
        CONTINUE foresting
    ELSE
        FIND such even - even edges
             to obtain a maximally
             disjoint set of
             augmenting paths
             (P1,...,Pk)
    M <- switching edges along P's
         from in-to-out of M and
         vice-versa
```

```
EXPAND all blossoms to obtain the
       maximum matching in the
       original graph G.
```

5 Results

In a simple run of our program, suppose we want 10 candidates and 2 courses, we will use the following parameters to the Python script:

```
python stable_marrige.py --num_candidate
10 --num_course 2
```

which will generate input data and produce an assignment under the assumption of 10 candidates and 2 courses. Due to random data generation, the output of each trial may differ. Here's one sample trial's output with parameters described above:

```
candidates preference to courses:
candidate      course0      course1
candidate0  Unqualified          0.2
candidate1          0.1          0.6
candidate2          0.2          0.4
candidate3          0.6            0
candidate4          0.6          0.3
candidate5  Unqualified          0.9
candidate6          0.2  Unqualified
candidate7          0.1  Unqualified
candidate8  Unqualified          0.9
candidate9          0.3  Unqualified

courses capacity and preference to
candidates:
course capacity   candidate5   candidate8
course0        4  Unqualified  Unqualified
course1        4          0.4          0.1

candidate3  candidate2  candidate0
       0.3         0.4  Unqualified
       0.6         0.6           0

candidate9  candidate4   candidate7
       0.5         0.4          0.9
Unqualified        0.7  Unqualified

candidate6  candidate1
0.1                0.7
Unqualified        0.2

Final TA assignment:
course0: candidate7, candidate1, candidate9,
candidate4
course1: candidate8, candidate2, candidate3,
candidate5
```

The portion of output under the line `Final TA assignment:` shows the produced assignment of TAs: for each course, the candidates assigned to that course are behind the : sign. In other words:

- candidate 7, 1, 9, 4 are assigned to course 0.
- candidate 8, 2, 3, 5 are assigned to course 1.
- candidate 0, 6 are unselected for the TA role.

In such way, we have our input data as we listed in the previous section and generate our assignment output

5.1 Group 1: 50 candidates and 5 courses

- candidates 25, 1, 35, 49, 28 are assigned to course0
- candidates 16, 14, 44 are assigned to course1
- candidates 21, 45 are assigned to course2
- candidates 46, 8 are assigned to course3
- candidates 1 are assigned to course4

5.2 Group 2: 100 candidates and 10 courses

- candidates 25, 70, 4 are assigned to course0
- candidates 10 are assigned to course1
- candidates 75, 3 are assigned to course2
- candidates 56, 62, 5, 85, 20 are assigned to course3
- candidates 72, 23 are assigned to course4
- candidates 45 are assigned to course5
- candidates 27, 1, 54, 19 are assigned to course6
- candidates 54, 33, 94, 64 are assigned to course7
- candidates 26, 42, 64, 5 are assigned to course8
- candidates 12 are assigned to course9

5.3 Group 3: 500 candidates and 20 courses

- candidates 164 are assigned to course0
- candidates 387, 482, 215, 405 are assigned to course1
- candidates 144, 450, 296, 479 are assigned to course2
- candidates 447, 422, 379, 286 are assigned to course3
- candidates 117, 125 are assigned to course4
- candidates 259, 29, 499, 404 are assigned to course5
- candidates 76, 177, 126 are assigned to course6
- candidates 310, 16, 436 are assigned to course7
- candidates 271, 425, 36, 314, 329 are assigned to course8
- candidates 217 are assigned to course9
- candidates 188, 49, 234 are assigned to course10
- candidates 213 are assigned to course11
- candidates 94, 494 are assigned to course12
- candidates 312, 406, 257 are assigned to course13
- candidates 161, 111, 135 are assigned to course14
- candidates 486, 90, 428, 98, 382 are assigned to course15
- candidates 478 are assigned to course16
- candidates 33 are assigned to course17
- candidates 395, 284, 365 are assigned to course18
- candidates 180 are assigned to course19

5.4 Group 4: 800 candidates and 25 courses

- candidates 592, 86, 156, 769, 350 are assigned to course0
- candidates 436, 784 are assigned to course1
- candidates 30, 752, 493, 706 are assigned to course2
- candidates 56, 623, 628, 646, 323 are assigned to course3
- candidates 560, 670, 390, 57 are assigned to course4
- candidates 31 are assigned to course5
- candidates 336, 33, 532 are assigned to course6
- candidates 731, 618 are assigned to course7
- candidates 315 are assigned to course8
- candidates 50, 287, 471, 213 are assigned to course9
- candidates 603, 565, 168 are assigned to course10
- candidates 182, 491 are assigned to course11
- candidates 498, 378, 5, 760 are assigned to course12
- candidates 630, 201 are assigned to course13
- candidates 279 are assigned to course14
- candidates 310 are assigned to course15
- candidates 568 are assigned to course16
- candidates 659, 16, 487 are assigned to course17
- candidates 420, 531 are assigned to course18
- candidates 395, 497, 588 are assigned to course19
- candidates 128, 236 are assigned to course20
- candidates 474 are assigned to course21
- candidates 4, 87 are assigned to course22
- candidates 578 are assigned to course23
- candidates 533, 720 are assigned to course24

5.5 Group 5: 1000 candidates and 30 courses

- candidates 462, 207 are assigned to course0
- candidates 876, 797, 908, 145 are assigned to course1
- candidates 857 are assigned to course2
- candidates 723 are assigned to course3
- candidates 44, 459, 725, 861, 485 are assigned to course4
- candidates 96, 345, 74, 496, 766 are assigned to course5
- candidates 506, 935 are assigned to course6
- candidates 671, 310, 809, 736, 979 are assigned to course7
- candidates 321, 295 are assigned to course8
- candidates 637, 594, 506 are assigned to course9
- candidates 71, 994, 688 are assigned to course10
- candidates 895, 717, 619, 626, 454 are assigned to course11
- candidates 775, 685, 851 are assigned to course12
- candidates 273, 922 are assigned to course13
- candidates 436, 411 are assigned to course14
- candidates 419 are assigned to course15
- candidates 859, 919, 904 are assigned to course16
- candidates 609, 342 are assigned to course17
- candidates 111, 265, 746, 307 are assigned to course18
- candidates 435, 671 are assigned to course19
- candidates 348, 928, 776 are assigned to course20
- candidates 189, 472 are assigned to course21

- candidates 837, 73, 743 are assigned to course22
- candidates 49, 976, 953 are assigned to course23
- candidates 634, 834, 219 are assigned to course24
- candidates 753 are assigned to course25
- candidates 608 are assigned to course26
- candidates 226 are assigned to course27
- candidates 361, 49, 775 are assigned to course28
- candidates 731, 246, 961 are assigned to course29

6 Improvements

In addition to stable marrige, we will develop other two methods: Maximal Matching Algorithm and Hungarian Algorithm with simulated data and make a comparison among three methods to realize their differences not only in output but also in the process of implementation and interpretation. Also, we are currently ignore candidates' GPA and past teaching experiences, which might be an important factor in decision making.

Additionally, we are currently only focusing on teaching assistant posistions, but the real world problem includes more various teaching positions such as graders and instructors. We would also like to consider various positions for further studies.

Another point to improve our method is to find out some other constraints to let our model fit the real world problem more. For example, we could add the constraints considering about the time confliction for candidate's course schedule.

In such way, I think our model could be more realistic and might be more accpetable to our community partners.

7 Conclusion

From the output, we could draw our conclusion that our method can build an appropriate TA assignment that satisfies candidates' preferences and Instructors' preferences as much as possible, while all requirements are met.

8 References

[1]"The college admission problem: many-to-one matching : Networks II Course blog for INFO 4220", Blogs.cornell.edu, 2018. [Online]. Available: https://blogs.cornell.edu/info4220/2016/03/18/the-college-admission-problem-many-to-one-matching/.

[2]D. Gale and L. Shapley, "College Admissions and the Stability of Marriage", The American Mathematical Monthly, vol. 69, no. 1, p. 9, 1962.

[3]"Stable Marriage Problem – from Wolfram MathWorld", Mathworld.wolfram.com, 2018. http://mathworld.wolfram.com/StableMarriageProblem.html.

[4]Cs.princeton.edu, 2018. [Online]. https://www.cs.princeton.edu/ wayne/kleinberg-tardos/pdf/01StableMatching.pdf.

```
import data_generator
import argparse
import pandas as pd

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

def run_stable_marrige(num_candidate, num_course, output):
    data = data_generator.generate(num_candidate, num_course)
    terminated = False
    applied_to = dict()
    unmatched = set()
    for candidate in data.candidates:
        applied_to[candidate] = set()
        unmatched.add(candidate)
    qualified = dict()
    for candidate, qualification in data.qualification.items():
        for course, score in qualification.items():
            if score == 1:
                courses = qualified.get(candidate, set())
                courses.add(course)
                qualified[candidate] = courses
    course_list = list(data.courses)
    current_match = dict()
    for course in data.courses:
        current_match[course] = list()
    while not terminated:
        curr_purpose = dict()
        for candidate in unmatched:
            courses_sorted = sorted(course_list, key = lambda x: data.candidate_preference[candidate][x]
            for course in courses_sorted:
                if course not in applied_to[candidate] and course in qualified[candidate]:
                    applied_to[candidate].add(course)
                    curr_purpose[candidate] = course
                    break
        for candidate, course in curr_purpose.items():
            current_match[course].append(candidate)
        new_match = dict()
        curr_matched_candidates = set()
        for course, candidates in current_match.items():
            new_candidates = sorted(candidates, key = lambda x: data.course_preference[course][x], reve
            if len(new_candidates) > data.course_capacity[course]:
                new_candidates = new_candidates[0:data.course_capacity[course]]
            new_match[course] = new_candidates
            for c in new_candidates:
                curr_matched_candidates.add(c)
```

```
        current_match = new_match.copy()
        for candidate in data.candidates:
            if candidate not in curr_matched_candidates:
                unmatched.add(candidate)
        terminated = True
        for candidate in unmatched:
            if len(applied_to[candidate]) < len(qualified[candidate]):
                terminated = False
    write_to_file(data, current_match, output)

def write_to_file(data, matching, output):
    output_file = open(output + "", 'w')

    output_file.write("candidates preference to courses:\n")
    column_list = list(data.courses)
    column_list.insert(0, 'candidate')
    candidate_preference = pd.DataFrame(columns = column_list)
    for candidate, preference in data.candidate_preference.items():
        new_data = preference.copy()
        for course, qualification_score in data.qualification[candidate].items():
            if qualification_score == 0:
                new_data[course] = "Unqualified"
        new_data['candidate'] = candidate
        candidate_preference = candidate_preference.append(new_data, ignore_index = True)
    output_file.write(candidate_preference.to_string(index=False) + "\n")

    output_file.write("\ncourses capacity and preference to candidates:\n")
    column_list = list(data.candidates)
    column_list.insert(0, 'course')
    column_list.insert(1, 'capacity')
    course_preference = pd.DataFrame(columns = column_list)
    for course, preference in data.course_preference.items():
        new_data = preference.copy()
        for candidate in new_data.keys():
            if data.qualification[candidate][course] == 0:
                new_data[candidate] = "Unqualified"
        new_data['course'] = course
        new_data['capacity'] = data.course_capacity[course]
        course_preference = course_preference.append(new_data, ignore_index = True)
    output_file.write(course_preference.to_string(index=False) + "\n")
    output_file.write("\n")

    output_file.write("Final TA assignment:\n")
    for course, TAs in matching.items():
        new_data = dict()
        people = ""
        if len(TAs) == 1:
```

```
            people = TAs[0]
        else:
            people = TAs[0]
            for i in range(1, len(TAs)):
                people += ", " + TAs[i]
        new_data['assigned candidates'] = people
        output_file.write(course + ": " + people + "\n")

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--num_candidate', type=int)
    parser.add_argument('--num_course', type=int)
    parser.add_argument('--output')
    args = parser.parse_args()

    run_stable_marrige(args.num_candidate, args.num_course, args.output)

if __name__ == "__main__":
    main()
```